NTNU
Norwegian University of Science and Technology

Procedures and Tools to Reset or Recover the Administrator Password on Popular Operating Systems

Jørgen Wahl Blakstad
Rune Walsø Nergård

Master of Science in Communication Technology

Submission date:	June 2009
Supervisor:	Danilo Gligoroski, ITEM
Co-supervisor:	Martin Gilje Jaatun, SINTEF

Norwegian University of Science and Technology
Department of Telematics

Problem Description

It is a fact that a lot of people forget their computers in public places such as airports and taxis. It may lead to serious consequences, both for business and private persons, if the information ends up in the wrong hands. Many people think that when a laptop with sensitive or confidential information is lost, their login password protection is adequate. Although the system is password protected, this thesis will show that this is not sufficient protection.

This thesis consists of three parts. In the first part we will present an overview of the login mechanism and password handling in the following operating systems: Windows XP, Windows Vista, Windows Server 2008, Windows 7, Ubuntu Server Edition 8.10, Fedora 10, FreeBSD 7.1 and Macintosh OSX 10.5.6.

In the second part we will describe a laboratory environment consisting of eight computers with the selected operating systems, in which we experimentally will apply different procedures and tools to reset or recover the administrator password.

The third part presents an empirical password study where we test the strength of 30 passwords, and see how many we are able to reveal during a period of 8 hours.

Assignment given: 15. January 2009
Supervisor: Danilo Gligoroski, ITEM

Abstract

Unauthorized access to computers and theft of proprietary information are two problems leading to large economical losses for organizations around the world. Thousands of laptops often containing vital information are lost at airports every day. Organizations and people in general often believe that the sensitive information is inaccessible because of the login mechanism. Even though we demand that our systems should prevent unauthorized access, we also expect that the access to the operating system can be restored when a password is lost. We require that *authorized* persons can regain the access to the computer, while *unauthorized* persons are prevented access. A good solution to reset or recover the Administrator password should exist on all operating systems.

This thesis begins with addressing weaknesses in 8 different operating systems. It presents a comprehensive step-by-step guide for already existing procedures and tools that can be used to reset or recover the Administrator password. In total 6 procedures and 10 tools are presented. Because some procedures required a lot of interaction from the user, we decided to automate these and include them in our self made tool named *Yet Another Local Password (tool) (YALP)*.

We were able to reset the passwords on all of the 8 operating systems. On some of the operating systems only a few passwords were recovered, and based on that, a more comprehensive password recovery study was desirable. It should be noted that even though Microsoft has introduced a more secure password handling mechanism on newer Windows operating system, many persons and corporations still use the outdated Windows XP operating system. This is partly because Windows Vista has been criticized for its weak performance. An empirical password study was carried out to see what percentage of 30 carefully chosen passwords could be revealed. Some disturbing results were obtained. During a period of 8 hours, 100% of the passwords created on a Windows XP system were revealed. The results from this study show that the use of password as an authentication mechanism for operating systems will not offer sufficient protection in the future, and that other mechanisms have to be considered.

Based on results obtained from this thesis, a paper named *All in a day's work: Password cracking for the rest of us* was submitted to The 14th Nordic Conference on Secure IT Systems, NordSec 2009, in Oslo, Norway [1]. In addition, a poster named *Generation of Rainbow Tables* was presented at The 8th Annual Meeting on High Performance Computing and Infrastructure, NOTUR2009, in Trondheim, Norway [2]. The poster and the paper are included in Appendix F and Appendix G, respectively.

Preface

The work on this thesis has been carried out in the 10th semester of the Master's Programme in Communication Technology at The Norwegian University of Science and Technology (NTNU), Department of Telematics. The topic for this thesis was formed in cooperation with Danilo Gligoroski at the Department of Telematics, and Martin Gilje Jaatun from SINTEF ICT. The experimental parts of this thesis were carried out at a laboratory located at NTNU.

Working on this thesis has been a challenging and informative task. Through our study on how to reset or recover a lost password the authors of this thesis have gained a lot of experience with different operating systems and their underlying login mechanisms. The authors have also gained valuable knowledge regarding the choice of a secure password.

We would like to thank our supervisor Martin Gilje Jaatun for his enthusiasm and valuable input throughout this thesis. He has constantly been a driving force and has devoted much time in order to give us constructive feedback. We would also like to thank our professor Danilo Gligoroski for his motivating comments and contributing ideas. We are also grateful for his assistance with the poster presented at NOTUR2009. We also appreciate their contribution with the scientific paper based on our thesis submitted to the NordSec 2009 conference.

Additionally, Jørgen would like to thank his beloved Birgitte Blaauw Nordal for her moral support and understanding during late work hours in the final stage of his master's degree.

Contents

Abstract **i**

Preface **iii**

List of Figures **xi**

List of Tables **xiii**

List of Listings **xv**

Abbreviations **xvii**

Definitions **xix**

1 Introduction **1**

1.1 Motivation 2
1.2 Related Work 3
1.3 Objectives 3
1.4 Limitations 4
1.5 Methodology 4
1.6 Document Structure 5
1.7 Conventions 6

2 Background **9**

2.1 Techniques 9
2.1.1 How to Choose a Password 9
2.1.2 Password Recovery Techniques 12
2.1.2.1 Password Guessing 12
2.1.2.2 Password Cracking 13
2.1.2.3 Rainbow Tables 14
2.2 Boot Process 18

2.2.1 Windows 18
2.2.2 Linux 24
2.2.3 BSD 25
2.2.3.1 FreeBSD 26
2.2.3.2 Mac OS X 28
2.3 Password Handling 29
2.3.1 Windows 30
2.3.1.1 SAM 30
2.3.1.2 Hashing of passwords 31
2.3.2 Linux 43
2.3.2.1 The passwd file 43
2.3.2.2 The shadow suite 44
2.3.3 BSD 45
2.3.3.1 FreeBSD 45
2.3.3.2 Mac OS X 46
2.4 Tools 49
2.4.1 Password Resetting 49
2.4.1.1 Offline NT Password & Registry Editor 49
2.4.1.2 Kon-Boot 50
2.4.1.3 Bart's Preinstalled Environment (BartPE) 51
2.4.1.4 PCLoginNow 2.1.0 53
2.4.1.5 DreamPackPL 54
2.4.1.6 The *passwd* Tool 55
2.4.2 Password Recovery 55
2.4.2.1 Ophcrack 55
2.4.2.2 John the Ripper 57
2.4.2.3 Cain & Abel 58
2.4.2.4 LCP 59
2.4.3 Supplementary 60
2.4.3.1 The fgdump Tool 60
2.4.3.2 Winrtgen 60

3 Laboratory **63**
3.1 Hardware 63
3.2 Operating Systems 64

4 Method **65**
4.1 Overview of Procedures and Tools 67
4.1.1 Password Resetting 67
4.1.1.1 Procedures for Windows Operating Systems 67
4.1.1.2 Tools for Windows Operating Systems 67
4.1.1.3 Procedures for Unix-based Operating Systems 68
4.1.1.4 Tools for Unix-based Operating Systems 68
4.1.2 Password Recovery 69

4.1.2.1 Tools for Windows Operating Systems 69
4.1.2.2 Tools for Unix-based Operating Systems 70
4.2 Overview of Supplementary Procedures and Tools 70
4.2.1 Procedures for Windows Operating Systems 70
4.2.2 Tools for Windows Operating System 71
4.2.3 Procedures for Unix-based Operating Systems 71
4.2.4 Tools for Unix-based Operating System 71

5 Procedures 73
5.1 Procedures for Password Resetting 73
Procedure 1: Windows XP Repair Backdoor 73
Procedure 2: Edit Shadow File in Unix Systems 76
Procedure 3: Shortcut for Bypassing Regular Login Procedure 77
Procedure 4: Replace a Pre-login Executable File 78
Procedure 5: Use Mac OS X Installation Disk 81
Procedure 6: Windows Password Reset Disk 82
5.2 Supplementary Procedures 84
Procedure 7: Edit Boot Parameters in Unix Systems 84
Procedure 7.1: Edit Boot Parameter: single 84
Procedure 7.2: Edit Boot Parameter: *init=/bin/bash* 85
Procedure 8: Reboot and Hold Apple+S 87
Procedure 9: Enter Single-user Mode Through Boot-menu 87
Procedure 10: Using Live CD to Obtain Password File 89
Procedure 11: Using Live CD to Change Root Into the Host Operating System 91

6 Tools 95
6.1 Tools for Password Resetting 95
Tool 1: PCLoginNow 2.0.1 95
Tool 2: Offline NT Password & Registry Editor 97
Tool 3: BartPE 99
Tool 3.1: Password Renew 100
Tool 3.2: WindowsGate 102
Tool 4: Kon-Boot 104
Tool 5: DreamPackPL 106
Tool 6: passwd 109
6.2 Tools for Password Recovery 110
Tool 7: Ophcrack Live CD 2.1.0 110
Tool 8: John the Ripper 1.7.2 112
Tool 9: Cain & Abel v4.9.30 115
Tool 10: LCP 5.04 117
6.3 Supplementary Tools 119
Tool 11: fgdump 2.1.0 119
Tool 12: Winrtgen 2.8 120

7 Results **123**
7.1 Procedures and Tools 123
7.1.1 Password Resetting 124
7.1.2 Password Recovery 125
7.2 YALP 126
7.2.1 Description of Yet Another Local Password (tool) (YALP) 126
7.2.2 Contents of YALP 127

8 Empirical Password Study **133**
8.1 Laboratory Environment 133
8.2 Line of Action 135
8.2.1 Selection of Passwords 136
8.2.2 Phase 1: Timing 138
8.2.3 Phase 2: Password Cracking 139
8.3 Results of The Study 140
8.3.1 Phase 1: Timing 140
8.3.1.1 LM hash 140
8.3.1.2 NT hash 144
8.3.1.3 MD5 hash 147
8.3.2 Phase 2: Password Cracking 149
8.3.2.1 LM hash 149
8.3.2.2 NT hash 156
8.3.2.3 MD5 hash 162

9 Discussion **167**
9.1 Procedures and Tools 167
9.1.1 Evaluation of Procedures 168
9.1.2 Evaluation of Tools 169
9.1.3 General Discussion 171
9.2 Empirical Password Study 173
9.2.1 Time Measurements for LM and NT 173
9.2.2 Password Cracking of LM and NT 177
9.2.3 Time Measurements and Password Cracking of MD5 182
9.2.4 Discussion of General Results 183
9.3 The YALP Tool 185
9.4 Implications of Poor Login Security 186
9.4.1 Implications of a Weak Login Mechanism 187
9.4.2 Implications of a Weak Password 187
9.5 Possible Solutions to Prevent Unauthorized Access 188
9.5.1 Choosing a Secure Password 188
9.5.2 A More Secure Password Handling 190
9.5.3 Protection Mechanisms 193
9.6 Windows versus Unix 194
9.7 Lessons Learned 195

9.7.1 Laboratory Equipment . . . 195
9.7.2 Selection of passwords . . . 196
9.7.3 Time measurements . . . 196
9.8 Further Work . . . 196
9.8.1 Disk Encryption . . . 196
9.8.2 Use of Supercomputers . . . 197
9.8.3 YALP . . . 198
9.8.4 Other Hardware and Operating Systems . . . 198
9.8.5 Other Exploitable Weaknesses . . . 198
9.8.6 Other Authentication Mechanisms . . . 199
9.8.7 Resetting and Recovery of Passwords Over Networks . . . 199

10 Conclusion 201

Bibliography 205

Web References 209

A How to Build YALP 219
A.1 build_YALP.sh . . . 219
A.2 build_kernel.sh . . . 221

B YALP Scripts 223
B.1 startup.sh . . . 223
B.2 replace_startup_files.sh . . . 228
B.3 editshadow.pl . . . 231
B.4 mountos.sh . . . 233

C How to Disable LM Hashing 237

D How to Add 30 Users 239
D.1 addusers_win.pl . . . 239
D.2 addusers_lin.pl . . . 240

E Attachments 241
E.1 Attached CD . . . 241
E.2 Electronic Attachments . . . 241

F Poster 243

G Paper 247

List of Figures

1.1 The market share of different operating systems. 2

2.1 The lookup process in rainbow tables. 17
2.2 *pwdump* password entry format. 30
2.3 A simplified overview of DES. 32
2.4 The construction of the LM hash. 34
2.5 The construction of the NT hash. 38
2.6 DES obfuscation of NT hash. 40
2.7 RC4 encryption of the obfuscated NT hash. 42
2.8 Offline NT Password & Registry Editor welcome screen. 49
2.9 Kon-Boot almost finished with the kernel hack. 51
2.10 Password Renew. 53
2.11 Windows Gate. 53
2.12 PCLoginNow's welcome screen. 54
2.13 DreamPackPL initiated from a Windows XP login screen. 55
2.14 LCP main window. 60

7.1 YALP initial welcome screen. 127
7.2 A flow chart displaying YALP's startup script. 128
7.3 YALP's main menu. 128
7.4 Password resetting tools menu. 129
7.5 Password recovery tools menu. 129
7.6 Supplementary tools menu. 130

8.1 Phase 1: Settings used for the dictionary attack (LM hash). 141
8.2 Phase 1: Timing of the dictionary attack on the LM hash. 142
8.3 Phase 1: Timing of the cryptanalysis attack on the LM hash. 143
8.4 Distribution of time between the techniques applied to the LM hash. . . . 144
8.5 Phase 1: Settings used for the dictionary attack (NT hash). 145
8.6 Timing of the cryptanalysis attack on the NT hash. 146

8.7 Distribution of time between the techniques applied to the NT hash. . . . 147
8.8 Result from the dictionary attack on the MD5 hash. 148
8.9 Distribution of used time for the dictionary attack on the MD5 hash. . . . 149
8.10 Phase 2: Settings used for the dictionary attack (LM hash). 150
8.11 Passwords revealed from the dictionary attack (LM hash). 151
8.12 Share of revealed passwords. 152
8.13 Passwords revealed from the cryptanalysis attack (LM hash). 153
8.14 Share of revealed passwords. 153
8.15 Phase 2: Timing of the cryptanalysis attack on the LM hashes. 155
8.16 Distribution of time between the techniques applied to the LM hashes. . . 156
8.17 Phase 2: Settings used for the dictionary attack (NT hash). 157
8.18 Passwords revealed from the dictionary attack (NT hash). 158
8.19 Share of revealed passwords. 159
8.20 Passwords revealed from the cryptanalysis attack (NT hash). 159
8.21 Share of revealed passwords. 161
8.22 Distribution of time between the techniques applied to the NT hashes. . . 162
8.23 Passwords revealed from the dictionary attack (MD5 hash). 163
8.24 A list of the revealed passwords sorted by username. 164
8.25 Share of revealed passwords. 165

9.1 Results from the empirical password study. 173

List of Tables

2.1 The different runlevels initiated by *init*. 26
2.2 Windows operating systems that stores LM hash or/and NT hash by default. 39
2.3 Id values available for the encrypted password in *shadow*. 45
2.4 Operating systems supported by the Kon-Boot tool. 50

3.1 Computer hardware specifications. 64
3.2 Operating system overview. 64

4.1 The names of the various procedures. 66
4.2 The names of the various tools. 66
4.3 Password resetting procedures for different Windows operating systems. . 67
4.4 Password resetting tools for different Windows operating systems. 68
4.5 Password resetting procedures for Unix-based operating systems. 68
4.6 Password resetting tools for Unix-based operating systems. 69
4.7 Password recovery tools for different Windows operating systems. 69
4.8 Password recovery tools for Unix-based operating systems. 70
4.9 Supplementary procedures for different Windows operating systems. . . . 70
4.10 Supplementary tools different Windows operating systems. 71
4.11 Supplementary procedures for Unix-based operating systems. 71
4.12 Supplementary tools for different Unix-based operating systems. 72

5.1 Procedure 1 *Windows XP Repair Backdoor*. 73
5.2 Procedure 2 *Edit Shadow File in Unix Systems*. 76
5.3 Procedure 3 *Shortcut for bypassing regular login procedure*. 77
5.4 Procedure 4 *Replace a pre-login executable file*. 78
5.5 Procedure 5 *Use Mac OS X Installation Disk*. 81
5.6 Procedure 6 *Windows Password Reset Disk*. 82
5.7 Procedure 7.1 *Edit Boot Parameter: single*. 84
5.8 Procedure 7.2 *Edit Boot Parameter: init=/bin/bash*. 85
5.9 Procedure 8: *Reboot and hold Apple+S*. 87

5.10 Procedure 9: *Enter Single-user Mode Through Boot-menu.* 87
5.11 Procedure 10 *Using Live CD to Obtain password file.* 89
5.12 Procedure 11 *Using Live CD to Change Root Into The Host Operating System.* 91

6.1 Tool 1 *PCLoginNow 2.0.1.* 95
6.2 Tool 2 *Offline NT Password & Registry Editor.* 97
6.3 Tool 3.0 *BartPE.* 99
6.4 Tool 3.1 *Password Renew.* 100
6.5 Tool 3.2 *WindowsGate.* 102
6.6 Tool 4 *Kon-Boot.* 104
6.7 Tool 5 *DreamPackPL.* 106
6.8 Tool 6 *passwd.* 109
6.9 Tool 7 *Ophcrack Live CD 2.1.0.* 110
6.10 Tool 8 *John the Ripper 1.7.2.* 112
6.11 Tool 9 *Cain & Abel v4.9.29.* 115
6.12 Tool 10 *LCP 5.04.* 117
6.13 Tool 11 *fgdump 2.1.0.* 119
6.14 Tool 12 *Winrtgen 2.8.* 120

7.1 Key statistics from the password resetting procedures. 124
7.2 Key statistics from the password resetting tools. 124
7.3 Key statistics from the password recovery tools. 125

8.1 The operating systems used in Phase 1 and Phase 2. 134
8.2 Applied techniques corresponding to their type of hash. 136
8.3 An overview of the 30 selected passwords, and their corresponding category. 137
8.4 10 mnemonic passwords with their corresponding phrase. 137
8.5 An overview of the tools used in Phase 1 and Phase 2. 139
8.6 Phase 1: Time measurements for the LM hash. 143
8.7 Phase 1: Time measurements for the NT hash. 146
8.8 Time measurement for the MD5 hash. 148
8.9 Revealed passwords. 152
8.10 Revealed passwords. 153
8.11 The results from both the attacks (LM hash). 154
8.12 Phase 2: Time measurements for the LM hashes. 155
8.13 Revealed passwords. 159
8.14 The results from both the attacks (NT hash). 160
8.15 Revealed passwords. 161
8.16 Phase 2: Time measurements for the NT hashes. 162
8.17 Revealed passwords. 165
8.18 The results from the dictionary attack (MD5 hash). 165
8.19 Phase 2: Time measurement for the MD5 hashes. 166

List of Listings

2.1 The 95 standard printable ASCII characters. 35
2.2 The users *nerblak* and *root* presented in the *passwd* file. 43
2.3 The users *nerblak* and *root* presented in the *shadow* file. 44
2.4 The user *root* presented in the *master.passwd* file. 46
2.5 This shows the hash files used in Mac OS X 10.4. 47
2.6 The user *nerblak* presented in its *uid* named hash file. 48
2.7 John the Ripper issued as root with wordlist and rules enabled. 57
7.1 How to run lm2ntcrack.pl with a John the Ripper output file. 131
8.1 The character sets used by the Vista special rainbow tables. 136
A.1 Script to build YALP. 219
A.2 Script to build custom kernel of YALP. 221
B.1 YALP startup script. 223
B.2 Replace startup file script. 228
B.3 Edit shadow file script. 231
B.4 Mount operating system script. 233
D.1 Script to add 30 users in a Windows system. 239
D.2 Script to add 30 users in a Unix system. 240

Abbreviations

SAM	Security Accounts Manager
DES	Data Encryption Standard
MD4	Message-Digest algorithm 4
MD5	Message-Digest algorithm 5
SHA	Secure Hash Algorithm
SHA1	Secure Hash Algorithm 1
LM	LAN Manager
ASCII	American Standard Code for Information Interchange
DES	Data Encryption Standard
LILO	LInux LOader
GRUB	GRand Unified Boot loader
ANSI	American National Standards Institute
BSD	Berkeley Software Distribution
BIOS	Basic Input/Output System
MBR	Master Boot Record
SysV	Unix System V
ISO	International Organization for Standardization
GUI	Graphical User Interface
SMB	Server Message Block
SSH	Secure Shell
FAT32	File Allocation Table 32

NTFS	New Technology File System
HFS	Hierarchical File System
HFS+	Hierarchical File System Plus
UFS	Unix File System
ext2	second extended file system
ext3	third extended file system
TPM	Trusted Platform Module
EFI	Extensible Firmware Interface
SSH	Secure Shell
YALP	Yet Another Local Password (tool)
NT	New Technology
NTLM	New Technology LanMan
WRP	Windows Resource Protection
SFC	System File Checker
PAM	Pluggable Authentication Modules
ROM	Read Only Memory
Ntldr	NT Loader
SCSI	Small Computer System Interface
ATA	Advanced Technology Attachment
GINA	The Graphical Identification And Authentification
LSASS	Local Security Authority Subsystem Service
SCM	Service Control Manager
SAS	Secure Attention Sequence
SID	Security Identifier
Smss	Session Manager Subsystem
HAL	Hardware Abstraction Layer
BCD	Boot Configuration Data

Definitions

Rainbow Tables Rainbow tables are tables consisting of pre-computed hash values used in the process of cracking passwords.

ASCII characters ASCII characters are a set of codes that is used by the computer to convert letters and other characters to and from numbers.

Unicode Unicode is an universal character encoding standard used for representation of text for computer processing.

Parity bit A parity bit is a bit added to ensure that the number of bits set to 1 is always even or odd. The parity bit is normally used as an error-detecting code.

Live CD A live CD is a bootable CD with an operating system that is pre-installed on the CD. This operating system does not need to be installed on the host system, it can boot directly from the CD.

Host operating system A host operating system is the operating system that is pre-installed on the computer.

Guest operating system A guest operating system is an operating system that is booted on top of the host operating system.

Password cracking Password cracking involves retrieving the encrypted password representation and trying to recover the original clear text password using an automated tool.

Mangling Mangling is for example used in dictionary attacks, and means that variations of each word listed in a wordlist is tried in order to reveal a password.

Runlevel A runlevel is a preset operating state on a Unix-like operating system.

Chapter 1

Introduction

According to a Dell-sponsored study [pon08], conducted by Ponemon Institute, business travelers lose more than 12,000 laptops per week in U.S airports. This shows that business travelers are putting sensitive information of their organizations at risk when they travel through airports. In the study, over 53% of the business travelers say that their laptops contain confidential or sensitive information.

In a computer crime and security survey carried out by CSI/FBI in 2006 [GLLR06], 616 computer security practitioners in U.S. corporations, government agencies, financial institutions, medical institutions and universities were questioned about the security in their organization. The survey reveals that the top four categories of losses are (1) viruses, (2) unauthorized access, (3) laptop or mobile hardware theft and (4) theft of proprietary information. A weak login mechanism can lead to both unauthorized access and theft of proprietary information. This shows the importance of highlighting these weaknesses.

When performing a quick search on Google for methods how to reset the password on a Windows machine, Microsoft presents a web page containing their advice when the password is lost [3]. Microsoft claims the following:

> *If you do not have a reset disk or cannot log on as an administrator, unfortunately, you may have to reinstall Windows XP and all other programs that were installed on the computer before you can use the computer again.*

As Microsoft claims Windows XP **may** have to be reinstalled, this thesis will among other things look further into other possible solutions available.

The market share of different operating systems is presented in Figure 1.1. This figure is based on data obtained from [4], and among other things illustrates that Windows have the biggest market share, with Windows XP as the most widely used operating system (63.76%). The figure also demonstrates how Windows Vista has not become as generally accepted as Microsoft may have wished for. The data presented in Figure 1.1 shows the popularity of operating systems, and supports our choice of operating systems selected for the experiments described in this thesis.

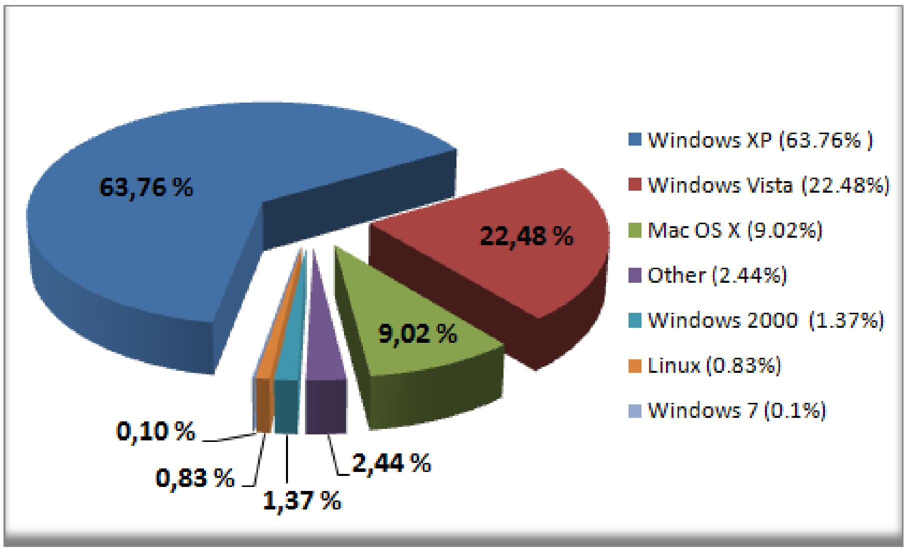


Figure 1.1: The market share of different operating systems.

1.1 Motivation

A system should not allow unauthorized persons to access the information stored on the computer. We think it is necessary to investigate different operating systems to give a description of how the systems are handling the login procedure when passwords are used as authentication token. The thesis will also present an evaluation regarding the degree of difficulty when it comes to resetting and/or recovering the administrator password. Even though there are procedures and tools already available on the Internet, no overview of these procedures and tools exists as far as we know. These procedures and tools are mostly described on different forums on the Internet, and a gathering of this information may be seen as a valuable contribution. The procedures and tools are often surprisingly easy to use, requiring no expertise from the user. It is also a fact that people forget their password and need to either reset or recover their passwords. The challenge is to allow authorized users to recover and/or reset their password, and prevent unauthorized users from exploiting these techniques to their advantage.

By evaluating the degree of difficulty to perform this task a conclusion can be made whether this is too simple and if there are ways to make it harder, but still possible, for the user to get access to the system after he has lost the password.

A study showing the strength of using passwords as an authentication token for accessing the operating system would be desirable. Will passwords offer sufficient protection in the future?

1.2 Related Work

Even though there already exists work related to password recovery and password resetting, this mostly consist of forum posts and unpublished notes and articles. As far as we know, no comprehensive collection describing procedures and tools regarding password resetting and recovery has yet been published.

Many security distributions can be found on the Internet, such as BackTrack [5], Protech [6] and Damn Vulnerable Linux [7]. None of these security distributions have their main focus on password resetting and recovery, and are thus containing a large amount of unnecessary tools, when the goal is to get access to a system where the password is lost. A distribution called Trinity Rescue Kit [8] has specialized on providing many different solutions when access to the system is needed. The disadvantage with this distribution is that it lacks some tools and implementations of procedures that are useful for a password resetting and recovery. This distribution is also large in size and quite complicated.

Several studies have been carried out to evaluate different aspects of password security. Klein presents in his article, *"Foiling the Cracker": A Survey of, and Improvements to, Password Security* [Kle91], some problems of the current password security. Various techniques used by crackers are presented and a proactive password checker is proposed.

Yan et al. [YBAG04] presents a survey containing some empirical results regarding the memorability and security of passwords. This survey highlights that mnemonic phrases are just as hard to crack as random passwords, yet just as easy to remember as naive user selections.

1.3 Objectives

This thesis consists of three main parts. First a thorough theoretical background study of the login procedure and password handling in the different operating systems is given. Secondly we will set up a testbed consisting of eight computers with different operating systems. In the second part we present a description on how to reset or recover the administrator or root password by applying different procedures and tools. The last part presents an empirical password study, where we test the strength of 30 passwords on different operating systems. We aspire to achieve the following objectives:

- Make an overview of procedures and tools used to reset or recover the Administrator password on the selected systems.
- Make a comparison of the procedures and tools regarding their characteristics when it comes to quickness and simplicity.
- Perform a study to test the strength of different passwords.
- Compare the security related to the login procedure on the selected operating systems.

- Try to give suggestions on how to improve the login security and/or the protection of files.

1.4 Limitations

As we shall see from this thesis there are a number of ways to get around a login procedure to get access to the files stored on the system. Although it is possible to just use a live CD of a different operating system, for example the Ubuntu live CD [9], to get access to the files stored on a system running for example Windows XP, this is not the focus in this thesis as we concentrate on password resetting and recovery. Even though *file access* is not the focus, we will describe how to use a live CD to obtain the *password file* stored on the operating systems, as this is relevant when it comes to recovery and resetting of passwords.

Although we describe some hashing techniques, for example in connection with Windows password handling, it is not the intention to go into detail about all the hashing techniques mentioned in this thesis. We have nevertheless, decided to look closer into the hashing techniques used by windows, as these techniques seem to have large security flaws.

In this thesis we have chosen not to look closer into encryption of file systems. It is discussed whether this is a solution to prevent unauthorized access or not, but this is not experimented with. This thesis also omits describing other login mechanisms with other authentication tokens than passwords.

The experiments carried out in this thesis require physical access to the computer. This means that the thesis will not cover password resetting and password recovery over networks.

1.5 Methodology

This section will outline the methodology used in this thesis. First the weaknesses of the login mechanisms in popular operating systems are identified. For each operating system the password handling and boot process is presented. A short introduction of some tools and techniques are also given. Then procedures and tools that exploit some of the weaknesses found are described. Each procedure and tool is structured in the same way. First the procedure or tool number with its corresponding name is given. Then a table containing key properties of the specific procedure or tool is displayed. Each procedure and tool is then presented with a short description, necessary prerequisites and a step-by-step approach. At last some possible variations of the approach are highlighted. It should be noted that an overview of all these procedures and tools are given in Chapter 4.

A tool created by the authors, described in the Chapter 7, was created as a reaction to the results received when the approaches for the procedures and tools were carried out. In the end of this thesis an empirical password study is presented. As this was a comprehensive study, both the approach and the results from the study were placed in a separate chapter, Chapter 8. The specific methodology used by the empirical password study is outlined in this chapter.

1.6 Document Structure

The remainder of this thesis is organized as follows:

Chapter 2: Background presents theory related to password handling and the login procedure in Windows-, Linux- and BSD based operating systems. A short presentation of some techniques and tools are also given in this chapter.

Chapter 3: Laboratory Environment describes the laboratory setup. The chosen hardware and operating systems is also specified.

Chapter 4: Method gives an overview of all the procedures and tools used to reset or recover the Administrator password.

Chapter 5: Procedure presents a step-by-step description of how to perform the selected procedures. It should be noted that this chapter does not necessarily have to be read in its entirety, and should instead be used as a reference work that presents various password resetting procedures.

Chapter 6: Tools presents a step-by-step description of how to perform the selected tools. This chapter as well does not necessarily have to be read in its entirety, it should be used as a reference work that is presenting various password resetting and recovery tools.

Chapter 7: Results presents the main findings of the experiments which are further evaluated in Chapter 9. Our self-made tool is also described in this chapter.

Chapter 8: Empirical Password Study gives a thorough evaluation of the strength of 30 carefully selected passwords. The purpose of this study is to find out how many of the passwords we were able to reveal during 8 hours, with the use of a dictionary attack and a cryptanalysis attack. Results regarding this study are also presented in this chapter.

Chapter 9: Discussion evaluates the achieved results from both Chapter 7 and 8. Finally, some ideas for future work are presented.

Chapter 10: Conclusion summarizes the major results and findings.

Additionally, the following appendices are included:

Appendix A: How to Build YALP shows the instructions needed to build YALP and its kernel, using two distinctive scripts.

Appendix B: YALP Scripts presents the different self-made scripts included in YALP.

Appendix C: How to Disable LM Hashing describes a possible way to disable the storing of the LM hash on a computer running Windows XP.

Appendix D: How to Add 30 Users presents the scripts made to easily add 30 users in Windows and Unix systems. These scripts were used in the Empirical Password Study.

Appendix E Attachments: presents the content of the attached CD and the files submitted to DAIM [10].

Appendix F Poster: includes the poster *Generation of Rainbow Tables*, which is based on findings from this thesis. The poster was presented at The 8th Annual Meeting on High Performance Computing and Infrastructure, NOTUR2009.

Appendix G Paper: includes the scientific article *All in a day's work: Password cracking for the rest of us*, which is based on this thesis and submitted to The 14th Nordic Conference on Secure IT Systems, NordSec 2009.

1.7 Conventions

Through the thesis we will use the terms *procedures*, *tools* and *techniques.*

Procedures are walkthroughs to reset a password.

Tools are already available programs that can recover and/or reset passwords.

Techniques There are many password recovery and resetting techniques used by procedures and tools, such as Ophcrack using the technique *password cracking with rainbow tables* to recover the lost password.

Even though some people [11] may argue that it is more correct to write *GNU/Linux* instead of *Linux* we have chosen to write *Linux* throughout this thesis.

We are differentiating between a *host* operating system and a *guest* operating system. A host operating system is the operating system that is already installed on the computer, while a guest operating system is an operating system that is booted on top of the host operating system. An example may be a computer with Windows Vista already installed as the host operating system. If for example an Ubuntu live CD is booted to access the files stored on the Windows Vista computer, Ubuntu is the guest operating system. It should be noted that even though the guest operating system does not fit on a CD and has to be saved on other media, such as DVD or USB, we still call it a live CD.

Chapter 2

Background

This chapter will first present some important techniques used in later chapters. It will then present theory related to the login procedure and password handling in Windows-, Linux- and BSD-based operating systems. In the end this chapter will also describe in detail the different tools used to reset and/or recover the administrator or root password to be able to log on to the different operating systems without knowing the password in advance.

2.1 Techniques

The techniques described in this section should not be confused with the procedures described in Chapter 4. The procedures used in Chapter 4 use one of the two main techniques password resetting or password recovery, and may use some of the tools described in Section 2.4. The two main techniques may again use other techniques. In password recovery we can for instance use rainbow tables as a technique to crack passwords. There are no well known techniques to reset a password, such as for password recovery, but some forum posts gives suggestions on how this can be accomplished for specific systems. Procedures that realize password resetting are described in Section 5.1 in Chapter 5. Many tools implement their own solution to reset a password. Although one may argue that the procedures and tools to reset or recover the password are only meant for the legitimate purposes, this is not always how it works. The tools are getting more and more advanced and exploits using new weaknesses are revealed every day. Choosing a good password is therefore very important.

2.1.1 How to Choose a Password

It is common to have a login process asking for a user name and a password to protect a system from unauthorized persons. If this process is secure it should create a barrier that prevents the intruder from entering the system, thus protecting the system from being compromised. The problem with this process is that the system is dependent on the user selecting a good password.

In Schneier's article *Choosing Secure Passwords* [12] it is discussed how a password is built up. A typical password consists of a root plus an appendage. A root is something pronounceable, not necessarily a word from a dictionary. The appendage is in 90 percent of the time a suffix and 10 percent of the time a prefix. *Ola83* is an example of a weak password having *Ola* as a root and *83* as appendage.

Usually the ignorant user selects a weak password thinking that the password is secure enough. According to Klein [Kle91] a user tends to select a password with one or more of the following properties:

- Passwords based on the user's account name.
- Passwords based on the user's initials or given name.
- Passwords which exactly match a word in a dictionary.
- Passwords which match a word in the dictionary with some or all letters capitalized.
- Passwords which match a reversed word in the dictionary.
- Passwords which match a reversed word in the dictionary with some or all letters capitalized.
- Passwords which match a word in a dictionary with an arbitrary letter turned into a control character.
- Passwords which match a dictionary word with numbers such as '0' and '1' substituted for the letters 'o' and 'l'.
- Passwords which are simple conjugations of a dictionary word (i.e., plurals, adding "ing" or "ed" to the end of the word, etc.).
- Passwords which are patterns from the keyboard (i.e., "aaaaaa" or "qwerty").
- Passwords which are shorter than advised.
- Passwords which consist solely of numeric characters (i.e., Social Security numbers, telephone numbers, house addresses or office numbers).
- Passwords which do not contain mixed upper and lower case, or mixed letters and numbers, or mixed letters and punctuation.
- Passwords which look like a state issued license plate number.

Most users also tend to prioritize an easy to remember password instead of a secure one. In a whitepaper published by AccessData [13] some simple facts, considering what users usually base their password on to remember them, are presented:

- The password will usually be in a language familiar to the owner.
- The password will usually be an aspect of the owner's life.
- New passwords might be a modification of old passwords.

As many studies show [14], [15] and [16], the most popular passwords selected by users across the world are common words, often with tiny changes. As presented later in this thesis, these passwords are easily revealed with the right tools.

The problem with selecting passwords with the properties presented in both of the lists above is that this can easily be exploited by unauthorized persons with the goal to reveal the password. When a person is trying to reveal a password, the first password guessed are the passwords containing the properties mentioned above. The next sections present the processes of guessing and cracking passwords more in detail.

A good question to ask is then; what passwords should be chosen? A good password is one that is easily remembered, yet difficult to guess. Bruce Schneier presented in his blog [17] after going through data retrieved from MySpace, that 65% of the MySpace passwords contain 8 characters or less. Most of the passwords (25%) had a length of 8 characters. According to Schneier [12] you should choose a password not containing a root or an appendage already listed in any dictionaries. You should mix upper and lower case letters in the middle of your root. You should add numbers and symbols in the middle of your root, not as common substitutions. You can also drop the appendage in the middle of your root, or just use two roots with an appendage in the middle.

So how would you choose a secure password? It is clear that passwords should be of some certain length to be secure. Microsoft suggests using a password that has 8 characters or more, ideally 14 characters or longer [18]. One may wonder if this has something to do with Microsoft's own password handling (further described in Section 2.3.1.2). The best password should appear as a random string of characters. Randomness makes passwords harder to guess, but also harder to remember. Schneier [Sch96] suggests using an entire phrase instead of a word, and to convert that into a key. These phrases are called *pass phrases*. He recommends choosing phrases that are unique and easy to remember, but advises not to use phrases from the literature. One of the problems with Schneier's approach is that some password schemes might not support such long phrases.

A good way to remember a password that looks random is to use so called mnemonic phrases. For example, take the first letter of each word in a phrase, then add or replace a few special characters or numbers. An example of such a phrase can be: *Fifth Grade Comm Tech waiting for their Graduation at NTNU.* This can be reduced to *5GCTw4tG@N*. Using mnemonic password phrases is a good technique, but you might need some patience to come up with different phrases every time the password needs to be changed. Kuo et al. [KRC06] hypothesize that users will select mnemonic phrases that are commonly available on the Internet, and that it is possible to build a dictionary to crack mnemonic phrase-based passwords.

Another way to choose a secure password might be to let a password manager do it for you. Of course you would have to trust this password manager to be secure, and the password might not be easy to remember anyway. When remembering the password is the worst thing about having a good and secure password, some security experts such as Schneier [19] suggest writing the password down.

2.1.2 Password Recovery Techniques

A password can be retrieved in many ways. Passwords can be recovered locally on the system or remotely through a network. It is common to separate between two main password recovery techniques, password guessing and password cracking. This thesis focuses mostly on the latter technique. Passwords can also be retrieved using other techniques such as through social engineering, phishing, and click fraud, but these techniques are not presented in this thesis. This thesis will focus on recovering the password locally. The password guessing will then either be performed locally on the system where the password is lost, or on a system that has already retrieved the passwords or password files on beforehand.

2.1.2.1 Password Guessing

When a password is lost, a common method to recover the password is to try and guess what the password was. The user identification is known and password guessing is usually performed on the computer subject to the lost password. The user inserts the user identification and a likely password, and then waits for a responding message from the system. If the system prompts that the login was successful the password was correct, or if the login went unsuccessfully the user tries another password using the same user identification. This technique is very simple and many tools exist to automate the guessing process. The tool can then interact directly with the login interface running locally on the machine or through a network. Popular password-guessing tools freely available on the Internet are Brutus [20] and THC Hydra [21]. In many systems default passwords are often left unchanged, and these are well known to people with criminal intentions. Many default passwords are listed on web pages such as [22]. When guessing passwords a dictionary can be used. The dictionary can be created to suit the guessing process with default passwords and common words, so that if the original password is a word in the dictionary, it will be guessed. Appended random characters and permutations of the words in the dictionary, often called mangling is another clever way to eventually guess the correct password. The mangling process will be described in more details later in this chapter.

There are some severe limitations using the password guessing technique [SL05]. The first limitation is that most login procedures include a maximum set of login attempts allowed. The user usually then has to wait a certain amount of time before another attempt to login is possible again. Some systems even lockout the account until the user securely can confirm his/her identity. This can result in that the process of guessing the password could take days, months or even years to perform. Additionally if the password is strong the process can seem endless. Another limitation of this technique is that the process of guessing a password often takes place remotely through a network.

The process will generate traffic not normally expected and intrusion detection systems and/or intrusion prevention systems might detect the anomaly from ordinary traffic.

The password guessing technique often never works because administrators and regular users change the default password. The tools that provide these techniques commonly use to much time. Thus, a much more powerful technique to recover the password is presented in the section below called password cracking.

2.1.2.2 Password Cracking

When passwords are used, a way to check if the user has typed in the correct password is necessary. Most systems store the password in some kind of form locally on the system or on a centralized server. To protect the passwords they are often stored in files meant to be inaccessible for others than administrators. These password files are also often obfuscated so that they become hard to interpret. An example of this is described more in detail in Section 2.3.1.2. If the password files risk ending up in wrong hands, additional steps are performed to protect the passwords. It is common to represent the password as a hash value located inside the password files. Some systems use hash algorithms based on encryption algorithms such as Data Encryption Standard (DES), while others use pure hash algorithms, such as Message-Digest algorithm 5 (MD5) and Secure Hash Algorithm 1 (SHA1), a one-way algorithm transforming the password. To verify that the user is the correct one, the password prompted is first transformed with an algorithm as mentioned above and then compared with the already stored copy of this transformation. If a match is present, the user has been authenticated and the correct password was typed.

Skoudis and Liston define password cracking in their book [SL05] the following way:

> *Password cracking involves stealing the encrypted password representation and trying to recover the original clear text password using an automated tool.*

This thesis has chosen to widen the definition provided above, in such a way that password cracking not only is a technique used for criminal purposes. In this thesis recovering the password by using this technique, is often necessary for the legitimate user/administrator. It is then better to replace the word *stealing* by *retrieving.* The definition used in this thesis is therefore as following:

> *Password cracking involves retrieving the encrypted password representation and trying to recover the original clear text password using an automated tool.*

When the password file already is retrieved in advance, none of the limitations mentioned in the password guessing section applies. The password cracking process can be presented through a loop consisting of three steps. These three steps are; first guessing a password, second generating a hash of the guessed password and third comparing the newly generated hash with the original hash. If the compared hash values match, a valid password is found. Depending on the computer hardware and software used, thousands of passwords per second can be guessed and matched.

When the password files, or passwords, are fed to the password cracking tool it can perform various techniques to increase the chance of guessing the right password in a shorter period of time. The most popular techniques to form the password guesses is either by using a dictionary or just by brute forcing all possible passwords.

By using a dictionary, also called wordlist, a collection of words is guessed. A large number of dictionaries are available on the Internet, in many different languages and forms. The wordlists can be found many places on the Internet for instance at *The A.R.G.O.N.* web page [23].

When the brute-force technique is used, the password recovery tool guesses every possible combination of characters to determine the password. The tool can then create guesses of combinations of all alphanumerical and special characters/symbols. The brute-force guessing technique can take an enormous amount of time, ranging from hours to centuries.

Many password recovery tools offer a hybrid of the dictionary and brute force techniques, called mangling, mixing or just hybrid mode. A tool using a hybrid technique starts with guessing every word in the dictionary. When this is done it creates other guesses by using so called mixing or mangling rules. The mixing/mangling rules tell the tool to change a portion of every word in the dictionary. These rules can for instance be appending or prepending characters to the dictionary words, or adding the dictionary word to itself of number of times creating a new word. The rules vary from tool to tool.

A last technique, which can be used in combination with the techniques mentioned above, is the use of rainbow tables. By using rainbow tables a time-memory trade-off is accomplished. The password recovery tool will check in the rainbow tables for a match with the target hash value to find the desired password. This is described in detail below.

By using the techniques mentioned above four password cracking attacks can be performed. These four attacks are often referred to as dictionary-, brute-force-, hybrid- and cryptanalysis-attack. The cryptanalysis-attack utilizes already precomputed rainbow tables. In advance a charset is defined and rainbow tables are created based on this charset. A charset can for instance be *alpha-numeric-symbol32-space* meaning all alpha characters, all numbers, 32 different symbols and space. The charset can be represented in a text file containing for instance this line (the last character is the whitespace):

alpha-numeric-symbol32-space = [ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
!@#$%^&*()-_+=~‘[]{}|\:;”’<>,.?/].

2.1.2.3 Rainbow Tables

Using precomputed rainbow tables can reduce the password cracking time drastically. The rainbow tables contain a connection between a hash and its password. By having this connection already available for all possible hash values, a quick search through the tables for a desired hash value can reveal its password. To reduce the size of the precomputed tables reduction functions and chains are used. The original concept of hash tables, chains, reduction functions, rainbow chains and rainbow tables are described more in detail below.

A password becomes hard to guess if three properties are fulfilled. The first property is that no common words are used; the second is that the password is of a certain length, thus making the set of passwords very large. The third property is that characters should vary between numbers, lower and uppercase letters and special symbols. The usual approach for cryptanalytic attacks on hashed password involves a function of three steps, as mentioned in the previous section, repeated until the correct password is guessed. When a password is found it can either be the original password or a password that generates the same hash value as original one. When two different passwords produce the same hash value, it is called a collision.

Instead of performing the four steps mentioned above innumerable times demanding a lot of computing power and lot of time to complete, it has been suggested to precompute step one and two for all possible passwords. Then only step three remains each time a cryptanalytic attack is performed. This approach is not practicable for long passwords because of the large amount of memory needed to store all possible hashes.

In 1980 Hellman introduced a time-memory trade-off when applying a cryptanalytic attack. According to [Hel80] this method cryptanalyzes any N key cryptosystem in $N^{2/3}$ operations with $N^{2/3}$ words of memory (average value) after a precomputation which requires N operations. Using Hellman's method the time-memory trade-off applies for the recovery of a key when the plaintext and the ciphertext are known. The method works for password hashes when the hashing scheme is poorly designed, the plaintext and the encryption method will be the same for all passwords [Oec03].

Hellman [Hel80] defines the formula $C_0 = S_k(P_0)$, where P_0 is indicated as a fixed given plaintext, C_0 as the corresponding ciphertext (in this thesis usually a hash), S as the given encryption algorithm and $k \in N$ as the key. As an example used in the real world, the formula can be compared to the LM hash mentioned in 2.3.1.2 on page 31, where the fixed plaintext input of the LM hash encryption scheme is P_0. C_0 is the corresponding LM hash and k is the password being hashed.

Hellman used something called *hash chains* to decrease the memory requirements mentioned above. Instead of storing all the hashes in memory, the hashes are organized in chains. Only the first and the last element of the chain are stored in the memory, saving memory at the cost of cryptanalysis time [Oec03]. According to Hellman [Hel80] the cryptanalyst choose m starting points, each an independent random variable drawn uniformly from a keys space {1,2,...,N}. This means that there is no connection between the endpoint of one chain and the starting point of another chain. A reduction function R is used to reduce the length of the hashes. The reduction function performs the reverse of what the hash function does, meaning that it maps hash values (ciphertext) to keys. It is a one-way function such as the hash function, but is not an inverse of the hash function. The chain process can be understood through the following formula:

$$k_i \xrightarrow{S_{k_i}(P_0)} C_i \xrightarrow{R(C_i)} k_{i+1} \tag{2.1}$$

An example of such a chain can be viewed below where the hash is represented with uppercase letters and the key with lowercase letters (P_0 is fixed as 1234):

$$\begin{aligned} &abcd \xrightarrow{S_{abcd}(1234)} KJWN \xrightarrow{R(KJWN)} qwdp \xrightarrow{S_{qwdp}(1234)} ASDK \xrightarrow{R(ASDK)} \\ &oier \xrightarrow{S_{oier}(1234)} DFGA \xrightarrow{R(DFGA)} idso \xrightarrow{S_{idso}(1234)} RPFU \xrightarrow{R(RPFU)} plsr \end{aligned} \tag{2.2}$$

Often the reduction function $R(S_k(P_0))$ is written as $f(k)$ leading to a chain of keys presented in the formula below [Oec03]:

$$k_i \xrightarrow{f} k_{i+1} \xrightarrow{f} k_{i+2} \rightarrow ... \tag{2.3}$$

The example above can then be written as:

$$abcd \xrightarrow{f} qwdp \xrightarrow{f} oier \xrightarrow{f} idso \xrightarrow{f} plsr \tag{2.4}$$

A set of initial passwords is chosen and m chains of length t are computed with the first key (starting point) and last key (endpoint) stored in a table. From the example above *abcd* and *plsr* will be stored in the table. When the table is fully generated the cracking process can be initiated. Given ciphertext C (in this thesis a hash), a key used to create C can be found using the table mentioned above. First a chain X of keys starting with $R(C)$ up to the length t is generated. If the ciphertext was generated with a key also used while creating the table, a key in X will match one of the endpoints stored in one of the chains in the precomputed table. Using the starting point corresponding to the matched endpoint, the chain with keys from the starting point up to $R(C)$ can be computed. The key before $R(C)$ is then the key used to generate C [Oec03]. For instance if the hash $DFGA$ is given as C, the X chain is generated as follows:

$$DFGA \xrightarrow{R(DFGA)} idso \xrightarrow{S_{idso}(1234)} RPFU \xrightarrow{R(RPFU)} plsr \tag{2.5}$$

The key *plsr* match with the last key in a chain stored in the table, and the key used to create C can easily be found by using the first key *abcd*, corresponding to *plsr* in the table. The final chain is generated and the correct key *oier* is found. The key *oier* is marked boldface below:

$$\begin{aligned} &abcd \xrightarrow{S_{abcd}(1234)} KJWN \xrightarrow{R(KJWN)} qwdp \xrightarrow{S_{qwdp}(1234)} ASDK \\ &\xrightarrow{R(ASDK)} \textbf{oier} \xrightarrow{S_{oier}(1234)} DFGA \end{aligned} \tag{2.6}$$

Hellman's hash chains have serious flaws. They can often generate *false alarms* when chains merge and ciphertext not included in one of the chains in the table lead to the same endpoint. If this is the case the chain X ignores the match and continues to expand.

A *collision* between the precomputed chains can also occur where the same values are produced by the reduction function. These chains will also merge. Fewer passwords covered by a table will be the consequence of these merges.

To avoid these merges, at the cost of increasing the key lookup time, Oechslin [Oec03] proposed a method in 2003 which he called *rainbow tables*. Instead of using a single reduction function R, Oechslin suggest using a sequence of related reduction functions R_1 up to R_{t-1}. The chains generated are called rainbow chains. According to Oechslin, if two chains collide, they merge only if the collision appears at the same position on both chains. If the collision does not appear at the same position, both chains will continue with a different reduction function and thus will not merge. This decreases the set of chain merges drastically.

Figure 2.1 demonstrates how a hash is found using rainbow tables. The rectangular boxes contain keys, and the oval circle contains the hash. R_x indicates the reduction functions where x is the reduction function number. S is the hash function used. On the left side of the figure a box containing a rainbow table is displayed. This table contains rows with the first key, the starting point, and the last key, the endpoint, used in a rainbow chain. In the figure the process begins with the hash *DFGA*. A key *idso* is retrieved from the reduction function R_3 and the hash. The key *idso* does not match any of the endpoints in the table, and S and R_4 has to be applied. The key *plsr* generates a match with the endpoint *plsr* in the rainbow table and a chain with starting point *abcd* can be generated. The key *oier* is detected as the key corresponding to the initial hash *DFGA*.

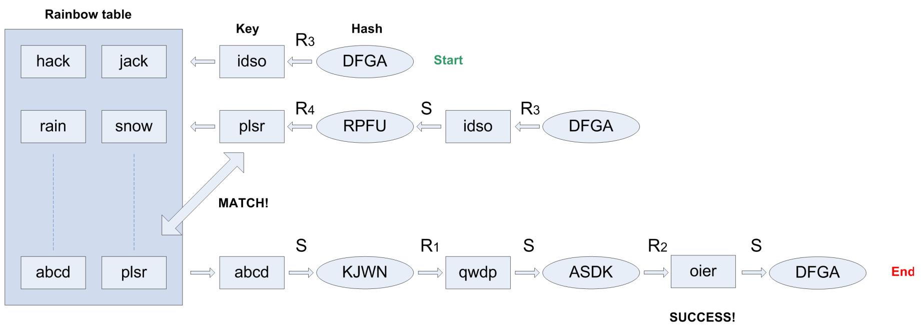


Figure 2.1: The lookup process in rainbow tables.

As presented in [24] the best defense against the use of rainbow tables is a *salt*. The salt is a random string added to the password before the password is hashed. This salt can safely be stored unencrypted with the hash of the password. Rainbow tables have to be created for each salt, making the set of rainbow tables extremely large. Many systems use a salt in their hashes, and even though a salt solves the problems to some extent, popular operating systems such as Windows omit this security feature. Popular hash algorithms and their salts are presented in more detail in Section 2.3.2.2 and 2.3.1.2.

2.2 Boot Process

In this section the boot processes for Windows-, Linux- and BSD-based systems are presented. This section will highlight important programs and sub-processes initiated during a system boot, programs and processes dealing with user login and system startup. It is important to know that this section does not present remote login mechanisms from a remote location using for instance *Secure Shell (SSH)* or *remote desktop.* The login mechanisms presented in this section only handles the local logins, where the users are physically interfering with the system and the systems are physically started.

All of the systems boot into similar or different file systems. Some of the systems have support for mounting other file systems than their primary file system, after the boot process has finished. Windows operating systems use the *File Allocation Table 32 (FAT32)* and *New Technology File System (NTFS)* file system, while Linux systems (Ubuntu and Fedora, used in the experiments) use mostly the *second extended file system (ext2)* and the *third extended file system (ext3). Hierarchical File System (HFS)* and *Hierarchical File System Plus (HFS+)* are the primary file systems used by Mac OS X. The *Unix File System (UFS)* file system originating from Version 7 Unix is used by FreeBSD. As NTFS seems to be the preferred file system for Windows systems, support for mounting read and write on NTFS from a Linux system is possible through programs such as *NTFS-3G.* Many other file system types exist, but they are not relevant for this thesis.

This section contains specific details concerning the boot process of various operating systems. These details are relevant to get a basic understanding of some of the procedures and tools presented later in this thesis.

2.2.1 Windows

The Windows boot process, or startup process, is the process where the Windows operating systems are initialized. It should be noted that there are some differences between the so called Windows NT boot process, which includes Windows NT, Windows 2000, Windows XP and Windows Server 2003, and the boot process of newer operating systems, including Windows Vista and Windows Server 2008. The differences between these boot processes will be highlighted. Note that this section only explains the boot process for x86 operating systems. The intention with this section is to highlight some of the important processes that are invoked during the boot process.

Already when Windows is installed on a computer, preparations for subsequent Windows boot processes is carried out. During the Windows setup process the system's primary hard disk is prepared with code that takes part in the boot process. Windows operating systems split hard disks into discrete areas known as *partitions*, and use file systems such as FAT and NTFS to format each partition into a *volume.* It may be added that a hard disk can contain up to four primary partitions [RS04].

Physical disks are addressed in units known as *sectors.* The *MS-DOS Fdisk* utility and the *Windows Setup program* are utilities that prepare hard disks for the definition of volumes. These utilities write a sector of data that is called a Master Boot Record (MBR) to the first sector on a hard disk. The MBR includes a fixed amount of space that contains *boot code*, which is executable instructions, and a *partition table*. The partition table contains four entries that define the locations of the primary partitions on the disk. When a computer boots, the first code it executes is called the Basic Input/Output System (BIOS). The BIOS is encoded into the computer's Read Only Memory (ROM). The BIOS then selects a boot device, reads that device's MBR into memory, and transfer control to the code in the MBR [RS04].

The code included in the MBR first scans the primary partition table until it locates a partition that contains a flag that signals that the partition is bootable. And when the MBR finds at least one such flag, it reads the first sector from the flagged partition into memory. Then the MBR transfer control to the code within the partition. This type of partition is called a *boot partition*, and the first sector of the boot partition is called a *boot sector*. The volume defined for the boot partitions is called the *system volume* [RS04].

It should also be added that operating systems write boot sectors to disk without the involvement from the user. An example is when Windows Setup writes the MBR to a disk, it also writes a boot sector to the first bootable partition of the disk. But before writing to the boot sector of a partition, Windows Setup ensures that the partition is formatted with a file system that is supported by Windows. This is done by formatting the boot partition with a file system type specified by the user. FAT, FAT32 and NTFS are file systems supported by Windows [RS04].

Windows Setup is also responsible for creating a boot menu file in the root directory of the system volume. This boot menu file is named *Boot.ini*. This file contains options for starting existing Windows installations, and the Boot.ini file therefore needs to specify the path to the particular Windows partition. This path is specified in a special syntax that conforms a to the Advanced RISC Computing (ARC) naming convention [RS04].

As mentioned above, the Windows Setup must know the partition format before it writes a boot sector because the contents of the boot sector may vary depending on the format. For example if the partition is a NTFS format, Windows writes NTFS-capable code. The role of the boot-sector code is to give Windows information about the structure and format of a volume and to read in the NT Loader (Ntldr) from the root directory of the volume. The boot sector code therefore contains just enough read-only file system code to accomplish this task. The Ntldr is the boot loader used by all releases of Microsoft's Windows NT operating system, up to and including Windows XP and Windows Server 2003. The Ntldr file among other things reads the Boot.ini and presents the boot menu. After the boot sector code has loaded the Ntldr file into memory, it transfers control to Ntldr's entry point. If the boot sector code is unable to find Ntldr in the root directory of the specific volume it displays an error message [RS04].

Ntldr begins its existence while the system is executing in an x86 operating mode called *real mode*. x86 is the generic term which refers to the most commercially successful instruction set architecture. Real mode is a mode where no virtual-to-physical translation of memory addresses occurs. This means that programs that use the memory addresses interpret them as physical addresses and that only the first MB of the computer's physical memory is accessible. Simple MS-DOS programs do for example execute in a real-mode environment. The first action that Ntldr takes is to switch the system to *protected mode*. Still no virtual-to-physical translation occurs, but a full 32 bits of memory becomes accessible. After the system is in protected mode, Ntldr can access all of the physical memory. After creating a sufficient amount of page tables, Ntldr enables paging. Paging is a memory-management scheme where the operating system retrieves data from secondary storage in the same size blocks that are called *pages*. Paging is an important part of virtual memory implementation in most operating systems, which allows the physical address space of a process to be noncontiguous. Protected mode with paging enabled is the mode in which Windows executes in normal operation [RS04].

After the paging is enabled, Ntldr is fully operational, but still relies on functions supplied by the boot code to access IDE-based system and boot disks as well as the display. The boot code functions briefly switch off paging and switch the processor back to a mode in which services provided by the BIOS can be executed. If the disk is containing the boot volume is Small Computer System Interface (SCSI) based, and is not accessible using BIOS firmware support, Ntldr loads a file named Ntbootdd.sys and uses this instead of the boot code functions for disk access. Ntbootdd.sys is a device driver used for disk I/O on SCSI and Advanced Technology Attachment (ATA) systems where the BIOS is not used. SCSI is a set of standards that allow personal computers to communicate with peripheral hardware such as disk drives. Next, Ntldr reads the Boot.ini form the root directory using built-in file system code. The Ntldr contains read-only NTFS and FAT code, and the Ntldr's file system can read subdirectories. Then Ntldr clears the screen. If there are a valid Hiberfil.sys file in the root of the system volume, it shortcuts the boot process by reading the contents of the file into memory and transfers control to code in the kernel that resumes a hibernated system. That code is restarting the drivers that were active when the system was hibernated. The Hiberfil.sys is valid if the computer was hibernated last time the computer was shut down.

If the Boot.ini file, which is read by Ntldr, contains more than one boot-selection entry, it presents the boot-selection menu to the user, and the user may choose which partition to boot. If the user does not select an entry from the selection menu within the timeout period which is specified by Boot.ini, Ntldr chooses the default selection. The default selection is the top-most entry in the Boot.ini file. The selection entries in the Boot.ini file direct Ntldr to the partition on which the Windows system directory of the selected installation resides. If there is only one entry in the Boot.ini file, Ntldr skips the boot-selection menu. The chosen partition can either be the same as the boot partition, or another primary partition. The entries in the Boot.ini file can include optional arguments that Ntldr and other components in the boot process interpret. These arguments will not be described, but an overview can be found in [RS04]. Any options that are included in the Boot.ini file is saved to the Registry value HKLM\System\CurrentControlSet\Control\SystemStartOptions. HKLM stands for HKEY_LOCAL_MACHINE. When the boot selection is made, Ntldr loads and executed Ntdetect.com. Ntdetect.com is a real-mode program that uses a system's BIOS to query the computer for basic device and configuration information. This information is stored under the HKLM\HARDWARE\DESCRIPTION registry key later in the boot. On Windows 2000, Ntldr then clears the screen and displays the *Starting Windows* progress bar, which remains empty until Ntldr begins to load the boot drivers. On Windows XP and Windows Server 2003, Ntldr presents a logo splash screen instead of a progress bar [RS04].

Even though this explanation is concentrated towards the x86 operating systems, it may be added that if it was an x64 version of Windows that was booted, the CPU would now be switched into *long mode*, which enables 64-bit addressing [RS04].

Next, Ntldr start to load the files from the boot volume that are needed to start the kernel initialization. The boot volume is the volume that corresponds to the partition on which the system directory (usually \Windows) of the installation that is booted is located. To load these files from the boot volume, Ntldr follows several steps. First it loads the Windows kernel (Ntoskml.exe) and a file named Hal.dll. Hal.dll is the core of Windows' Hardware Abstraction Layer. The function of the Hardware Abstraction Layer is to hide differences in hardware from most of the operating system kernel, so that most of the kernel-mode code does not need to be changed to run on systems with different hardware. If Ntldr fails to load either of these files (Ntoskml.exe or Hal.dll), an error message appears and the boot process halts. Then Ntldr reads in the SYSTEM registry hive, \Windows\System32\Config\System, so that it can determine which device drivers that need to be loaded to accomplish the boot. It may be added that a hive is a file that contains a registry subtree. Then Ntldr scans the in-memory SYSTEM registry hive and locates all the boot device drivers, which are drivers that are necessary to boot the system. These drivers are indicated in the registry with the following value: SERVICE_BOOT_START (0). Every device driver has a registry subkey under HKLM\SYSTEM\CurrentControlSet\Services. Now Ntldr calls the main function in Ntoskrnl.exe to perform the rest of the system initialization [RS04].

When Ntldr calls Ntoskrnl, it passes a data structure that contains a copy of the line in Boot.ini that represents the selected menu option for this boot, a pointer to the memory tables generated by Ntldr to describe the physical memory on the system, a pointer to the in-memory copy of the HARDWARE and SYSTEM registry hives, and a pointer to the list of boot drivers that Ntldr loaded. Ntoskrnl then begins the first phase of its two-phase initialization process. These phases are referred to as phase 0 and phase 1. During phase 0, basic internal memory structures are created, and each CPU's interrupt controller is initialized. The memory manager is also initialized, crating areas for the file system cache. Phase 1 consists of several steps that are performed during the boot splash screen. These steps will not be presented in detail, but the steps involve initializing the device drivers which were identified by Ntldr as system drivers [RS04].

Once the boot and system drivers have been loaded, the kernel starts the smss.exe. Session Manager Subsystem (Smss) (smss.exe) is almost like any other user-mode process. But there are two differences, and the first difference is that Smss is considered a trusted part of the operating system. The second difference is that Ntldr is a native application. Because it is a trusted operating system component, Smss can perform actions that few other processes can perform. An example is that Smss can create security tokens. And because Smss is a native application it does not use Windows APIs, but instead uses only core executive APIs known as Windows native API. The reason why Smss does not use the Windows APIs is because the Windows subsystem is not executing when Smss is launched. Actually, one of the first tasks performed by Smss is to start the Windows subsystem. Then the Smss calls the configuration manager executive subsystem to finish initializing the registry. The configuration manager is programmed to know where the core registry hives are stored on disk, and it records the paths to the hives it loads in the HKLM\SYSTEM\CurrentControlSet\Control\hivelist key [RS04].

The Session Manager Subsystem performs a number of actions. Smss among other things creates environment variables, starts the kernel-mode part of the Windows subsystem (win32k.sys) which allows Windows to switch into graphical mode, starts the user-mode part of the Windows subsystem (csrss.exe) which makes Win32 available to user-mode applications, creates virtual memory paging files, performs rename operations that might have queued up and starts the Windows Logon Manager (winlogon.exe). One of the actions mentioned above pointed out that Smss performed renaming operations that might have queued up. This is the reason why previously in-use files can be replaced as part of a reboot. Before any files are opened, Smss also starts *Autochk*, which mounts all drives and checks them one by one whether they have had a clean shut down or not. If the disk was not shut down cleanly, Smss will automatically start *chkdsk*. This disk check may be skipped by pressing any key within 10 seconds [RS04].

Winlogon, which is one of the processes started by Smss, is responsible for handling interactive logins to a Windows system, both locally and remotely. Winlogon again starts the Local Security Authority Subsystem Service (LSASS) (LSASS.EXE) and Service Control Manager (SCM), which in turn will start all the Windows services that are set to auto-start. Winlogon is also responsible for responding to the Secure Attention Sequence (SAS), loading the user profile, and optionally locking the computer when a screensaver is running. It may be added that SAS is a special key combination that might be required to enter before a login screen is presented [RS04].

The login process begins when a user presses the SAS (Ctrl+Alt+Delete), if this is activated. After the SAS is pressed, Winlogon calls The Graphical Identification And Authentification (GINA) to obtain a username and a password with use of the login display. The GINA library is a component of some Microsoft Windows operating systems that provides secure authentication and interactive login services. When GINA has loaded a few programs can be initiated. These are programs that Microsoft has included for people with physical disabilities. One of these programs is the Sticky Keys program which is a feature that makes it possible to press a modifier key. A modifier key is for example *SHIFT* or *CTRL*, and if the Sticky Keys program is running the modifier key remains active until another key is pressed. When the username and password is entered, GINA passes the credentials back to Winlogon. Then, Winlogon in addition creates a unique local logon Security Identifier (SID) for this user that it assigns to this instance of the desktop. Winlogon passes this SID to Lsass as part of the LsaLogonUSer call. If the user is successfully logged in, the SID will be included in the logon process token. When the username and password have been entered, Winlogon calls the Lsass function *LsaLookupAuthenticationPackage*. Authentication packages are listed in the Registry under HKLM\SYSTEM\CurrentControlSet\Control\Lsa. Winlogon passes logon information to the authentication package via *LsaLogonUser*. Once a package authenticates a user, Winlogon continues the logon process for that user. Winlogon loads the registry hive from the profile of the user that is logging in, and maps it to HKEY_CURRENT_USER (HKCU). It then sets the user's environment variables, and startup programs are run from different locations. If none of the authentication packages indicates a successful login, the login process is aborted. It may be added that it is the Lsass that determines which account database to be used; local SAM, domain SAM or Active Directory. This thesis concentrates on the local SAM. It is also the Lsass that enforces the local security policy, for example checking the user permissions. A user can have regular user privileges, administrator privileges or system privileges. A user with system privileges is even more powerful than an administrator user, and may be compared to the root or super-user (su) in Unix [RS04].

As already mentioned, the Windows Vista boot process differs from the Windows NT boot process, and some of the differences will now be highlighted. Also for the Windows Vista boot process the BIOS reads the master boot record (MBR) and transfers control to the MBR's code. Then the MBR transfers control to the code that loads the operating system. For the Windows NT boot process the primary application that is responsible for loading Windows has been Ntldr. Windows Vista instead introduced winload.exe, which is the Windows operating system loader. Each version of Windows Vista and Windows Server 2008 that is installed on a computer has its own instance of winload.exe. The operating system loader creates the execution environment for the operating system and also loads the Windows Vista kernel, Hardware Abstraction Layer (HAL) and boot drivers into memory [25].

Windows Vista also introduces Boot Configuration Data (BCD). This new data store serves essentially the same purpose as boot.ini. However, BCD abstracts the underlying firmware and provides a common programming interface to manipulate the boot environment for all Windows-supported computer platforms. In addition to the BCD, Windows Vista introduces Windows Boot Manager (Bootmgr), which is an application that controls boot flow. With a multi boot system, the boot manager displays an operating system selection menu. Actually the Windows Boot Manager reads the BCD and displays the operating system selection menu. The Windows Vista operating system loader (winload.exe), which is already mentioned above, is invoked by the Windows Boot Manager. Another application is the winresume.exe, which is the Windows Vista resume loader. Each version of Windows Vista and Windows Server 2008 that is installed on a computer has its own instance of winresume.exe. The resume loader restores Windows to its running state when a computer resumes from hibernation [25].

At last, it may be added that you may enter the Windows Registry be entering *regedit* in the Windows Command Console (cmd.exe)

2.2.2 Linux

When a Linux system boots up, usually a boot loader such as LInux LOader (LILO) or GRand Unified Boot loader (GRUB) starts the kernel. The Linux kernel locates and mounts the root file system, and then initiates a program called */sbin/init.* Init will start the rest of the system. The *init* process, becoming process number one, determines which runlevel and processes to start by looking into a file called */etc/inittab* [26]. A runlevel is defined by [27] as:

A runlevel is a preset operating state on a Unix-like operating system.

This approach is inherited from the famous Unix System V (SysV) both acting as an operating system and as an Unix Standard with the goal of uniting the many Unix derivatives. SysV is also known as Unix version 7. In many Linux distributions the runlevel can be changed in the boot loader. This is done either through a menu or, by appending specific arguments, such as *single*, in the kernel line of the boot loader.

An event handling mechanism listens for events such as *control-alt-delete* typed with the keyboard, initiating a shutdown, or the change of runlevels. An *rcx* file, where *x* is a number from 0 to 6 or the character *S*, is initiated with the corresponding default or user chosen runlevel. As shown in Figure 2.1, Fedora has runlevel 5 as default, and Ubuntu has 2 as default.

If for instance a system is booting at runlevel 2, */etc/event.d/rc2* initiates a program called */etc/init.d/rc* with parameter 2, referring to the user's runlevel. The location of *rc* varies from one Linux system to another, but the functionality is usually the same. The *rc* program runs scripts located in the directory */etc/rc.d/rcx.d* where *x* corresponds to the runlevel. Mark Allen [26] describes the program *rc*'s main tasks as follows:

1. It gets the previous and current system runlevels.
2. If the word confirm is in the file /proc/cmdline it sets up to run the scripts below in user confirmation mode.
3. All kill files (files whose first letter is K) in the subdirectory /etc/rc.d/rc3.d (assuming the previous runlevel was 3) are run. The parameter stop is usually passed on the command line to the kill script.
4. All startup files (files whose first letter is S) in the subdirectory /etc/rc.d/rc5.d (assuming the current runlevel is 5) are run. The parameter start is usually passed on the command line to the kill script.

In a Linux system, 8 runlevels are used as default. It is possible to add more runlevels, but in most distributions 8 are sufficient. These runlevels are 0-6 and S or s. Runlevel S is usually not meant to be used directly by a user, but more for the scripts that are executed when entering runlevel 1. Each of the runlevels has different purposes, initiating different scripts and processes. The *init* process can only run one of these levels at a time. Runlevel 0,1 and 6 are reserved and identical in most distributions [26]. The runlevels are often configured differently in various operating systems. In Table 2.1, the different runlevels, except from *S*, are described for Ubuntu, Fedora and standard Linux (generic).

As shown Table 2.1, single-user and multi-user mode are mentioned. Single-user mode is much like a *rescue*, or *trouble-shooting* mode. In this mode no daemons/services are started. This mode is included to allow users to fix whatever made the transition to rescue mode necessary [28].

To authenticate users a login program is used. When the system has booted and the some of the init processes have finished, a process called *getty* invokes the login program. Mark Allen [26] identifies the 3 important functions performed by *getty*:

> *1. Open tty (Terminal Type - Unix terminal interface) lines and set their modes*
> *2. Print the login prompt and get the user's name*
> *3. Begin a login process for the user*

The process *getty* initiates a login process with the username prompted by the user as an argument. In the manual of login [29] two security features are described: Firstly, echoing is disabled to prevent revealing the password. Secondly, only a small number of password failures are permitted before login exits and the communications link is severed.

Fedora and Ubuntu Server are both popular Linux distributions. Fedora boots as described above, and Ubuntu Server has some minor differences. The scripts listed above, located in */etc/rc.d/rcx.d/*, were mentioned as Linux's initial startup scripts. On the other hand, the program *rc* and the directories *rcx.d* are located directly under the folder */etc* in Ubuntu Server 8.10. In newer versions of Ubuntu the file *inittab* is replaced by a file called */etc/event.d/rc-default.* This file contains a script that first checks if the word *single* exists in */proc/cmdline*, a file containing parameters provided at boot. These parameters can be edited in the boot loader menu. If the word *single* is present, the program *telinit* with parameter *S* will run changing the runlevel to *Single-user mode* as described earlier. If the word *single* is not added during boot, the script *rc-default* will look for an *inittab* file and if it exists change the runlevel to what it contains. If the *inittab* file is non-existing, *telinit* will be initiated with parameter 2, Ubuntu's default runlevel.

2.2.3 BSD

Both Berkeley Software Distribution (BSD) and Linux originate from the late 1960's operating system Unix, developed by a group of AT&T employees at Bell Labs [RT74]. A more detailed view of the originating of Linux and BSD is shown in Lévénez's excellent Unix time line [31]. Although BSD and Linux evolved in different directions, many similarities can be identified.

Run-level	**Generic**	**Fedora**	**Ubuntu**
0	System Halt	System Halt	System Halt
1	Single-user mode	Single-user mode	Single-user mode
2	Basic multi-user mode (without networking)	User definable (Unused)	Full multi-user mode (Default)
3	Full (text based) multi-user mode	Multi-user mode	Same as runlevel 2
4	Not used	Not used	Same as runlevel 2
5	Full (GUI based) multi-user mode	Full multi-user mode (with an X-based login screen) (Default)	Same as runlevel 2
6	System Reboot	System Reboot	System Reboot

Table 2.1: The different runlevels initiated by *init* [28] [30].

The process of booting a BSD system varies from one BSD system to another. While the two Linux distributions mentioned in Section 2.2.2 both use the same kernel, Mac OS X and FreeBSD use their own kernel. Mac OS X and FreeBSD have related kernels but they are not identical. Their kernels are unrelated to the Linux kernel. The systems might look the same mainly because they often have the same application level programs such as *vi* and *Emacs*, and use *bash* and *X11* window systems.

Although FreeBSD and Mac OS X have different kernels, much of the boot process looks the same. As with Linux, the boot process of BSD utilizes a boot loader to start the kernel. According to Smith [Smi03], when the kernel has loaded, no programs can bypass the kernel to directly address the computer's hardware.

Smith [Smi03] presents two startup schemes. One scheme is the *BSD startup method* used by most BSD systems and the other is the *SysV startup scripts* used by Linux systems. The BSD approach uses a few large scripts, whereas the SysV approach uses many smaller scripts. SysV also supports runlevels described in more details in Section 2.2.2.

The boot processes of Mac OS X and FreeBSD have some similar, but also many different steps. The process of booting FreeBSD and Mac OS X is thus described in more details in the separate sections below.

2.2.3.1 FreeBSD

Boeger and Tominaga [BT04] present a three-phase bootstrap process. This three-phase bootstrap process consists of three programs *boot0*, *boot1* and *boot2*, often also referred to as *Stage 1 boot loader*, *Stage 2 boot loader* and *Stage 3 boot loader*. After the BIOS has completed its system tests, the first track from disk0 is loaded into the memory. This track is the MBR and is the boot0 program. Boot0 is the first program to be loaded and executed by the computer, and it loads and executes the boot1 program which knows enough to read disk slice information and load boot2. When the computer has loaded the boot2 program, the computer has the ability to either boot the system directly or load a loader program. The loader program gives the user the ability to interact with the boot process to perform system recovery and maintenance, and to load and unload the kernel or kernel modules.

As with Linux, FreeBSD's kernel executes first a program called */sbin/init*. During the boot, parameters can be specified to be passed to the */sbin/init* program. The program *init* use these values to determine whether it should bring up the system to single-user or to multiuser operation. In single-user operation, */sbin/init* forks a process that invokes the standard shell, */bin/sh* [MKM04]. In Linux and System V systems scripts corresponding to their runlevel are initiated. This is handled much easier in FreeBSD according to [BT04]. FreeBSD executes the following *rc* scripts found in */etc*:

- /etc/rc - main script and first to be called. Mounts the file system and then runs all the needed rc scripts.
- /etc/rc.atm - used to configure ATM networking
- /etc/rc.devfs - set up /dev/ permissions and links
- /etc/rc.diskless1 - first diskless client script
- /etc/rc.diskless2 - second diskless client script
- /etc/rc.firewall - used for firewall rules
- /etc/rc.firewall6 - used for ipv6 firewall rules
- /etc/rc.i386 - Intel system specific needs
- /etc/rc.isdn - ISDN network settings
- /etc/rc.network - IPv4 network settings
- /etc/rc.network6 - IPv6 network settings
- /etc/rc.pccard - for laptop pc card controllers
- /etc/rc.serial - set up serial devices
- /etc/rc.sysctl - used to set sysctl options at boot time
- /usr/local/etc/rc.d/ - general directory with custom start up scripts

Two files, */etc/rc.conf* and */etc/defaults/rc.conf*, where the first is usually edited by system administrators, control which user-level services are started at boot time. Each of these files is loaded by the system startup scripts when they execute [MKM04]. An administrator can for instance enable SSH at login by appending *sshd_enable="yes"* in the */etc/rc.conf* file.

When these scripts are completed successfully, */sbin/init* invoke other system programs such as */usr/libexec/getty* handling the login procedure, also mentioned in Section 2.2.2. The *getty* program reads the login name and invokes the */usr/bin/login* program to complete the login process. The login program interacts with password files and databases to authenticate the user. This is described more in detail in Section 2.3.3.

In FreeBSD several login methods are supported. The methods are presented and accessed in many ways, both graphical and text-based, remotely and locally, but most use the same authentication tools at their core (*/usr/bin/login*).

2.2.3.2 Mac OS X

This section will look closer into the boot process of *Mac OS X v10.5 Leopard*. A few important boot changes occurred from *Mac OS X v10.3 Panther* to *Mac OS X v10.4 Tiger*. These changes will also be addressed below.

The Mac OS X boot process is quite unique and differs from traditional Linux systems with */etc/inittab*, presented in Section 2.2.2, by using BSD-like startup sequences sandwiched between a *Mach* foundation and the *Aqua* user interface [RJR08]. *Mach* is a microkernel operating system also descended from Unix. The Mac OS X kernel, *xnu*, is a hybrid of Mach and BSD.

When a machine is booted, the firmware BootROM is activated. BootROM, a part of the computer's hardware, has two primary responsibilities: it initializes the system hardware and it selects an operating system to run. Open Firmware on PowerPC Macs and Extensible Firmware Interface (EFI) on Intel Macs are loaded by the BootROM. According to [33], Intel designed EFI to modernize firmware technology in today's computers in order to move past the limitations of a legacy BIOS which is found in most computers running Windows systems.

The EFI or Open Firmware initiates the rest of the hardware and hands off control to the boot loader. Two boot loaders are common in Mac OS X, the *BootX* for PowerPCs and *boot.efi* for Intel based machines, both located in */System/Library/CoreServices*. The boot loader looks for an up-to-date kernel that has been pre-linked to all required kernel extensions (drivers, also known as *kexts*). When the kernel extensions are successfully loaded, the boot loader hands off the full control to the kernel (*xnu* located at */mach_kernel*). Before any processes can be initiated, *xnu* initializes all the data structures needed to support Mach and BSD. The kernel initiates an I/O Kit, a device driver framework, which connects the kernel with the set of extensions that correspond to the machine's hardware configuration. After the hardware is configured correctly, the kernel finds and mounts the root file system [RJR08].

At this point of the boot process a few important changes were made from Mac OS X v10.3 to v10.4. In Panther (v10.3) and earlier the first process loaded by the kernel was *mach_init*. The *mach_init* process launched the BSD *init* process determining runlevels and initiating */etc/rc** scripts as described in Section 2.2.3.1. Beginning with Tiger (v10.4) both of these processes were replaced by */sbins/launchd*. The manual for the startup process *rc* still exists with the following message [34]:

> *Prior to Mac OS X 10.5, the rc script was used to bootstrap the OS. As of Leopard, the system is self-bootstrapped via launchd(8) which uses the launchctl(1) bootstrap subcommand to read in launchd jobs from the standard locations. For compatibility reasons, the rc.local script still continues to work.*

The program *launchd* manages two types of services, *launch daemons* and *launch agents.* The launch daemons are services that can run even when no user is logged in, and launch agents are services which can run on behalf of a logged-in user and access files in that user's home directory. The service type *launch daemons* cannot connect to the window server, thus cannot display any Graphical User Interface (GUI). The program *launchd* also initiates *SystemStarter*, a program used to start programs located in */System/Library/StartupItems* and the */Library/StartupItems.* Although */etc/rc** scripts are no longer used, as described above, SystemStarter runs every command in */etc/rc.local* at system startup and */etc/rc.shutdown.local* at system shutdown.

When SystemStarter has completed its initial scripts, a program called *loginwindow* is initiated. This program starts the login process which authenticates users and sets up their user sessions [RJR08]. The *loginwindow* program displays a login panel containing fields where the user can provide the login information. When the login credentials have been submitted, *loginwindow* authenticates the user. If a user types >*sleep*, >*restart* or >*shutdown* as username in the graphical user interface the system will sleep, restart or shutdown. If the user types >*console* the user will be turned over to a text-based login prompt. This text-based login prompt uses the program */usr/libexec/getty* to handle the login process. The program *getty* can be identified in other systems such as FreeBSD, Ubuntu and Fedora, as mentioned in Section 2.2.2 and 2.2.3.1. Standard shells *csh* and *sh*, do not fork before executing the login utility. Thus, when *getty* handles the login process it first initiates the program */usr/bin/login.* This program then prompts for username and password and interact with several files described more in detail in Section 2.3.3.2.

When the new releases of Mac OS X with support for EFI came, more and more reports of successful installations of Mac OS X on non-Apple Intel-based computers appeared. The Intel-based versions of Mac OS X is released with a built in interaction with Trusted Platform Module (TPM), a computer chip embedded inside all Intel-based Macs to prevent the Intel-based versions of Mac OS X from running on non-Apple hardware [35]. During the installation of Mac OS X, Mac OS X interact with the TPM. If Mac OS X finds out that the TPM does not exist, Mac OS X refuses to install or run. Hacks have been released on the Internet to bypass this TPM requirement. The hacks are either embedded into the Mac OS X installation disk or loaded before the installation process is initiated. The latter case will allow the user to use a retail version of the Mac OS X installation disk. The installed Mac OS X system is then led to believe that a TPM exists.

2.3 Password Handling

In this section we will highlight the password handling related to the login procedure for Windows-, Linux- and BSD based systems.

2.3.1 Windows

This section will present what happens to a password from when it is created to when the corresponding password hash value is stored in the password file on a Windows operating system. Explanations of different hashing techniques that are used for this password handling will be given.

2.3.1.1 SAM

When a password is created in order to protect an user account on a Windows operating system, the *Security Accounts Manager (SAM)* file is where the password hash is stored. When users enter their password to log into a system, the password is hashed and the result is compared to the hash value already stored in the SAM file on the computer. SAM is a database stored as a registry file in Windows and is located at \Windows\System32\config\SAM or \WINDOWS\system32\config\SAM, depending on which Windows operating system that is used. However, this folder is locked to all accounts including the Administrator while the computer is running. The only account that can access the SAM file during operation is the *System* account, which is the account with the highest level of privileges in a Windows operating system [Tod07]. It may be added that the Windows registry is a database which stores settings and options for Windows operating systems. It for example contains information and settings for hardware, operating system software and per-user settings.

As will be described below, the password hashes included in the SAM file is obfuscated. The other fields such as username and account description appear in plaintext and Unicode format. As a result of this obfuscation, there exist tools that are able to dump the content of the SAM file. Figure 2.2 shows how the tool *pwdump* [36] represents the dumped content, and this format is often used by similar utilities. Figure 2.2 is based on a figure in [Tod07]. From the figure it can be seen that username, user RID, the LM hash and the NT hash may be included in the entry. The RID (Relative ID) is a part of the SID (Security identifier). SID is a string of alphanumerical characters that is for example assigned to each user. In other words, the SID is a unique ID number that a computer uses to identify the user. The operating system uses the SID to determine the access privileges of users and groups. The RID, which is a part of this unique ID number, indicates which computer or network that created it, whether the user is a normal user, a guest or an administrator, and also in what order the user's account was created [Tod07] [37].

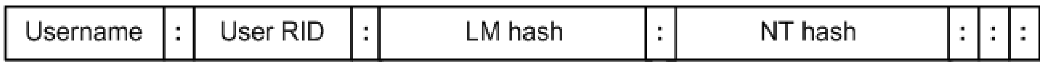


Figure 2.2: *pwdump* password entry format.

2.3.1.2 Hashing of passwords

Microsoft does not give detailed information on how these hashing techniques are carried out but due to reverse engineering people have been able to find out how these techniques work, and an explanation will be presented here.

In the late 80's Microsoft designed what they have called *LAN Manager (LM) hashing* that should prevent the passwords from being stored on the computer in plain text. As already described above, the hash of the passwords is stored in the SAM file. Although LM hashing was considered sufficiently secure when it was introduced, this is not the case today. Actually Microsoft addressed this problem already in the early 90's by coming up with a new and more secure hash algorithm called *NT (New Technology) hashing*. We will give a description of these two hash algorithms below. The problem was that Microsoft decided to continue with LM hashing to preserve the backward compatibility. The user account password on a machine running for example Windows XP is hashed using both the insecure LM algorithm and the more secure NT algorithm. In Windows XP both the LM hash and the NT hash is stored together on the computer. This is a big advantage for the hacker since he will propably ignore the NT hash and focus on the insecure LM hash [38].

Before we go into detail about the different hashing techniques some clarification has to be done. Both LM hash and NT hash are hash values that are stored locally on the computer, more specific in the SAM file. NT hash is also called NTLM (New Technology LanMan) hash or Unicode hash. There also exist acLM, NTLMv1 and NTLMv2 *authentication protocols* used with the Server Message Block (SMB) protocol that is a typical challenge-response authentication protocol. These authentication protocols use the LM and NT hash that is described in this section, for authentication procedures. These authentication protocols are out of scope for this thesis and the focus has been on the locally stored LM and NT hash values.

LM hashing As an introduction to how LM hashing works, a brief explanation of American Standard Code for Information Interchange (ASCII) will be given. ASCII is a standard 7-bit code, representing 128 different characters with values ranging from 0 to 127, which was proposed by American National Standards Institute (ANSI) in 1963. Out of these 128 different characters only 95 characters are printable, as will be described later. The standard printable ASCII characters can be seen in Listing 2.1. As computers operate on numbers, 1s and 0s to be specific, a way to convert characters to and from numbers is needed. This is made possible by the use of the ASCII codes where each character is assigned a unique code or value [39].

The number of written symbols used may exceed the limited range of the ASCII code. All Norwegian characters are for example not defined in the standard ASCII chart. As a result an extra bit can be added as the leftmost bit, the most significant bit. In this way the ASCII code can be stored in an 8-bit byte which is a very common data unit for computers to operate on. So, if the extra bit is needed to represent a character it is set to 1, and when it is not in use it is set to 0. It is important to emphasize that this is an extension to the standard ASCII chart and not a new encoding. As already mentioned, the extra bit may be used to represent additional special, mathematical, graphical and foreign characters. When this extra bit is used, it extends the character set with 128 characters, ranging from 128 to 255. This extra set of characters is referred to as *The Extended ASCII Character Set* and is not defined by ASCII. This means that Windows for example have their own extension to the ASCII table that may not exist in other operating systems. There exist various extensions to the ASCII codes but giving a description of these is out of scope of this thesis [40] [39].

The LM hash construction is based on the block cipher named *DES*. As can be seen from the simplified illustration of DES in Figure 2.3 the inputs are a 56 bit key and a 64 bit plaintext. We are not going into detail about how DES is working but it produces a 64 bit ciphertext.

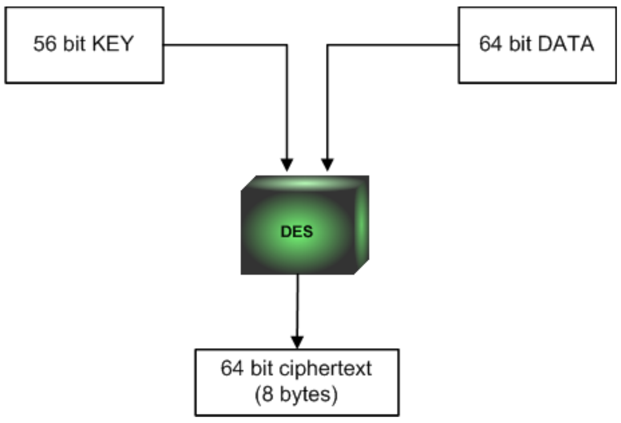


Figure 2.3: A simplified overview of DES.

Now we will give an explanation of how the LM hash, which is based on the block cipher named *DES*, is constructed. The DES algorithm is typically used for encryption, and not for generating hashes. However, in this case, DES is used with a fixed-length string to generate a fixed-length result that depends on the secret user password. Hence, it can be considered as similar to hash functions, and both Microsoft and the security community have adopted the name LM hash. As Figure 2.4, which is based on a figure in [41] and a figure in [Jaa00], illustrates you start out with a password of 14 characters. If the password is less than 14 characters long the password is padded or extended with null characters to get 14 characters (14 bytes) in length. Then the password is split into two 7 character (7 bytes) segments and the characters in the two segments are converted to uppercase. The conversion of the characters to uppercase is labeled *UpCase* in Figure 2.4. Then each 7-byte segment is converted to a bit stream consisting of 56 bit (7 byte) each. And instead of 7 blocks with 8 bit each we now turn it into 8 blocks with 7 bit.

Then, as can be seen from the figure, a *parity bit* is added after every 7 bit. This parity bit must not be confused with the extra bit used for the Extended ASCII Character Set mentioned above. A parity bit is a bit added to ensure that the number of bits set to 1 is always even or odd. The parity bit is normally used as an error-detecting code. In the construction of LM hash an odd-parity bit is used, meaning that the parity bit is set to 1 if there are even numbers of bits set to the value 1 and thus making the number of 1's an odd number. If there are already an odd number of bits set to 1 then the parity bit remains 0. This extra bit results in 8 blocks with 8 bit in each block, as illustrated in Figure 2.4 and we have now two DES keys, one from each 7 character segment, consisting of 64 bits each.

Although the input key for DES is 64 bits, the actual key used by DES is 56 bits. The least significant bit or the right-most bit in each byte is, as explained above, a parity bit. These parity bits are ignored and only the 7 most significant bits, the 7 left-most bits, in each byte is used. This results in two keys consisting of 56 bits. The parity is, as already explained, used as error-detecting code during transmission and is therefore not included when constructing the LM hash [42] [43]. This 56 bit key is together with the *fixed* 8 character plaintext ASCII string *KGS!@#$%* used as input to DES, which is shown as a *black-box* labeled *DES* in Figure 2.4. Note that there are two separate DES black-boxes for each of the two 7 character segments. DES then produces a 64 bit ciphertext and the ciphertext from each of the two DES black-boxes are concatenated into a 128 bit (16 bytes) *LM hash* [Jaa00].
This sums up the process of constructing the LM hash [42]:

- If the password is less than 14 characters long the password is padded with null characters to get 14 characters (14 bytes) in length.
- The password is split into two 7 character segments (7 bytes).
- Lowercase letters in each of the segments are converted to uppercase. Other characters remain unchanged.
- These two 7 character segments are used to create two DES keys. This process is explained above.
- Each of these keys is together with the *fixed* ASCII string *KGS!@#$%* used as input to DES. The result is two 64 bit (8 bytes) ciphertext values.
- The two 64 bit ciphertext values are concatenated to form the *LM hash*, which then consists of 128 bits (16 bytes).

The first weakness with the construction of the LM hash is that the user's password is converted to uppercase. This is obviously not as strong as using mixed-case letters because it decreases the number of permutations available when a password is created and then makes it easier for a hacker to crack the password. In other words, the LM hash is case-insensitive, which means that is does not matter if the created password is entered with uppercase or lowercase letters. For example, if the password is written *password* when it is created, permission will be granted for subsequent login attempts regardless of that you are writing *password* or *PASSWORD*. The authors of this thesis experimented with this on a computer running Windows 98. A further investigation of the case-insensitivity is carried out below.

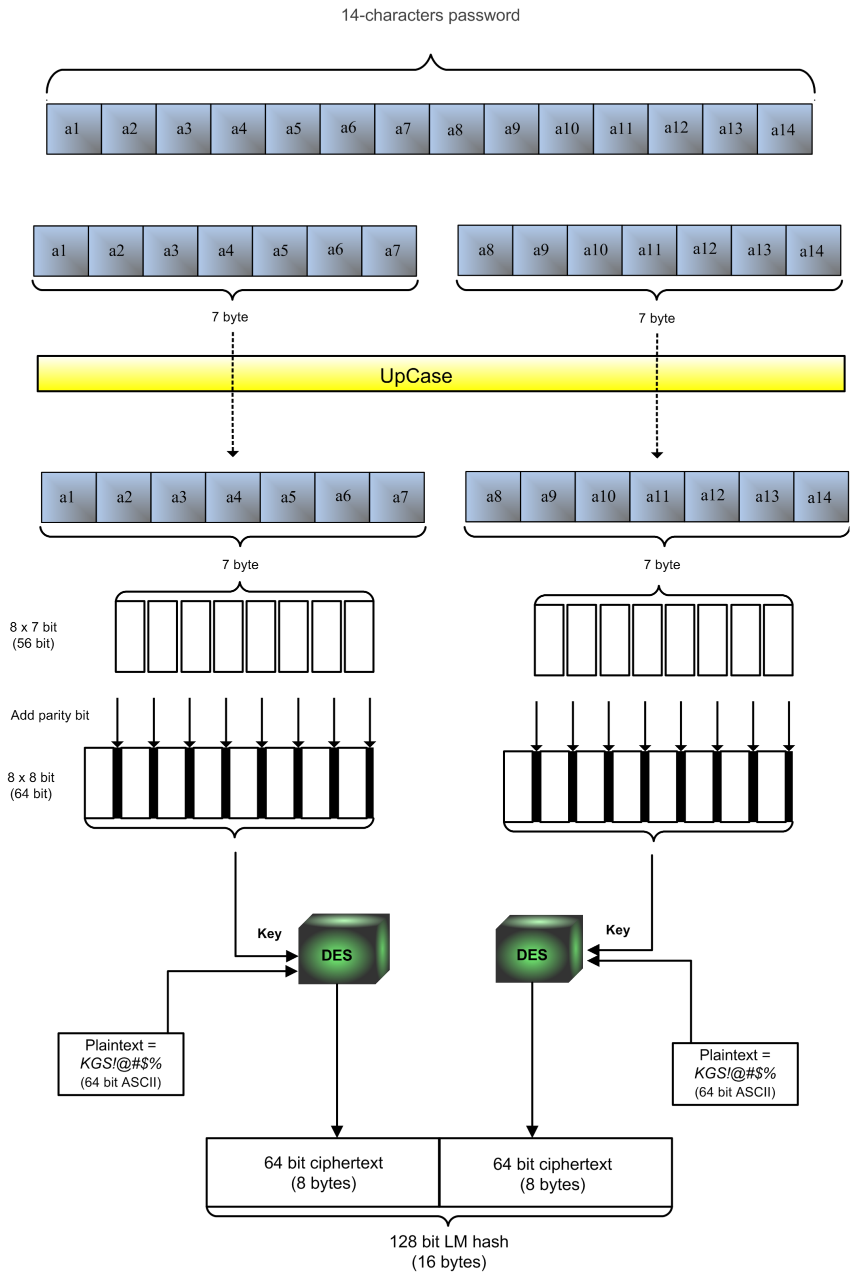

14-characters password
a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14
7 byte
7 byte
UpCase
8 x 7 bit (56 bit)
Add parity bit
8 x 8 bit (64 bit)
Key
DES
Plaintext = KGS!@#$% (64 bit ASCII)
64 bit ciphertext (8 bytes)
128 bit LM hash (16 bytes)

Figure 2.4: The construction of the LM hash.

Another weakness with the LM hash construction is that the password is split into two 7 character segments, still with the use of uppercase. As an example the password *LMisNotPeRfeCt* would turn into *LMISNOT* and *PERFECT*. A password that is shorter than 14 characters will, as already described, be padded with null characters to obtain a length of 14 characters. A general rule is that the longer a password becomes, the more secure it gets. In this case it does not matter if you have a long password since it is split into two 7 character segments. As discussed in Section 9.5.1 and in Section 9.2, a seven character password is not very secure. As a consequence, the hacker now just needs to concentrate on cracking two separate 7 character passwords and with today's computer power this can be done by brute-force or by a dictionary attack in a relatively short time. This is also explained below.

Brute-force is when you try every possible combination to guess the correct password and a dictionary attack is when a precomputed list of possible solutions is used to reveal the correct password. An example is to use so called *rainbow tables* which exploit the weakness of LM hashes. In this case the rainbow tables will consist of precomputed keys of LM hashes for every possible combination of seven uppercase characters arranged in chains. A tool that uses rainbow tables to recover passwords is Ophcrack [44], which is described in Section 2.4.2.1. These kinds of tools use rainbow tables to try to crack the weak LM hash. A more detailed explanation of rainbow tables can be found in Section 2.1.2.3.

A reason why it is possible to use rainbow tables to crack the password is that the LM hash does not include a *salt*. A salt should consist of random bits to increase the randomness of the hash values. Salting a password complicates an attack because a given password should then produce different hashes every time if a different salt is used when producing the hash value [38] [41]. This highlights another weakness with the construction of the LM hash, which uses the same fixed and known ASCII string as input for every LM hash that is produced. This means that two matching plaintext passwords will have matching hash values and the LM hash is therefore susceptible to known-plaintext cryptanalysis. Instead of making the task of cracking the password easier for the hacker by using this fixed string, a solution could have been to change this string for each LM hash that is constructed, e.g. use it as a salt. By doing this it would be very hard to for example use rainbow tables to crack the password. It should be noted that the construction of the LM hash, which do not include a salt, makes it easy to *crack* the password using rainbow tables but the task is not that easy if a hacker wants to *guess* the password. This is because the login procedure for the operating system is not restricted to uppercase. The *UpCase* process, as shown in Figure 2.4, is only carried out when the LM hash is constructed [Tod07].

As mentioned above, *standard* ASCII represents each character with seven bits which results in a total of 128 different characters (2^7 bits = 128 possible values). The first 33 characters do not carry printable information as they are only used as control characters. This leaves us with 95 characters where 94 of them are printable and 1 of them is the space-character. From now on we include the space-character in the printable characters such that there are 95 usable ASCII characters, which are listed in Listing 2.1. Listing 2.1 shows the standard ASCII characters defined by ANSI and therefore exclude all existing ASCII *extensions*. Listing 2.1 is based on the ASCII character overview given in [45].

```
space ! " # $ % & ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @
ABCDEFGHIJKLMNOPQRSTUVWXYZ [ " ] ^ { } _ ` abcdefghijklmnopqrstuvwxyz {
| } ~
```

Listing 2.1: The 95 standard printable ASCII characters.

As already mentioned, when the LM hash is constructed, the password is divided into two 7 character segments. These segments are hashed separately, and may then also be attacked individually. While there are

$$95^{14} = 4876749791155298590087890625 \tag{2.7}$$

different passwords available when using *one* segment consisting of all the 14 characters there are only

$$95^{7} = 69833729609375 \tag{2.8}$$

different passwords available when splitting the password into two 7 character segments. Instead of finding the password among 95^{14} different passwords the hacker just need to find two passwords, one for each segment, among a total of

$$2 * (95^{7}) = 139667459218750 \tag{2.9}$$

different passwords. This shows that splitting the password into two segments makes the task much easier for a hacker to launch a brute-force attack as the set of possible passwords is much smaller [41].

In addition, each segment is converted to uppercase. This means that instead of 95 possible ASCII characters there are only 95-26 = 69 possible ASCII characters left, as there are 26 lowercase characters. This result in

$$69^{7} = 7446353252589 \tag{2.10}$$

different passwords for each segment. The total amount of possible passwords for the two segments is then reduced to

$$2 * (69^{7}) = 14892706505178 \tag{2.11}$$

Equation (2.11) shows that a hacker, in worst case, needs to check less than $1.5 * 10^{13}$ different passwords or hashes because the password is split into two segments, and therefore with today's computer power it should be a quick task to crack the LM hash. It is important to emphasize the fact that these calculations are based on the 95 *standard* printable ASCII characters and does not take into account various ASCII extensions that increases the number of characters available [41].

As already mentioned above, the more characters a password contains the more secure is the password. At least this is the common sense. But by using LM hash this may not actually be true. Consider that we increase our password length from seven to eight characters to increase the strength of the password and that we continue to use LM hashing. Now the first segment will consist of the seven first characters and the second segment will consist of the eighth character padded with six null characters. The hash of the second segment is now actually the hash of a single character password and is quite fast to brute-force. Now you know that the original password is eight characters long and you know the last character. This may ease the process of guessing the password. It may also help if you can find out what language is used [38].

A simple way to avoid the generation of a LM hash is to use passwords that are longer than 14 characters. Under these circumstances Windows XP does not generate a LM hash at all but only stores the more secure NT hash. There is also a possibility to manually disable the LM hashing on for example Windows XP machines. An overview on how to disable the LM hashing is presented in Appendix C. This is done by default in Windows Vista [38].

NT hashing Before we start giving an explanation of how the NT hash works, it is necessary to give a brief introduction to *Unicode*, which is a universal character encoding standard used for representation of text for computer processing. As already described, the standard ASCII character set, also an encoding standard, consists of 128 different characters. Each character is represented with a 7-bit code. This is not sufficient to include all the characters used in existing languages. This is why there exist 80 ASCII extensions. Unicode, developed by ISO and the Unicode Consortium, is a successor to ASCII and other ASCII extensions in an attempt to define codes for characters used in all the major languages written today, no matter what platform or program that is used. The Unicode Standard Version 5.1, which was released in March 2008, provides codes for in all 100,713 characters and symbols. This shows that the Unicode Standard covers more characters than the ASCII Standard does. We do not intend to go into more details about the Unicode Standard as this is not the focus of this thesis, but it should be mentioned that the Unicode Standard defines three encoding forms and that the Unicode Standard includes the standard ASCII codes [46]. As already mentioned, the NT hash is also known as Unicode hash, and this is because it supports the full Unicode character set.

NT hashing is considered more secure than LM hashing. The reason is that the password can be up to 128 Unicode characters when producing the NT hash, as opposed to the LM hash where only passwords up to 14 ASCII characters are possible. In addition, the password is not split into two segments which is the case when constructing the LM hash. In both cases, the result is a 128 bit hash value, but the difference is that the whole NT hash depends on all the 128 characters instead of being a concatenation of two hashes that depend on 7 characters each. This makes the NT hash harder to crack than the LM hash since you have to find a password of 128 characters instead of two passwords of 7 characters each. With NT hashing the password is also case sensitive which leads to an increased number of possible hash values compared to LM hashing

[Tod07]. It should be added that when both LM and the NT hash are enabled, only the NT hash is used when authenticating users at login. This means that it is not the same if the user enters *password* or *PASSWORD* which was the case for the LM hash, as the NT hash is case sensitive. When both the LM hash and the NT hash are enabled, the LM hash is only used to support backward capabilities. An attacker would of course exploit the weakest link, which in this case is the LM hash.

Figure 2.5 illustrates the construction process of the NT hash, and as can be seen from this figure, the NT hash is generated using the MD4 algorithm. First, the user password that may contain up to 128 characters is converted to zero terminated Unicode. Zero terminated Unicode is a Unicode string that has NULL (0x00) as the last character to indicate the end of the string. The Unicode string is then used as input to the MD4 algorithm, which then produces the 128-bit NT hash [Tod07].

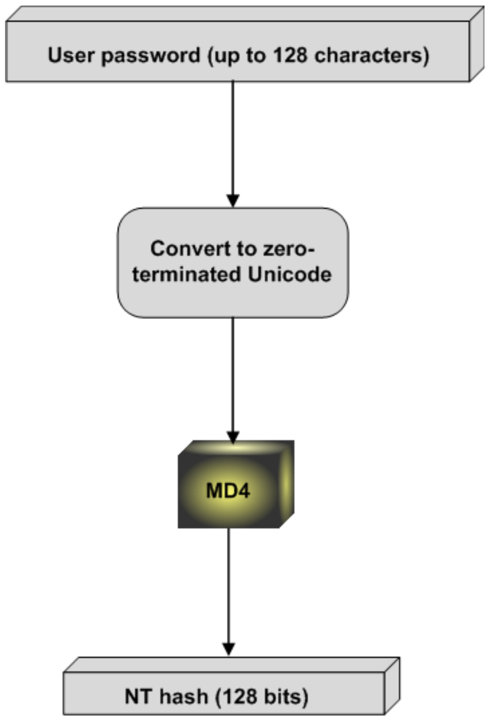

User password (up to 128 characters)
Convert to zero-terminated Unicode
MD4
NT hash (128 bits)

Figure 2.5: The construction of the NT hash.

This sums up the process of constructing the NT hash [Tod07]:

- The password is converted to a zero-terminated Unicode string.
- The Message-Digest algorithm 4 (MD4) algorithm is used to create a hash of the Unicode password. This results in a 128-bit hash value, which is called the NT hash.

Even though NT hashing is considered more secure than LM hashing the MD4 algorithm, which NT hashing is based on, has been analyzed by cryptographic researchers and has been proven to have a number of security weaknesses. MD4 is for example not collision free, which means that two or more plaintext messages can generate the same hash. This makes the NT hash susceptible to brute-force and dictionary attacks. There exist some MD4 specific attacks as well. As for LM hashing, the generation of the NT hash also lacks the use of a salt. This contributes to the exposure of attacks on the NT hash algorithm [Tod07].

Table 2.2 shows which of our Windows versions that store an LM hash, an NT hash or both *by default*.

Operating System	**LM hash**	**NT hash**
Windows XP	Yes	Yes
Windows Vista	No	Yes
Windows Server 2008	No	Yes
Windows 7	No	Yes

Table 2.2: Windows operating systems that stores LM hash or/and NT hash by default.

Password Hash Obfuscation To additionally protect password hashes while being stored in the SAM file, Windows applies additional obfuscation. The obfuscation is done by the use of an additional round of DES encryption and the keys that are used are derived from the user RID (Relative Identifier). A description of RID can be found in Section 2.3.1.1. The password hash obfuscation, together with the LM and NT hash creation, have not been published in detail by Microsoft, but security researchers have been able to do some reverse engineering to get an understanding of what is happening. Note that this obfuscation is applied to both the LM hash and the NT hash, which are both described in details above. An illustration, which is based on a figure in [Tod07], of how the password hash obfuscation is done can be seen from Figure 2.6.

The DES encryption used for password hashes requires two 56-bit (7 bytes) keys, *k1* and *k2*, that are calculated individually for each user. These two keys are both functions of the *4 byte* RID value. The first 4 bytes of *k1* are equivalent to the 4 bytes of the user RID, and the last 3 bytes (byte 5-7) of k1 are the same as the first 3 bytes of k1, and therefore also equal to the first 3 bytes of the RID. The first 4 bytes of *k2* consist of the fourth, the first, the second and then the third byte of the user RID, and the last 3 bytes (byte 5-7) are the same as the first 3 bytes of k2 (therefore also the same as the fourth, the first, and the second byte of the user RID). The keys *k1* and *k2* are included in Figure 2.6 [Tod07].

As already described, both the LM hash and the NT hash are 128-bit (16 bytes) values. Even though Figure 2.6 only shows the obfuscation process of the NT hash, the same scheme applies if the LM hash is used. The only difference is that the result is an obfuscated LM hash instead of an obfuscated NT hash, as a LM hash is used instead of a NT hash. In Figure 2.6 the 128-bit NT hash is first divided into two halves of 64 bits (8 bytes) each. Each of the two halves is then encrypted separately. The first half is encrypted using DES with *k1* as the key, and the second half is encrypted using DES

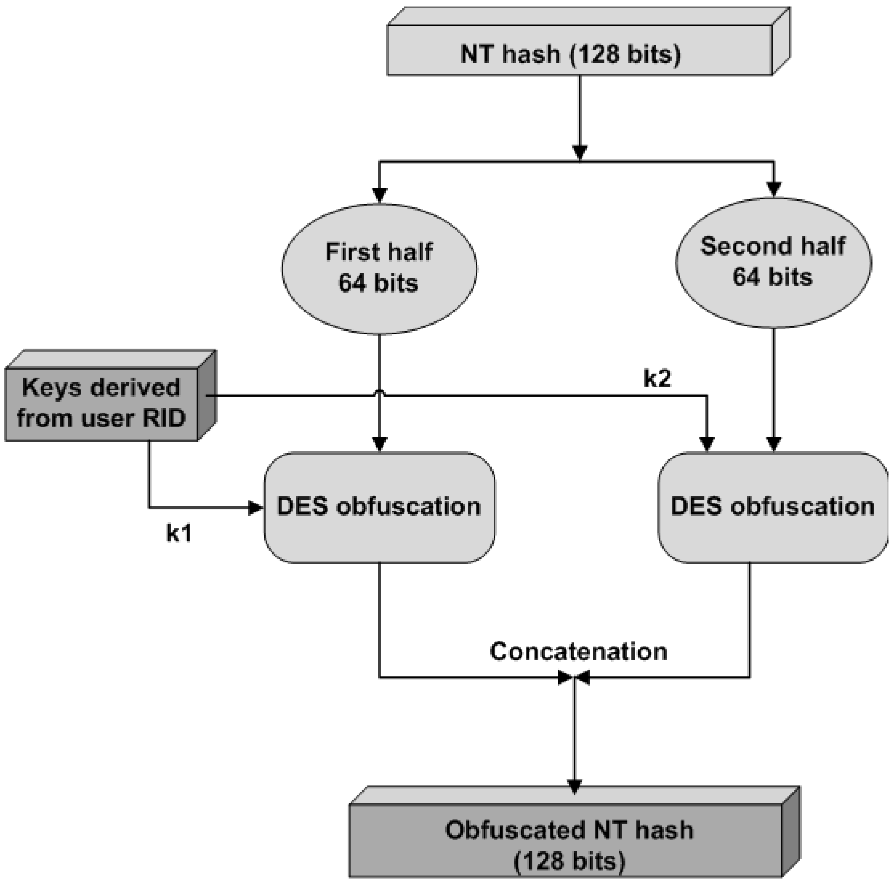


Figure 2.6: DES obfuscation of NT hash.

with *k2* as the key. The two results from the DES encryption are then concatenated to produce the obfuscated NT hash value of 128 bits. The obfuscated NT hash is then stored in the registry in the respective password hash fields [Tod07].

Because the encryption keys are known, as they are derived from the user RID, and also the encryption algorithm (DES) is known, the encryption process described above is for obfuscation purposes only. It does not provide true protection as an attacker who has access to the SAM file can easily eliminate the obfuscation as a result of that both the keys and the encryption algorithm is well know, and thus obtain the initial password hash. Note that the plaintext of the password is not revealed. Because the user RID is used to generate the DES obfuscation key, and because the user RIDs are different for every user, the obfuscated hash will be different for all users, even though they use the same password. This should not be confused with salt that is random bits unknown to the users. The salt is added to provide protection. But since the user RIDs are well-known they do not provide sufficient protection, and they can be used to decrypt the obfuscated hash [Tod07].

As a consequence of that the DES obfuscation only provides additional steps before access to the password hash values are gained, Microsoft invented the SYSKEY utility to mitigate the risk of attacks on the password hashes, as a result of direct access to the SAM file. The SYSKEY utility consists of a *password encryption key* (PEK) and the RID, which is already described. Windows 2000 and later Windows operating systems use SYSKEY encryption by default. The SYSKEY encryption is not used instead of the obfuscation encryption described above but rather adds an additional layer of encryption for the password hashes and other system secrets, such as LSA secrets. LSA (Local Security Authority) is a protected subsystem that authenticates and logs users onto the local system. LSA also maintains information about all aspects of local security on a system, and this information is referred to as LSA secrets [47]. The purpose with SYSKEY is to additionally encrypt password hashes and LSA secrets in order to protect them from attackers who may try to copy the SAM file from a computer. To accomplish this protection, SYSKEY uses a system-generated *password encryption key* (PEK), which is used to encrypt only the password-related fields in the SAM file, as for example the LM hash and the NT hash. As a consequence of that, a person that wants to reveal a password stored in the SAM file first have to gain access to the PEK and then decrypt the password field. Then the obfuscation, which is already described above, must be removed, and still only the password hash will be revealed, not the password itself. The PEK key material, which is randomly generated and used to calculate the PEK, is generated for every single Windows system when SYSKEY is enabled for the first time. The generation of the PEK key material will not be explained in further details in this thesis. The PEK is used to protect the local user accounts on the machine. The operating system needs to know the PEK material to be able to calculate the PEK, and then decrypt the information in the SAM to obtain the password hashes. As a result of this the PEK key material is stored in the Windows registry. To protect the PEK from attackers, the operating system generates the PEK using the random PEK key material and a *system* key. The system key is the reason why it is called SYSKEY utility. The system key is also called the boot key or the startup key. This is because the system key, which is stored in the Windows registry by default, is read from this registry when the operating system boots. Every time the operating system or authorized applications need access to SYSKEY encrypted data, LSASS.EXE, which is a process in Windows operating systems that is responsible for enforcing the security policy on the system, will use the system key stored in the registry to decrypt the data and provide the data to the application or service. This implies that the system key is for example not effective against attacks carried out from applications with administrative privileges running on the same computer as LSASS.EXE, as this process will provide the data when requested. It may be added that the LSASS.EXE process, among other things verifies users logging on to a Windows computer or server, handles passwords changes and creates access tokens [Tod07].

Even though Microsoft does not provide details regarding the use of the SYSKEY, the security researchers believe that the SYSKEY information is stored in the following location in the registry: *HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\LSA*. The keys *JD*, *Skew1*, *GBG* and *Data*, in that order, each contain 4 bytes of the *system* key.

To obtain the system key, these 4 bytes are concatenated, and the bytes are reordered as follows: [8th, 10th, 3rd, 7th, 2nd, 1st, 9th, 15th, 0th, 5th, 13th, 4th, 11th, 6th, 12th, 14th]. The result is the *system* key, and together with the PEK key material, it is used by the operating system to calculate the PEK. The PEK that is used to encrypt the NT hash is obtained by calculating the MD5 hash of the PEK key material, concatenated with the user RID, concatenated with the string *NTPASSWORD*. The result is named *key* in Figure 2.7. The PEK that is used to encrypt the LM hash, on the other hand, is obtained by calculating the MD5 hash of the PEK key material, concatenated with the user RID, concatenated with the string *LMPASSWORD*. The obfuscated password hash is then encrypted with the RC4 algorithm using the respective PEK as the key. The RC4 encryption of the obfuscated *NT hash* is shown in Figure 2.7. As illustrated by the same figure, the result is then stored in the SAM file. In an earlier implementation of SYSKEY was both the LM hash and the NT hash encrypted using the same PEK. Microsoft then experienced that the password hashes were vulnerable to relatively simple cryptanalysis attack, and therefore introduced the use of different PEKs for the LM hash and the NT hash [Tod07].

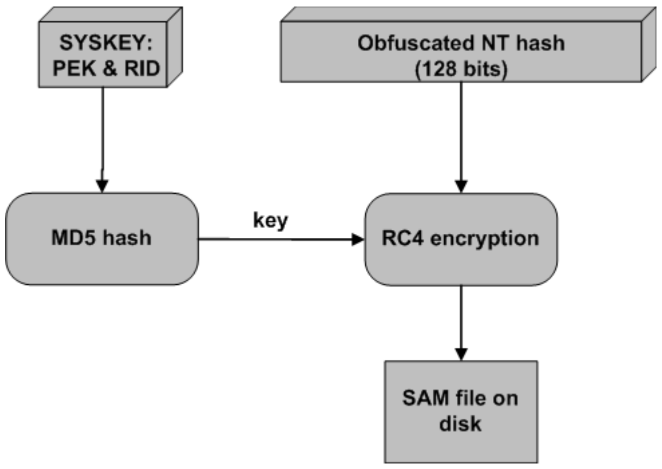


Figure 2.7: RC4 encryption of the obfuscated NT hash.

It should be highlighted that all the components used to protect the SAM file are available from the Windows registry, including the system key, which is stored in the registry by default. In other words, all the information that is used in the protection of the SAM file is stored on the hard disk, and is therefore used for obfuscation purposes only. The obfuscation process only results in that it takes longer time to reveal the password hashes. Because the system file is stored in a separate file in the registry called SYSTEM, both the SAM file and the SYSTEM file will be necessary in order to reveal the password hashes. These will not be a problem to obtain as they are both available in the registry. Before SYSKEY was invented, some older password hash dumping utilities tried to access password hashes directly, instead of trying to decrypt them first using the PEK. However, as will be shown later in this thesis, newer tools will be able to compute the PEK and will use it to decrypt the password field and obtain the hash. In this sense, the SYSKEY utility protect against older tools. It is important to note that SYSKEY encryption is only effective for passwords when they are stored in the SAM file. Applications that are able to dump password hashes will see DES obfuscated LM and NT hashes, and not SYSKEY encrypted hashes.

This section has now illustrated the process from a password is created until the obfuscated password hash is encrypted and stored in the SAM file.

2.3.2 Linux

Most Linux systems save the user information in two important files */etc/passwd* and */etc/shadow*. In earlier Linux systems the username and password hashes were stored together in the */etc/passwd* file, and the */etc/shadow* file did not exist. To increase the security a shadow suite (Section 2.3.2.2) with a */etc/shadow* file was introduced. This file is only readable by root. The */etc/shadow* file is used frequently to authenticate users, especially during the login procedure. The *etc/passwd* file is world readable but both of the files are accessible through the root account or can be accessed through another operating system, presented in more details in Procedure 10. The files are human readable, thus not obfuscated as other operating systems tend to do (Section 2.3.1.2). The Linux distributions presented in this thesis, utilize the same types of files.

2.3.2.1 The passwd file

The login process interacts with a file called */etc/passwd*. This file contains relevant information about the users, one user per line. An example of a user presented in the *passwd* file can be viewed in Listing 2.2:

```
nerblak:x:500:500:nerblak:/home/nerblak:/bin/bash
root:x:0:0:root:/root:/bin/bash
```

Listing 2.2: The users *nerblak* and *root* presented in the *passwd* file.

The fields separated by *:* are the following [48]:

- login name
- optional encrypted password
- numerical user ID
- numerical group ID
- user name or comment field
- user home directory
- optional user command interpreter

If the *optional encrypted password* field is left blank, no password is prompted [48]. Many early Linux distributions store the user passwords in */etc/passwd*. This is unsafe because */etc/passwd* is (and must be) world readable [Ano99], because the group IDs, user IDs and home directory information must be accessible for other users and processes. To avoid exposing the encrypted passwords most modern distributions have included or have support for implementing password shadowing.

2.3.2.2 The shadow suite

If the encrypted password field in */etc/passwd* is replaced by an *x* the encrypted password is stored in the */etc/shadow* file. This *shadow* file is only accessible by root. The *shadow* file contains the encrypted password for all the users and some optional password aging information. An example of the users *nerblak* and *root* is presented in Listing 2.3 below:

```
nerblak:$1$w8UNkyLw$P5Y4MqGOmKeog1eFeXjn90:14270:0:99999:7:::
root:$6$n2rIb6KV$zkgUHQjBZsR34MRknmeHK/zzamjgV04Rc5XP3JHM0jZfFbJVy889H2R
LA6PCx2FUFCWt.i/TSDCp9SrhcCZJJ.:14284:0:99999:7:::
```

Listing 2.3: The users *nerblak* and *root* presented in the *shadow* file.

The fields separated by *:* are the following [49]:

- Login name.
- Encrypted password.
- Days since Jan 1, 1970 that the password was last changed.
- Days before the password may be changed.
- Days after which the password must be changed.
- Days before the user is warned about the expiration of the password.
- Days after the password expires that account is disabled.
- Days since Jan 1, 1970 that the account is disabled.
- A reserved field.

The encrypted password field consists of minimum 13 characters from the 64 character alphabet a through z, A through Z, 0 through 9, . and /. This encrypted password is generated by the *crypt()* function. The *crypt()* function is based on the DES algorithm, but also includes variations mentioned below. A 56 bit key is generated using the lowest 7 bits of each of the first eight characters of the user typed password. This key is then used to encrypt repeatedly a constant string (usually a string consisting of all zeros) [50]. For the DES-based algorithm the return value of *crypt()* is the encrypted password represented in 13 printable ASCII characters, of which the two last characters comprise the salt. When the user enters their password for the first time, the salt should be set to a new string which is reasonably random [51]. When the *crypt()* function encrypts with DES, the salt consist of two characters from the alphabet mentioned above. The encrypted password field is separated into three fields in the form of *$id$salt$encrypted* when DES is not used. The salt is actually a combination of the two first fields, thus a character string starting with the characters *$id$*, where *id* represents the values shown in Table 2.3. This is followed by a string, referred to as *salt*, which is terminated by *$* [50]. The reason for including a salt when generating the hash value is because this ensures that two identical passwords not necessarily generate the same password hash. This is a desired feature to increase the set of password hashes possible per password. It is nevertheless important that the **same** salt is used when the hash value used to verify the password is generated. The *id* field can contain one of the four values presented in Table 2.3 below:

ID	**Method**	**Length**
1	MD5	22 characters
2a	Blowfish (not in mainline glibc; added in some Linux distributions and BSD)	
md5	Sun MD5	
5	SHA-256 (since glibc 2.7)	43 characters
6	SHA-512 (since glibc 2.7)	86 characters

Table 2.3: Id values available for the encrypted password in *shadow* [50] and [52].

The *id* field determines the encryption method and how the rest of the password string is interpreted. The two users presented in Listing 2.3 use *id* 1 and 6, meaning that their passwords are hashed with MD5 and Secure Hash Algorithm (SHA)-512. As you can see in this table, a shadow file can contain hash values produced by different hash algorithms.To configure the shadow password suite a file */etc/login.defs* is used [53]. This file has many configuration items with many different parameters available. One of the most interesting configuration items is *ENCRYPT_METHOD(string)*. In Ubuntu this item can take one of the following values: DES(default), MD5, SHA256 or SHA512 [53]. These values correspond to the *id* value in the *shadow* file.

2.3.3 BSD

The files presented in this section vary from one operating system to another. The two operating systems, *FreeBSD* and *Mac OS X*, presented in this section are therefore separated. The FreeBSD system use password files much like in various Linux systems, but with different names. Mac OS X stores the password in a different way, but the password files are human readable.

2.3.3.1 FreeBSD

The way FreeBSD handles the user's passwords is very similar to Linux. As in most Linux systems, FreeBSD uses a */etc/passwd* file. This file does not contain the passwords and is presented in more detail in Section 2.3.2.1. FreeBSD use a file called */etc/master.passwd* as storage for the password hash values and the username. The */etc/master.passwd* file is almost identical to the */etc/shadow* file mentioned in Section 2.3.2.2, but some of the fields contain different values and the file is structured a little bit different.

An example of the user *root* is presented in Listing 2.4:

```
root:$1$0cB4EA3Q$2v8tSjqlabQU19FYpKG541:0:0::0:0:Charlie &:/root:/bin/
csh
```

Listing 2.4: The user *root* presented in the *master.passwd* file.

The fields separated by *:* are the following [Leh03]:

- User name.
- Encrypted password.
- User number.
- Group number.
- Login class, which describes a number of parameters for the user. (In the table above empty, for the user *nerblak*)
- Password change time. If non-0, it is the time in seconds after which the password must be changed.
- Account expiration time. If non-0, it is the time in seconds after which the user expires.
- The so-called gecos field, which describes the user. This field is used by a number of programs, in particular mail readers, to extract the real name of the user.
- The name of the home directory.
- The shell to be started when the user logs in.

While the password handling and the password files in FreeBSD are almost identical to the ones in Linux, there is one noticeable difference. FreeBSD has also introduced two files for performance reasons. These files are the database versions of */etc/passwd* and */etc/master.passwd*, */etc/pwd.db* and */etc/spwd.db*, respectively. None of these database files are user-readable. If the passwords are changed in the */etc/master.passwd* file, a program called *pwd_mkdb* can rebuild the passwords files */etc/passwd*, */etc/pwd.db* and */etc/spwd.db* [Leh03].

2.3.3.2 Mac OS X

The password handling in Mac OS X has changed a lot since Mac OS X 10.0 ‘Puma’ was released. The operating system versions 10.0, 10.1 and 10.2, the latter released in 2002 used the Unix DES Crypt(3) function to hash the passwords. The hash values of the password were stored in a database called *NetInfo*. The hash values were not shadowed, as mentioned for Linux systems and FreeBSD, thus stored directly in the NetInfo database. This was different from the traditional way Unix systems implement their user information in flat files such as */etc/passwd*. All the users had access to the NetInfo database, allowing all users to read the hash values. According to Dribin [54], DES hashed password could easily be dumped in standard Unix format with the command *nidump*. Although DES was effectively proven weak in 1997 and shadowing of password was introduced by AT&T in 1987, Mac OS X had no intention of learning from the past.

In Mac OS X 10.3 released in 2003 some big changes were applied to the password handling. The passwords were now shadowed and stored outside the NetInfo database. Instead of using the standard */etc/shadow* file to store the hash values, they are stored files in the */var/db/shadow/hash/* directory. Each user’s password was stored in its own flat file. The files had names originating from the users *generateduid* NetInfo field. Like */etc/shadow*, these files are readable only by root. The passwords were no longer hashed with standard Unix *DES*. According to [54], the password was hashed twice in a string of 104 characters. The first 64 characters of the string containing the first hash value hashed with the Windows LM hash and the second part consisting of 40 characters containing a hash value hashed with SHA1. The LM hash is described in detail in Section 2.3.1.2. The Windows LM hash was included to among other things support *Windows file sharing*.

```
NTNTNTNTNTNTNTNTNTNTNTNTNTNTNTNTNTNTNTNTNTNTNTNTNTNTNTNTNTNTNTNTNTNT000000000000
0000000000000000000000000000000000000000SHA1SHA1SHA1SHA1SHA1SHA1SHA1SHA1SHA1SHA1
0000000000000000SaSHA1SaSHA1SaSHA1SaSHA1SaSHA1SaSHA1SaSHA1SaSHA10000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000
```

Listing 2.5: This shows the hash files used in Mac OS X 10.4.

The worst problem with Mac OS X version 10.3's password handling was that no salt was included and that the weak LM hash was included in the flat password file.

Dribin [54] mentions some vital changes that were applied when Mac OS X 10.4 was released in 2005. The passwords are still hashed with SHA1 and shadowed in the same file as described above. The LM hash is replaced with the more secure NT hash produced by Windows. The NT hash, described further in Section 2.3.1.2, is only generated if Windows sharing is enabled, and Mac OS X gives a warning that the password are being stored in a less secure manner. In 10.4 the SHA1 hash also uses salt.

Listing 2.5 above shows an example [55] of how the hash file is represented in Mac OS X 10.4 Tiger. It contains different fields where the NT, SHA1 and SaSHA1 field represents the legend of where the corresponding hash values are stored in a hash file. The NT field is included to support Windows file share, as mentioned above, and the SHA1 field is included to have backward compatibility with earlier versions of Mac OS X. The SaSHA1 (salted SHA1) field is saved for the SHA1 used in current versions of Mac OS X with their corresponding salt appended as the first part of the field. Why Apple has chosen to append the file with additional zeros, is probably for future compatibility in newer versions of Mac OS X.

The hash files in Mac OS X 10.5 Leopard all looks the same, and an example hash file with name *B90BA604-CAE1-4A69-9D3D-953F8CD7E422* corresponding to the user *nerblak* is presented in Listing 2.6 below:

```
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000533F07A2833EEB8107B2DA9F81A3AF08F744378651A57C5C000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000
```

Listing 2.6: The user *nerblak* presented in its *uid* named hash file.

For the two listings above it should be noted that these originally contains no new lines. These are just included to make the file more readable.

The salted SHA1 hash begins at character 169 and is 48 characters long. The first 8 characters are the hex value. In Mac OS X 10.5.6 Leopard described in this thesis, it is verified that this version of the operating system also use SHA1 with salt and the same shadow technique as mentioned above.

2.4 Tools

This section will describe the tools used in the experiments presented in Chapter 5 , 6 and 8. This section separate between tools used for password resetting and tools used for password recovery.

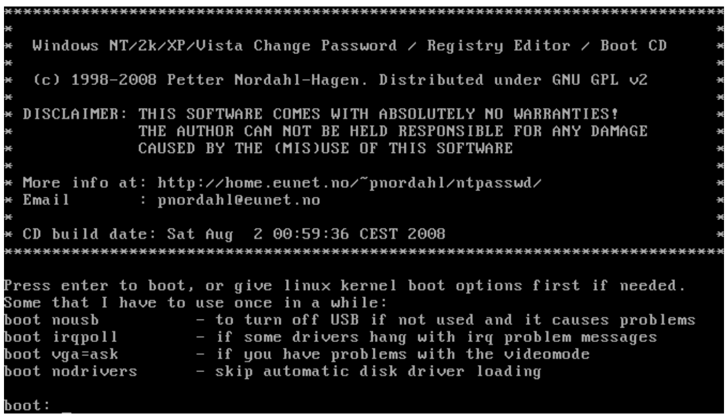
2.4.1 Password Resetting

Below we present the tools used for password resetting.

2.4.1.1 Offline NT Password & Registry Editor

Offline NT Password & Registry Editor [56] is a password resetting tool created by Petter Nordahl-Hagen. The tool can either be downloaded as a bootable live CD or floppy disk image, which includes the *chntpw* tool, or as plain source code. Some Linux distributions such as *Debian* have included the *chntpw* tool in their package repositories, providing a simple way to install the tool locally. Offline NT Password & Registry Editor can reset/set the password of any user that has a valid (local) account on a Windows NT/2k/XP/Vista etc system. Like most of the other tools described in this thesis, this tool works offline, meaning that the tool is initiated locally on the system subject to password resetting. The tool will automatically detect accounts and offer various options to modify the account. It can set or reset the password, unlock accounts or act as a registry editor. Offline NT Password & Registry Editor contains among other tools the password tool *chntpw*. Nordahl-Hagen gives a good explanation of how the tool works [56]:

> *NT stores its user information, including crypted versions of the passwords, in a file called 'sam', usually found in \windows\system32\config. This file is a part of the registry, in a binary format previously undocumented, and not easily accessible. But thanks to a German(?) named B.D, I've now made a program that understands the registry.*



```
****************************************************************************
*                                                                          *
*  Windows NT/2k/XP/Vista Change Password / Registry Editor / Boot CD      *
*                                                                          *
*  (c) 1998-2008 Petter Nordahl-Hagen. Distributed under GNU GPL v2        *
*                                                                          *
* DISCLAIMER: THIS SOFTWARE COMES WITH ABSOLUTELY NO WARRANTIES!           *
*             THE AUTHOR CAN NOT BE HELD RESPONSIBLE FOR ANY DAMAGE        *
*             CAUSED BY THE (MIS)USE OF THIS SOFTWARE                      *
*                                                                          *
* More info at: http://home.eunet.no/~pnordahl/ntpasswd/                   *
* Email       : pnordahl@eunet.no                                          *
*                                                                          *
* CD build date: Sat Aug  2 00:59:36 CEST 2008                             *
****************************************************************************

Press enter to boot, or give linux kernel boot options first if needed.
Some that I have to use once in a while:
boot nousb          - to turn off USB if not used and it causes problems
boot irqpoll        - if some drivers hang with irq problem messages
boot vga=ask        - if you have problems with the videomode
boot nodrivers      - skip automatic disk driver loading

boot: _
```

Figure 2.8: Offline NT Password & Registry Editor welcome screen.

A screen shot of the initial welcome screen is displayed in Figure 2.8. It should be mentioned that there exists many rescue CDs and other tools for resetting passwords, which *uses* the *chntpw* tool, and therefore do not contribute with a new tool but is rather a collection of different tools. The *chntpw* tool is for instance included in Trinity Rescue Kit [8].

2.4.1.2 Kon-Boot

Kon-Boot is tool to bypass the password authentication process on different operating systems. It is produced by Piotr Bania [57] who states that it has been tested to work correctly on the following Windows operating systems and Linux distributions with the following GRUB versions:

Windows Operating Systems
Windows Server 2008 Standard SP2 (v.275)
Windows Vista Business SP0
Windows Vista Ultimate SP1
Windows Vista Ultimate SP0
Windows Server 2003 Enterprise
Windows XP
Windows XP SP1
Windows XP SP2
Windows XP SP3
Windows 7

(a) Windows operating systems.

Linux kernels	**GRUB versions**
Gentoo 2.6.24-gentoo-r5	GRUB 0.97
Ubuntu 2.6.24.3-debug	GRUB 0.97
Debian 2.6.18-6-6861	GRUB 0.97
Fedora 2.6.25.9-76.fc9.i6862	GRUB 0.97

(b) Linux Distributions.

Table 2.4: Operating systems supported by the Kon-Boot tool.

Please note that Table 2.4(a) and Table 2.4(b) list system that the Kon-Boot tool has been tested to work correctly on by Piotr Bania, and that the tool also may work on other systems.

Kon-Boot is a tool, written in x86 assembly, which allows to change contents of a Linux and Windows kernel on the fly, while booting. It allows you to log in to an Administrator account without typing the correct password or to elevate privileges on regular user accounts. It should be noted that everything is done virtually, without any interferences with physical system changes [57].

According to information received through email contact with Piotr Bania, the general concept of Kon-Boot is the following:

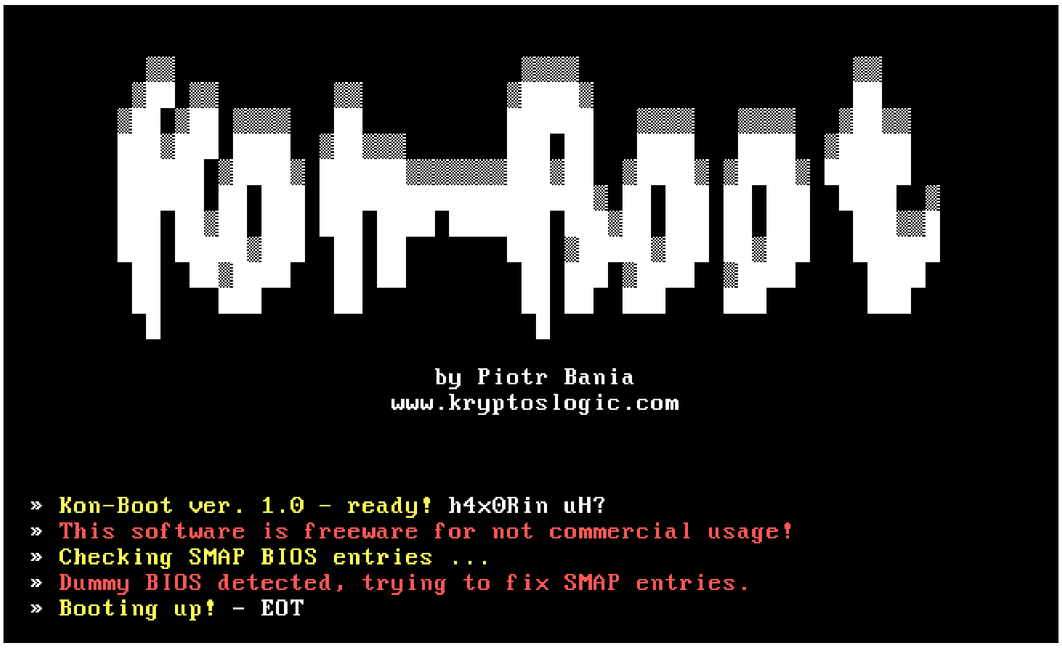
1. The computer boots up from the CD/Floppy with Kon-Boot.
2. Kon-Boot allocates necessary amount of memory in the BIOS and marks it as unavailable to avoid overwriting it by the operating system loader.
3. Then Kon-Boot copies itself to the allocated memory.
4. Kon-Boot hooks some of the BIOS interrupts responsible for disk-access meaning that it tries to monitor all the data read from the disk and then looks for signatures.

5. Depending on the signature that is found, Kon-Boot begins to make necessary changes to the operating system loader and then the real kernel.

Piotr Bania claims that the last point described above actually is the hard part. In some cases like for example in Windows Server 2003 both the operating system loaded and the kernel itself use checksum algorithms to prevent modification. Although this seems like a hard thing to circumvent, Kon-Boot takes care of such situations too.

When it comes to the case that Kon-Boot works on so many operating systems, one may wonder if the approach mentioned is the same for all operating systems, especially because of the large differences between the different Windows and Linux operating systems. According to Piotr Bania, the initial concepts are the same, but changes are required when placing a "password-bypass-backdoor". This varies depending on the kernel.

A screen shot of Kon-Boot is displayed in Figure 2.9 below:



```
by Piotr Bania
www.kryptoslogic.com

» Kon-Boot ver. 1.0 - ready! h4x0Rin uH?
» This software is freeware for not commercial usage!
» Checking SMAP BIOS entries ...
» Dummy BIOS detected, trying to fix SMAP entries.
» Booting up! - EOT
```

Figure 2.9: Kon-Boot almost finished with the kernel hack.

2.4.1.3 Bart's Preinstalled Environment (BartPE)

In many of the procedures described in Chapter 5, it is mentioned that a guest operating system is used to configure a host operating system. A Linux live CD is often thought of as the only solution that has the possibility to boot on top of the host system. This is not entirely correct. Bart's Preinstalled Environment (BartPE) [58] is a Windows Live CD solution build using a program called Bart's PE Builder and an original Windows XP or Windows Server 2003 installation/setup CD. This CD will according to [58] give the user a complete Win32 environment with network support, a graphical user interface

(800x600) and FAT/NTFS/CDFS file system support. It has the look of Windows, and it has many of the most important Windows tools available. There are also many other solutions to get a Windows like guest operating system, such as VistaPE [59], NTFSDOS [60] and NTFS4DOS [61].

There are many variants of Linux that may be used as a guest operating system, which may have support for many file systems. In this thesis Ubuntu 8.10 and 9.04 [9] has been used frequently. The tool *YALP*, described in detail in Section 7.2, was produced as a result of the lack of a live CD with desired password resetting and recovery capabilities.

Password Renew Together with BartPE a tool called *Password Renew* [62] can be used to reset an Administrator password. According to the creator, sala, the plug-in *Password Renew* has the following features:

- (Re)Set the passwords of any user that has a valid local account.
- Create a new local user with administrator rights.
- Set administrator rights to existing user on your NT system.
- Reset the local administrator password on a computer that is in a domain, using NULL Administrator password feature.

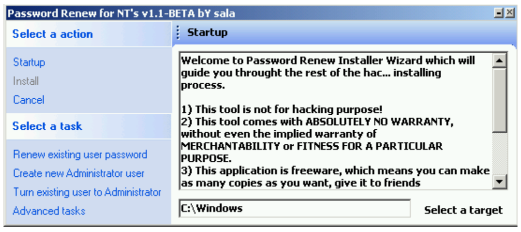
WindowsGate 1.1 Another password resetting plug-in for BartPE is *WindowsGate*. According to [63], the author of WindowsGate is the same author as the author of DreamPackPL described later in this chapter. WindowsGate bear resemblance in DreamPackPL and is embedded with many password handling mechanisms. Both plug-ins can only operate on the Windows password handling mechanism.

According to Damian [63], WindowsGate has the following features (note that the two latter features are Windows weaknesses exploited by WindowsGate):

- Unlock/Lock functions (enabling/disabling login password validation)
- Existing passwords remains untouched (without password reset or overwrite operations)
- Windows registry remains untouched.
- Without password (hash) cracking.
- Windows System File Checker (SFC) and Windows Resource Protection (WRP) remain active.
- Support for all NT version: 2000/XP/2003/Vista/2008
- Include support for *msv1_0.dll patch*.

- Run any process with system privileges from WindowsGate (*utilman.exe* replacement (WinKey+U shortcut)).

Figure 2.10 and Figure 2.11 presented below show Password Renew and Windows gate, respectively. The figures are presented side by side, because both are plug-ins to Bart's Preinstalled Environment.

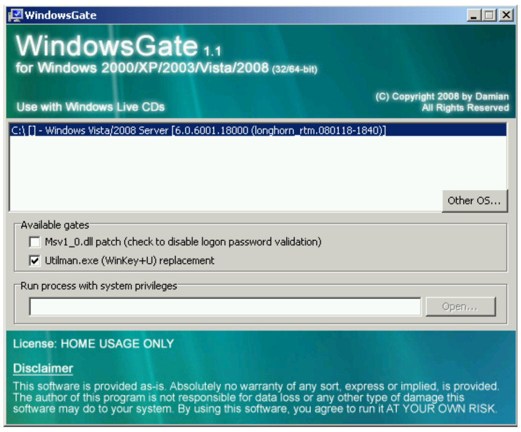

Password Renew for NT's v1.1-BETA bY sala
Select a action
Startup
Install
Cancel
Select a task
Renew existing user password
Create new Administrator user
Turn existing user to Administrator
Advanced tasks
Startup
Select a target

Welcome to Password Renew Installer Wizard which will guide you throught the rest of the hac... installing process.

1) This tool is not for hacking purpose!
2) This tool comes with ABSOLUTELY NO WARRANTY, without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
3) This application is freeware, which means you can make as many copies as you want, give it to friends

C:\Windows

Figure 2.10: Password Renew.


WindowsGate
WindowsGate 1.1
for Windows 2000/XP/2003/Vista/2008 (32/64-bit)
Use with Windows Live CDs
(C) Copyright 2008 by Damian
All Rights Reserved
Other OS...
Open...

C:\ [] - Windows Vista/2008 Server [6.0.6001.18000 (longhorn_rtm.080118-1840)]

Available gates
- [ ] Msv1_0.dll patch (check to disable logon password validation)
- [✓] Utilman.exe (WinKey+U) replacement

Run process with system privileges

License: HOME USAGE ONLY

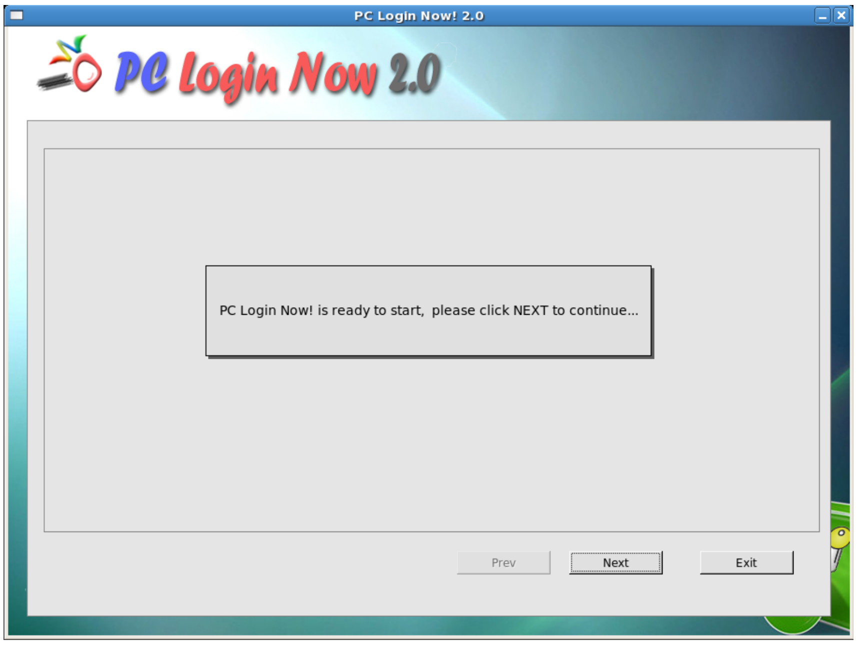
Disclaimer
This software is provided as-is. Absolutely no warranty of any sort, express or implied, is provided. The author of this program is not responsible for data loss or any other type of damage this software may do to your system. By using this software, you agree to run it AT YOUR OWN RISK.

Figure 2.11: Windows Gate.

2.4.1.4 PCLoginNow 2.1.0

PCLoginNow is one of many password resetting tools that are mentioned in this thesis. The tool is according to PCLoginNow's web page [64] an easy-to-use tool to reset local administrator and other account's passwords on Windows operating systems. The tool, shipped as a bootable live CD, works on all Windows systems and is controlled through a graphical interface. In addition to resetting the password, this tool is can perform some other vital user account changes. It can change any account to administrator level, lock/disable accounts and set accounts to never expire. The tool supports circumvention of the SYSKEY protection embedded in most of the newer versions of Windows.

Figure 2.12 shows a screen shot of PCLoginNow right after it has booted.



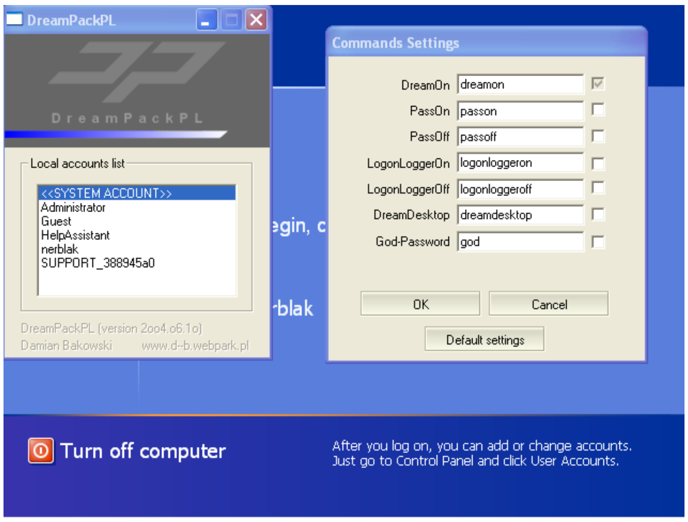
PC Login Now! is ready to start, please click NEXT to continue...

Figure 2.12: PCLoginNow's welcome screen.

2.4.1.5 DreamPackPL

A tool that has many password handling features is DreamPackPL. The tool is currently only supporting Windows 2000 and XP. When configured and installed correctly it is accessed directly from the login prompt of the operating system and provides a menu containing various tools. In this thesis we describe how to reset the administrator password by using this tool at page 106. The author, said to be Damian Bakowski, does not longer seem to provide a homepage for this tool, but it can be found on various web pages across the Internet, for instance [65]. The tool also exists as a plug-in for BartPE mentioned in Section 2.4.1.3. This plug-in can be found at this web page [66], but is not described any further in this thesis.

The screen shot in Figure 2.13 shows DreamPackPL booted on top of a Windows XP operating system. The main menu displays the users, and the *Command Settings* window has been started on the right side of the figure. In this window you can see textitdreamon as activated, making DreamPackPL possible to start from the login prompt, with the password *dreamon*.



Local accounts list

<<SYSTEM ACCOUNT>>
Administrator
Guest
HelpAssistant
nerblak
SUPPORT_388945a0

DreamPackPL (version 2oo4.o6.1o)
Damian Bakowski www.d-b.webpark.pl

DreamOn	dreamon	✓
PassOn	passon	
PassOff	passoff	
LogonLoggerOn	logonloggeron	
LogonLoggerOff	logonloggeroff	
DreamDesktop	dreamdesktop	
God-Password	god	

After you log on, you can add or change accounts.
Just go to Control Panel and click User Accounts.

Figure 2.13: DreamPackPL initiated from a Windows XP login screen.

2.4.1.6 The *passwd* Tool

The *passwd* tool is a password resetting tool included in most Unix-based systems. It can reset the password for any account residing on the system, as long as it is initiated with root/Administrator privileges. As the tool requires these privileges some may say that it is useless as a password resetting tool when the Administrator password is lost, and you have to be an administrator to initiate this tool The reason why this tool has its own approach presented in this thesis is because this tool is very useful in combination with other procedures. This thesis will present various ways to grant administrator access on all of the Unix-based operating systems used in this thesis. When this access is granted, *passwd* serves as an excellent tool to reset the password on any desirable account. An approach showing how to use this tool is presented at page 108.

2.4.2 Password Recovery

In the following pages we present tools that can be used for password recovery.

2.4.2.1 Ophcrack

Ophcrack [44] is a Windows password cracker based on rainbow tables. The tool is created by the inventor of rainbow tables, and offer a fast and effective way to disclosure a password hidden from a hash value (Section 2.1.2.3). The tool have both a graphical and a text-based interface, and support various operating systems. Ophcrack's web page presents a small list of features that the tool provides:

- Runs on Windows, Linux/Unix and Mac OS X
- Cracks LM and NT hashes.

- Free tables available for Windows XP and Vista.
- Brute-force module for simple passwords.
- Audit mode and CSV export.
- Real-time graphs to analyze the passwords.
- Live CD available to simplify the cracking.
- Loads hashes from encrypted SAM recovered from a Windows partition, Vista included.
- Free and open source software (GPL) [44].

The tool can be downloaded in four different variants. The first variant is a Windows installable version, where Windows 2000, XP and Vista are supported. The second variant is the source, providing a possibility to install the tool on most Unix-, Linux- and Mac based operating systems. The third and fourth variants are live CD's independent of an installation. Ophcrack delivers an Ophcrack Vista Live CD and an Ophcrack XP Live CD. The main difference is that the first includes *Vista free* rainbow tables and the latter includes *XP free small* rainbow tables. These rainbow tables take up a big portion of the Live-CD, thus only one of them can be included in the CD. The free XP rainbow tables are specially designed to crack the *LM hash* used in Windows XP, and the Vista free tables are designed to crack the *NT hash* used in both Windows XP and Vista. Due to the weaknesses of the LM hash, if Windows XP is the operating system where the password recovery is desired, rainbow tables meant for LM hash cracking should be used.

It is nevertheless possible to use the Ophcrack Vista Live CD on Windows XP, but it is better to use the Ophcrack XP Live CD on a Windows XP machine because of the properties of the rainbow tables. It is not possible to crack a Windows Vista password with the use of the Ophcrack XP Live CD, as XP uses the LM hash and Vista only uses the more secure NT hash. Screen shots of this tool can be found in Chapter 8.

As mentioned above, Ophcrack is also shipped as an installable version, and the support for including external rainbow tables is embedded in the program. From Ophcrack's web page [44] various rainbow tables can be downloaded. Some of them are free and some can be purchased and downloaded directly from the Internet.

Another famous password cracker using rainbow tables is *Rainbowcrack* [67]. This cracker is also an implementation of the theory presented in [Oec03], described in Section 2.1.2.3. With this cracker another tool called *rtgen* is included. The rtgen tool serves the possibility of creating rainbow tables. By using rtgen large collections of rainbow tables for many different hash algorithms can be created. The thesis will not present rtgen in detail.

2.4.2.2 John the Ripper

John the Ripper [68] often referred to as *jtr* or just *John*, is another famous password cracker compatible with most operating systems. According to the John the Ripper's web page, the tool's primary purpose is to detect weak Unix passwords. The tool supports many popular hash types used by various operating systems. John the Ripper can reveal passwords from hash values created by *crypt*, and John the Ripper has out of the box support for Kerberos/AFS and Windows LM Hashes.

In the experiments described in Chapter 6 and 8 the command line version of John the Ripper was used. The command line version of John the Ripper supports four different cracking modes [69]. The four primary modes are fully customizable and capable of running sequentially. The simplest cracking mode is called *Wordlist mode.* This mode allows the user to specify a default or customized wordlist. By using wordlist mode John the Ripper is used to perform a dictionary attack against the specified hash values. In the wordlist mode word mangling rules can be enabled. This is an option to modify or "mangle" words in the wordlist, thus enabling the testing of other likely passwords. By enabling rules a set of rules will be applied on every word in the wordlist. On Openwall's web page [70], a good description on how to disable specific rules is presented. A wordlist can be created manually [71], downloaded from the Internet as a single file [23] or purchased directly from Openwall's web page. Many of the rules available are very similar to those presented in Section 2.4.2.3.

To run John the Ripper on a password file (mypasswd) by using an external wordlist (Wordlist.lst) and all the rules enabled, the following command can be typed in a Unix shell with root permission:

```
john --wordlist=Wordlist.lst --rules mypasswd
```

Listing 2.7: John the Ripper issued as root with wordlist and rules enabled.

Another mode supported by John the Ripper is *Single crack mode.* It will for instance use the user's login name and home directory name as password candidates with a large set of mangling rules applied. According to Openwall, this is the best mode to begin with:

> *Since the information is only used against passwords for the accounts it was taken from (and against password hashes which happened to be assigned the same salt),"single crack" mode is much faster than wordlist mode. Successfully guessed passwords are also tried against all loaded password hashes just in case more users have the same password.*

The third mode supported by John the Ripper is called *Incremental mode.* This is according to Openwall the most powerful cracking mode; it will try all possible character combinations as passwords. This mode is John the Ripper's brute-force attack and it is assumed that cracking with this mode will never terminate because of the number of combinations being too large. This mode can be configured in John the Ripper's configuration file to specifically suit one type of hash algorithm.

The last mode John the Ripper supports, *External mode*, will not be described here as it is not used in the experiments described later in this thesis. Screen shots of John the Ripper can be found in Chapter 8.

2.4.2.3 Cain & Abel

Cain & Abel [72] is a password recovery tool for Microsoft Windows operating systems. It has support for many recovery techniques, and the techniques used in the experiments described later are cracking encrypted passwords using Dictionary, Brute-Force and Cryptanalysis attacks.

Cain & Abel can be divided into two main parts. Cain, the first part of the program is a graphical interface with the purpose of concentrating several hacking techniques and proof of concepts. It provides a simplified tool with focus on the recovery of passwords and authentication credentials from various sources. The second part of the program, which is called Abel, is designed as a Windows NT service consisting of an *Abel.exe* executable and a dynamic link library *Abel.dll*. The service runs, according to Montoro [72], using the local System account and provides some interesting features like the Remote NT Hashes Dumper, the Remote LSA Secrets Dumper and the Remote Console. More information on these features can be found on Cain & Abel's web page. Abel can be installed without the first part of the program Cain, through Cain locally or remotely.

Cain & Abel supports a large set of different hash types created by many popular hash algorithms. In Chapter 6 and 8 Cain & Abel is used to crack the LM and NT hash used by various Windows operating systems. Cain & Abel's graphical interface is simple and intuitive to use, and also other features can be added and activated. Screen shots of this tool can also be found in Chapter 8.

As mentioned with John the Ripper, Cain & Abel has different modes/crackers to reveal the password. One of the crackers supported by Cain & Abel is the *Dictionary Password Cracker*. This cracker works the same way as the wordlist mode described in John the Ripper. The tool Cain & Abel also has support for external wordlists and string manipulation (mangling of the words in a wordlist). Using multiple wordlists at the same time is a feature possible in both John the Ripper and Cain & Abel. Included with the Cain & Abel installation is a wordlist file. This file contains a list of words that can be used when performing a dictionary attack. Through Cain & Abel's graphical interface, in the dictionary attack menu, the following mangling rules can be selected [73]:

- As Is: -> the password id checked as written in the dictionary file.
- Reverse -> the reverse form of the password is tried (password -> drowssap).
- Double -> double the tried password (Pass -> PassPass)
- Lowercase -> the lowercase form of the password is tried (Password -> password).
- Uppercase -> the uppercase password is tried (Password -> PASSWORD).

- Numbers substitution permutations -> replace certain letters with numbers (Pass, P4ss, Pa5s, ... P45s, ...P455).

- Case Perms -> all case permutation of the password are checked (password, Password, pAssword, PAssword, ... PASSWORD).

- Two numbers Hybrid-Brute -> appends a maximum of two digits after each word (Password0, Password1,...Password9, Password00, Password01, ... Password99).

Another mode/cracker supported by Cain & Abel is *Cryptanalysis*. This mode allows for the use of rainbow tables as described in 2.1.2.3 to crack hash values created by various hash algorithms. Cain & Abel supports rainbow tables generated by *rtgen* and *Winrtgen* and Ophcrack rainbow tables mentioned in Section 2.4.2.1. This cracker can be selected from the Cain & Abel menu with a corresponding *charset* defining which character set the rainbow tables support. *Winrtgen* is a graphical rainbow table generator that supports many hash algorithms. We describe *Winrtgen* in more details as Tool 12 at page 120 and in Section 2.4.3.2.

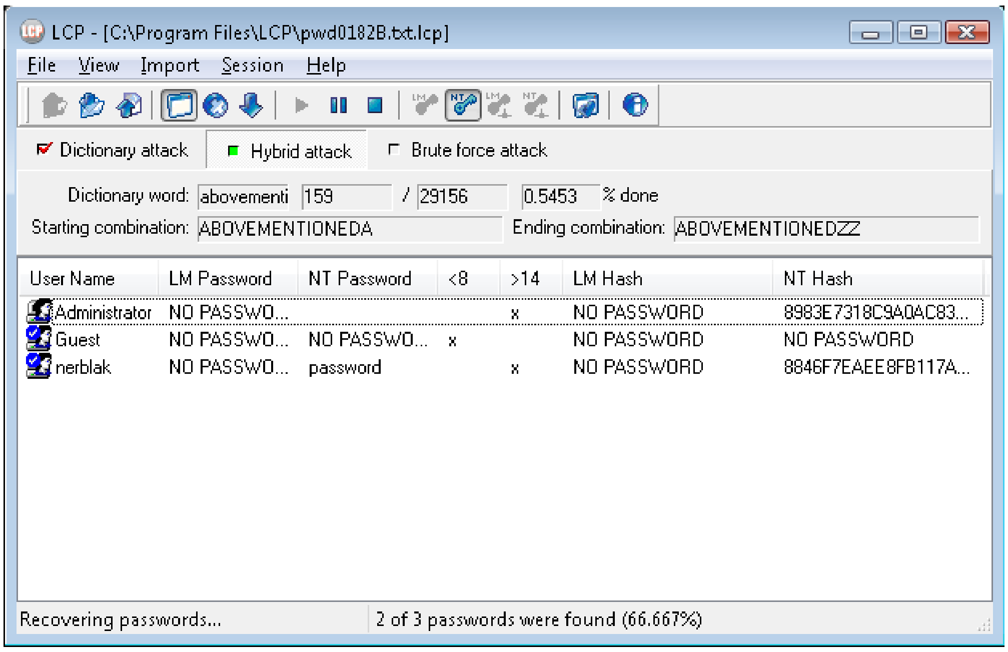
Brute-force Password Cracker is also a mode/cracker implemented in Cain & Abel. How brute-force works is described more in detail in Section 2.1.2.2. This thesis will not present how Cain & Abel works with this feature, but a description can be found at Cain & Abel's web page [72].

2.4.2.4 LCP

The last password cracker used in the experiments presented in Chapter 6 is called *LCP* [74]. The main purpose of LCP is to recover or audit a user account password in Windows NT/2000/XP/2003. The tool supports various methods to import account information, through a SAM-, .LC-, .LCS-, PwDump- or a Sniff- file, or just directly through a remote or local computer. The tool is delivered with three main password recovery attacks. These attacks are dictionary attack, brute-force attack and a hybrid of brute-force and dictionary attacks.

Figure 2.14 shows the LCP main window where three user accounts are under attack. LCP has revealed the user *nerblak's* password, simply just *password*.

There are many other password recovery tools that are similar to LCP such as *Lophtcrack (LC5)*, *MDcrack* and *lm2ntcrack* freely available on the Internet. Some password recovery tools can also be purchased such as *PasswordsPro* and *SAMInside*. This thesis will not describe these tools as many of them offer the same features as the tools described in this chapter. A good comparison of LCP and LC5 can be found at LCP's web page [74].


LCP - [C:\Program Files\LCP\pwd0182B.txt.lcp]
File View Import Session Help

[✓ Dictionary attack] Hybrid attack Brute force attack

Dictionary word: abovementi 159 / 29156 0.5453 % done
Starting combination: ABOVEMENTIONEDA Ending combination: ABOVEMENTIONEDZZ

User Name	LM Password	NT Password	<8	>14	LM Hash	NT Hash
Administrator	NO PASSWO...			x	NO PASSWORD	8983E7318C9A0AC83...
Guest	NO PASSWO...	NO PASSWO...	x		NO PASSWORD	NO PASSWORD
nerblak	NO PASSWO...	password		x	NO PASSWORD	8846F7EAEE8FB117A...

Recovering passwords... 2 of 3 passwords were found (66.667%)

Figure 2.14: LCP main window.

2.4.3 Supplementary

In this section we will look closer into some of the supplementary tools that are presented in this thesis.

2.4.3.1 The fgdump Tool

The *fgdump* tool is one of the tools that has an approach presented in this thesis. Many tools use this tool to fetch the hash values out of the obfuscated password file (*SAM*). On fgdump's web page [75], fgdump is described as a utility for dumping passwords on Windows NT/2000/XP/2003 machines. At this web page it is also stated that this tool can avoid all anti-virus programs, which is a problem that many of the password resetting/recovery tools we have tested suffer from. The anti-virus programs react to these tools, because they are often used with criminal purposes.

The *fgdump* tool has neat features such as fetching the password hashes through a network, although this is out of scope for this thesis. Many other similar tools that can dump and edit password files also exist. Some popular tools that will not be described in this thesis are *pwdump* [36], *samdump2* [76], *bkhive* [76] and *creddump* [77].

2.4.3.2 Winrtgen

Winrtgen is a very simple rainbow table generator for Windows systems. It has a nice little graphical interface, needs no installation, and can be easily configured. At Winrtgen's web page [77] the tool is described as a *Graphical Rainbow Tables Generator* that supports many different hash algorithms such as LM, NT and MD5. The tool is said to be a graphical implementation of the popular rainbow table tools *rtgen.exe* and *rtsort.exe*. Both of the latter tools can be found in the RainbowCrack software presented at this web page [67].

We present an approach for this tool as Tool 12 at page 115. At this page the tool is described in more details. We presented a poster at the *NOTUR2009* (The Norwegian Metacenter For Computational Science 2009) conference [2] held at NTNU, containing information about the generation of rainbow tables using for instance Winrtgen. This poster is included in Appendix F.

Chapter 3

Laboratory

In this chapter we will present the different computers used in our experiments described in Chapter 5, 6 and 8. The experiments were carried out at NTNU in Trondheim, with equipment provided by NTNU. Most of the procedures and tools used in the experiments did not require any network connection. We decided to update the computers with the latest software updates, in case of new important security updates. The lab consisted of 8 computers, each with a different operating system.

We chose to name the computers after 8 famous Norwegian fortresses/castles; Fredriksholm, Hegra, Akershus, Sion, Kristiansten, Steinviksholmen, Tunsberghus and Oscarsborg. We used names of fortresses/castles to symbolize the login mechanism on the operating systems installed on these computers. An operating system should be like a fortress, it should be hard for an unauthorized person to get access.

In the next sections we will present tables showing the hardware specifications and operating systems used for each computer. In addition to the operating systems we use tools already described in Section 2.4.

3.1 Hardware

The computers used in this lab contained relatively old hardware, but this was not a problem with the experiments presented in Chapter 5 and 6. It could have been desirable to have stronger computers for the Empirical Password Study presented in Chapter 8, but we decided to make the study possible to carry out by everyone, and came to a conclusion that the hardware were suitable. To carry out the experiments we could have chosen to use virtual machines instead of physical machines when installing the different operating system, with for instance VirtualBox [78]. We chose to use separate physical computers and not virtual computers to give the experiments a real-life aspect.

It should be noted that there are minor variations when it comes to entering the boot menu on different systems. As an example, some systems require the use of *F12* to enter the boot menu, while on other systems *F9* is required.

The hardware specifications for the 7 computers are presented in Table 3.1.

Computer name	**System type**	**Processor**	**RAM**
Fredriksholm	Dell OptiPlex GX270	Intel Pentium 4 2.60 GHz	1 GB
Hegra	Hewlett-Packard (HP) Compaq dc7700 Convertible Minitower	Intel Core 2 CPU 2.40 GHz	2 GB
Akershus	Dell OptiPlex GX280	Intel Pentium 4 3.00 GHz	2 GB
Sion	Dell OptiPlex GX280	Intel Pentium 4 3.00 GHz	2 GB
Kristiansten	Dell OptiPlex GX270	Intel Pentium 4 2.60 GHz	1 GB
Steinvikholmen	Dell OptiPlex GX270	Intel Pentium 4 2.60 GHz	1 GB
Tunsberghus	Dell OptiPlex GX270	Intel Pentium 4 2.60 GHz	1 GB
Oscarsborg	Apple Power Mac G5	Dual 2.3 GHz PowerPC G5	512 MB

Table 3.1: Computer hardware specifications.

3.2 Operating Systems

As mentioned above, all of the operating systems installed on these computers were updated with the latest Service Packs and updates. During the testing, a new release of Ubuntu Server was released, but we did not perform the procedures and tools on the new release. All of the operating systems installed are 32 bit versions, thus no 64 bit versions were experimented with in this thesis. Some may argue that Fedora 10 and Ubuntu Server 8.10 are just distributions of the Linux operating system. This is right, but these systems, use different kernel versions and have different ways of handling the passwords and the login mechanisms. To not create any confusion, we have chosen to name *distributions* as if they were *operating systems*.

We chose these operating systems, because they are popular and easy to handle. All of the operating systems work on the computers we have experimented with, and can be used for servers or desktop computers. The operating systems provide a login mechanism that is created to prevent unauthorized persons access to the system. In this thesis the login mechanism is tested.

The operating systems installed on the 7 computers are displayed in Table 3.2.

Computer name	**Operating system**
Fredriksholm	Windows XP Professional 5.1.2600 Service Pack 3
Hegra	Windows Vista Enterprise 6.0.6001 Service Pack 1
Akershus	Windows Server 2008 Standard 6.0.6002 Service Pack 2
Sion	Windows 7 Ultimate 6.1.7100 (Release Candidate)
Kristiansten	Ubuntu Server 8.10 Intrepid Ibex
Steinvikholmen	Fedora 10 Cambridge
Tunsberghus	FreeBSD 7.1-RELEASE
Oscarsborg	Mac OS X 10.5.6 Leopard

Table 3.2: Operating system overview.

Chapter 4

Method

First, this chapter outlines in which order the remaining studies of this thesis are performed. Then different overviews of the procedures and tools presented in this thesis are displayed in various tables. These overviews give an indication of which procedure and tool to use and where to use it.

In this thesis we have chosen to divide the part that describes approaches for the different procedures and tools into two chapters. Chapter 5 *Procedures*, presents all the *procedures* that were successfully carried out when trying to reset a lost Administrator password. Chapter 6 *Tools*, presents the *tools* that were successfully applied to reset or recover a lost Administrator password. In these chapters we are distinguishing between a *host* operating system and a *guest* operating system. A host operating system is the operating system that is already installed on the computer, while a guest operating system is an operating system that is booted on top of the host operating system. This is also described in Section 1.7. Both Chapter 5 and 6 present step-by-step approaches on how to reset or recover an administrator password. These chapters should be used as a reference and need not necessarily be read in its entirety.

In Chapter 7 *Results*, we will first present the results obtained from the experiments presented in Chapter 5 and 6. As a response to the results gained from these chapters, we decided to produce a tool called YALP, which also is further described in Chapter 7. A description of how to build YALP and a presentation of its tools are presented in Appendix A and Appendix B, respectively.

One of the main parts in this thesis is the *Empirical Password Study.* This study is presented in Chapter 8. As this study was pretty comprehensive, we decided to separate the results of this study from other results presented in this thesis. The results of the Empirical Password Study are thus also presented in Chapter 8.

In the overview sections presented next, all of the procedures and tools with at least one successful approach are presented. In each section a table is included, and all of the procedures or tools are listed. The tables give an overview of whether or not the procedure or tool works for the specific operating system. This means that the tables can be used as a reference to check which procedure or tool to use. Some of the procedures and tools might also be dependent on specific hardware, giving a successful result with our hardware, but an unsuccessful result on other occurrences. It is important to notice that the support of different operating systems presented in the tables below corresponds to which host system the procedure or tool work on, not which guest system it can run on. The tables in the sections below are only containing operating systems we have experimented with. The procedures and tools may support additional operating systems than the ones we experimented with. The operating systems we experimented with are listed in Table 3.2 Chapter 3.

Procedures	**Name**
Procedure: 1	Windows XP Repair Backdoor
Procedure: 2	Edit Shadow File in Unix Systems
Procedure: 3	Shortcut for Bypassing Regular Login Procedure
Procedure: 4	Replace a Pre-login Executable File
Procedure: 5	Use Mac OS X Installation Disk
Procedure: 6	Windows Password Reset Disk
Procedure: 7.1	Edit Boot Parameter: single
Procedure: 7.2	Edit Boot Parameter: init=/bin/bash
Procedure: 8	Reboot and Hold Apple+S
Procedure: 9	Enter Single-user Mode Through Boot-menu
Procedure: 10	Using Live CD to Obtain Password File
Procedure: 11	Using Live CD to Change Root Into the Host Operating System

Table 4.1: The names of the various procedures.

Procedures	**Name**
Tool: 1	PCLoginNow 2.0.1
Tool: 2	Offine NT Password & Registry Editor
Tool: 3.1	Password Renew
Tool: 3.2	WindowsGate
Tool: 4	Kon-Boot
Tool: 5	DreamPackPL
Tool: 6	passwd
Tool: 7	Ophcrack Live CD 2.1.0
Tool: 8	John the Ripper
Tool: 9	Cain & Abel v4.9.30
Tool: 10	LCP 5.04
Tool: 11	fgdump 2.1.0
Tool: 12	Winrtgen 2.8

Table 4.2: The names of the various tools.

4.1 Overview of Procedures and Tools

In this section an overview of the different procedures and tools that are used in our experiments is given. This is to give an indication of which procedures and tools that have been focused on, and what can be achieved by using them. We have first separated between whether the procedures and tools are used to reset or recover a password. Table 4.1 and Table 4.2 above show the names of the various procedures and tools.

4.1.1 Password Resetting

The tables presented in this section will quickly identify if a procedure or tool described in Section 5.1 and 6.1, can be used to reset an Administrator password. We have chosen to both separate between procedures and tools, and between Windows and Unix-based systems.

4.1.1.1 Procedures for Windows Operating Systems

In Table 4.3 below all procedures for password resetting described in this thesis are presented. This table will give an answer to whether the procedure works for any of the Windows operating systems focused on in this thesis.

Procedures	**Windows XP**	**Windows Vista**	**Windows Server 2008**	**Windows 7 RC**
Procedure: 1	**Yes**	No	No	No
Procedure: 2	No	No	No	No
Procedure: 3	**Yes**	No	No	No
Procedure: 4	**Yes**	**Yes**	**Yes**	**Yes**
Procedure: 5	No	No	No	No
Procedure: 6	**Yes**	**Yes**	**Yes**	**Yes**

Table 4.3: Password resetting procedures for different Windows operating systems.

4.1.1.2 Tools for Windows Operating Systems

In Table 4.4 all the tools for password resetting described in this thesis are displayed. The table will present whether the tool works for any of the Windows operating systems focused on in this thesis.

Tools	**Windows XP**	**Windows Vista**	**Windows Server 2008**	**Windows 7 RC**
Tool: 1	**Yes**	**Yes**	**Yes**	**Yes**
Tool: 2	**Yes**	**Yes**	**Yes**	**Yes**
Tool: 3.1	**Yes**	**Yes**	No	**Yes**
Tool: 3.2	**Yes**	**Yes**	**Yes**	**Yes**
Tool: 4	**Yes**	**Yes**	**Yes**	No
Tool: 5	**Yes**	No	No	No
Tool: 6	No	No	No	No

Table 4.4: Password resetting tools for different Windows operating systems.

4.1.1.3 Procedures for Unix-based Operating Systems

In Table 4.5 all procedures for password resetting described in this thesis are presented. This table will give an answer to whether the procedure works for any of the Unix-based operating systems focused on in this thesis.

Procedures	**Ubuntu Server 8.10**	**Fedora 10**	**FreeBSD 7.1**	**Mac OS X Leopard**
Procedure: 1	No	No	No	No
Procedure: 2	**Yes**	No	No	No
Procedure: 3	No	No	No	No
Procedure: 4	No	No	No	No
Procedure: 5	No	No	No	**Yes**
Procedure: 6	No	No	No	No

Table 4.5: Password resetting procedures for Unix-based operating systems.

4.1.1.4 Tools for Unix-based Operating Systems

Table 4.6 presents the tools used for password resetting. The table will give an answer to whether the tool works for any of the Unix-based operating systems focused on throughout this thesis.

Tools	**Ubuntu Server 8.10**	**Fedora 10**	**FreeBSD 7.1**	**Mac OS X Leopard**
Tool: 1	No	No	No	No
Tool: 2	No	No	No	No
Tool: 3.1	No	No	No	No
Tool: 3.2	No	No	No	No
Tool: 4	**Yes**	No	No	No
Tool: 5	No	No	No	No
Tool: 6	**Yes**	**Yes**	**Yes**	**Yes**

Table 4.6: Password resetting tools for Unix-based operating systems.

4.1.2 Password Recovery

The tables presented in this section will quickly identify if a tool described in Section 6.2 in Chapter 6 can be used to recover an Administrator password. We did not experiment with all of the tools presented in this thesis on all the selected operating systems. Nevertheless, the tables below present which operating system the specific tool supports, according to the tool's web page. We present no procedures that can be used to recover the Administrator password, thus tables for these are omitted. If the word *patch* is included in a table entry, there exist one or more patches that can enable the support for this operating system. Often the tools presented are said to support password recovery from a hash value created with a specific hash algorithm, instead of the support for the whole operating system. A normal person has usually no knowledge of the type of hash algorithm being used by his/her operating system, thus we chose to indicate which operating system is supported instead of supported hash algorithm.

4.1.2.1 Tools for Windows Operating Systems

In the table below, Table 4.7, all tools for password recovery described in this thesis are given. This table will give an answer to whether the tool works for any of the Windows operating systems concentrated on in this thesis.

Tools	**Windows XP**	**Windows Vista**	**Windows Server 2008**	**Windows 7 RC**
Tool: 7	**Yes**	**Yes**	**Yes**	**Yes**
Tool: 8	**Yes**	**Yes** (patch)	**Yes** (patch)	**Yes** (patch)
Tool: 9	**Yes**	**Yes**	**Yes**	**Yes**
Tool: 10	**Yes**	**Yes**	**Yes**	**Yes**

Table 4.7: Password recovery tools for different Windows operating systems.

4.1.2.2 Tools for Unix-based Operating Systems

In Table 4.8 all tools for password recovery described in this thesis are presented. The table will give an answer to if the tool works for any of the Unix-based operating systems focused on in this thesis. Cain & Abel has support for MD5, SHA1 and SHA2 hash values. We could not find any support for salt, and crypt() used by most Unix systems have their own implementation of these hash algorithms. We have therefore chosen the answer *No* for this password recovery tool as shown below, even though there might be patches for crypt() and salt support residing on the Internet.

Tools	**Ubuntu Server 8.10**	**Fedora 10**	**FreeBSD 7.1**	**Mac OS X Leopard**
Tool: 7	No	No	No	No
Tool: 8	**Yes**	**Yes** (patch)	**Yes**	**Yes**
Tool: 9	No	No	No	No
Tool: 10	No	No	No	No

Table 4.8: Password recovery tools for Unix-based operating systems.

4.2 Overview of Supplementary Procedures and Tools

In this section an overview of the different supplementary procedures and tools that are used in our experiment is given. The overview will give an indication of which procedure and tool that works on what operating system.

4.2.1 Procedures for Windows Operating Systems

Table 4.9 presents all supplementary procedures described in this thesis. The table will present whether the tool works for any of the Windows operating systems concentrated on in this thesis.

Procedures	**Windows XP**	**Windows Vista**	**Windows Server 2008**	**Windows 7 RC**
Procedure: 7.1	No	No	No	No
Procedure: 7.2	No	No	No	No
Procedure: 8	No	No	No	No
Procedure: 9	No	No	No	No
Procedure: 10	**Yes**	**Yes**	**Yes**	**Yes**
Procedure: 11	No	No	No	No

Table 4.9: Supplementary procedures for different Windows operating systems.

4.2.2 Tools for Windows Operating System

In Table 4.10 below all supplementary tools described in this thesis are introduced. This table will present an answer to whether the tool works for any of the Windows operating systems focused on in this thesis.

Tools	**Windows XP**	**Windows Vista**	**Windows Server 2008**	**Windows 7 RC**
Tool: 11	**Yes**	**Yes**	**Yes**	**Yes**
Tool: 12	**Yes**	**Yes**	**Yes**	**Yes**

Table 4.10: Supplementary tools different Windows operating systems.

4.2.3 Procedures for Unix-based Operating Systems

All supplementary procedures described in this thesis are presented in table 4.11. This table will give an answer to whether or not the tool works for any of the Unix-based operating systems focused on in this thesis. In Procedure 11 a success is highly dependent on the guest operating system used. If the guest operating system, the live CD, use the same file system as the host operating system, this procedure will probably succeed. With the guest operating system used in the approach of Procedure 11 at page 90, a success is only verified on the operating systems containing a *Yes* in the table below.

Procedures	**Ubuntu Server 8.10**	**Fedora 10**	**FreeBSD 7.1**	**Mac OS X Leopard**
Procedure: 7.1	No	**Yes**	No	No
Procedure: 7.2	**Yes**	**Yes**	No	No
Procedure: 8	No	No	No	**Yes**
Procedure: 9	No	No	**Yes**	No
Procedure: 10	**Yes**	**Yes**	**Yes**	**Yes**
Procedure: 11	**Yes**	**Yes**	No	No

Table 4.11: Supplementary procedures for Unix-based operating systems.

4.2.4 Tools for Unix-based Operating System

In Table 4.12 all supplementary tools described in this thesis are presented. The table will give an answer to whether the tool works for any of the Unix-based operating systems focused on in this thesis. As all of the Unix-based system use salt in their hash generation, Tool 12 does not support generation of tables if hash algorithms use salt.

Tools	**Ubuntu Server 8.10**	**Fedora 10**	**FreeBSD 7.1**	**Mac OS X Leopard**
Tool: 11	No	No	No	No
Tool: 12	No	No	No	No

Table 4.12: Supplementary tools for different Unix-based operating systems.

Chapter 5

Procedures

In this chapter we present step-by-step approaches to reset an Administrator password by using various procedures. It should be noted that this chapter does not necessarily have to be read in its entirety, and should instead be used as a reference work when password resetting is needed. We also present some supplementary procedures that can be used together with other procedures and tools to reset or recover the Administrator password. Most of the approaches to recover a lost password are based on the password recovery techniques described in Section 2.1.2. As password recovery techniques are generally realized through automated tools, no procedures have been presented in this thesis that can carry out password recovery.

5.1 Procedures for Password Resetting

In this section the different password resetting procedures are presented.

Procedure 1: Windows XP Repair Backdoor

Procedure	**Short description**	**Operating System**	**Objective**	**Prerequisites**
Procedure 1	Windows XP Repair Backdoor	Windows XP	Password resetting	Windows XP installation CD and the corresponding product key

Table 5.1: Procedure 1 *Windows XP Repair Backdoor.*

Description
This procedure exploits a backdoor in Windows XP repair mode to reset the administrator password. We found this procedure at the *Tweaking with Vishal v2* forum at [79]. This is carried out by inserting the Windows XP installation CD and entering repair mode, as described in detail below. By pressing SHIFT+F10 during the repair process you are able to enter Windows User Accounts menu and reset the password from there. In addition, the *User Accounts* menu offers other features regarding the login procedure as well.

Prerequisites
To be able to perform this procedure you must have the Windows XP installation CD and the corresponding product key available. It should be noted that, among the operating systems involved in this experiment, this procedure is only applicable to Windows XP.

Approach
This approach is a step-by-step description on how to exploit the Windows XP repair backdoor:

1. Boot the host system and press *F12* to enter the boot menu.
2. Insert the Windows XP installation CD and select *IDE CD-ROM Device* to boot the CD.
3. Press a key when *Press any key to boot from CD* appears on the screen.
4. When you get to the *Welcome to Setup* screen press *ENTER* to continue. Do **not** enter repair mode at this stage.
5. Press *F8* to agree to the *Windows XP Licensing Agreement*
6. At the Setup screen, highlight the desired operating system by using the arrows and then press *R* to repair.
7. Let the repair process run without any interaction. This may take a while.
8. After the *Copying Files* stage, a reboot is required. This will happen automatically.
9. During the reboot, do **not** press any key to boot from CD when the *Press any key to boot from CD* message appears.
10. When the host system starts the *Installing Windows* screen will be presented.
11. Keep your eyes on the lower left hand side of the screen, and when you see the *Installing Devices* progress bar, press *SHIFT+F10.*
12. A Windows command console (cmd.exe) will now appear.
13. At the prompt, type *NUSRMGR.CPL* and press *ENTER.*

14. You have now gained access to the *User Accounts* menu in the Control Panel without typing any password.

15. Pick the desired account and choose *Change the password.*

16. Now, change the password and close the *User Accounts* window.

17. Exit the command box by writing *exit* and pressing *ENTER.*

18. Continue on with the repair process (have your product key ready).

19. Once the repair is done, you are now able to log in with your new password.

Variations

- When the *User Accounts* menu is displayed, it is also possible to create a new administrator account. This will not reset your originally account but this account may be used to reset the original password at a later point in time.

- When you have selected the desired account in the *User Accounts* menu, you may also choose to remove the password instead.

- Instead of writing *NUSRMGR.CPL* in the Windows command console, you can type *control userpasswords2* to enter a different *User Accounts* menu. In this menu you can add or remove user accounts, reset the password and disable the *Users must enter a user name and password to use this computer* field. There are also other features available through this *User Accounts* menu, but these will not be described.

- Another possibility is to use the *net user* command in the Windows command console to reset the password. Instead of writing NUSRMGR.CPL or control userpasswords2, as described above, you can write *net user username password* in the Windows command console, where *username* is the username of the desired account and *password* is the new password you want to set. Then you can exit the command console and continue on with the repair process. When the repair process is done you should be able to log on with the new password. More details about the net user command can be found in [80]. It should be noted that Microsoft has not included Windows XP on the *Applies to* list for the net user command found at this web page.

Procedure 2: Edit Shadow File in Unix Systems

Procedure	Short description	Operating System	Objective	Prerequisites
Procedure 2	Edit shadow file in Unix systems	Ubuntu Server 8.10	Password resetting	None

Table 5.2: Procedure 2 *Edit Shadow File in Unix Systems.*

Description
As mentioned in the Section 2.3.2.2 the shadow file located at */etc/shadow* in a Unix system contains user's passwords hashed. Revealing the hashed password can be a time consuming process. This procedure involves resetting the password. By replacing the encryption field with a new encryption field in the shadow file, the password can be changed. This field contains information about which hash algorithm and salt were used to create the hash value. The shadow file can contain hash values from many different hash algorithms, as already described in Section 2.3.2.2. When this procedure is carried out, it is important to replace the whole encryption filed, namely the hash value, the salt and the hash algorithm id. It is also important to replace the original encryption field with an encryption field created with same hash algorithm as the original encryption field. If the hash value is produced by a different hash algorithm or the hash value does not correspond to the algorithm or salt, the system may be locked and unable to use. We got the idea for this approach from a user with nickname *phirleon* at this web page [81].

Prerequisites
This procedure requires access to the shadow file, and a precomputed hash values originating from the same hash algorithm and the desired password.

Approach
We use a precomputed hash value to replace the hash value for a specific user in the shadow file. The precomputed hash can for example be obtained from another operating system capable of creating the same hash type. The hash value we use is the same as in Listing 2.3 for user *nerblak*, where the hash value corresponds to the password *Hax0R*. The list below shows an example on how this is accomplished:

1. Obtain access to the shadow file using Procedure 10 or Procedure 7.
2. Open the file using a text editor, for instance *nano*.
3. Locate the encrypted password field for the desired user as described in Section 2.3.2.2.
4. Edit the shadow file by replacing the encrypted password field with the precomputed hash value mentioned above.

5. Save and reboot.

6. The password should now be changed.

Variations

- No variations.

Procedure 3: Shortcut for Bypassing Regular Login Procedure

Procedure	**Short description**	**Operating System**	**Objective**	**Prerequisites**
Procedure 3	Bypass the regular login procedure	Windows XP	Password resetting	No password set for the hidden *Administrator* account

Table 5.3: Procedure 3 *Shortcut for bypassing regular login procedure.*

Description
This procedure serves the objective of bypassing the regular login procedure in Windows XP, providing the possibility to reset the login password for user accounts. We found this procedure at the *Tweaking with Vishal v2* forum at [79]. When Windows XP is installed, it automatically creates an account named *Administrator*, which is not shown at the regular login screen. When installing Windows XP you have the choice to enable a password for this administrator account, but this is not done by default. The user is not given a warning if a password is not set. For this reason many users will continue the installation process without setting a password. The hidden administrator account may then be used to reset the password for a regular user account, which may also be an administrator account.

Prerequisites
This procedure only works if no password for the *Administrator* account is set during the installation process of Windows XP. It is also an assumption that the regular user account is password protected.

Approach
This is how you get access to the hidden *Administrator* account and then can reset the password on different user accounts:

1. Boot system.

2. When the Windows XP login screen is loaded, press CTRL+ALT+DEL twice.

3. You should now have entered the classic Windows login screen.
4. Write *Administrator* in the *User name* field.
5. Leave the password field blank and press *Enter*.
6. You have now accessed the hidden administrator account.
7. Go to Start -> Control Panel -> User Accounts.
8. Click on the user account of interest and then click the *Remove the password* link.
9. Select the *Remove Password* button and then the password is reset.
10. You may now log on to the respective user account with a blank password.

Variations

- There are no variations of this procedure as it apparently does not work on the other operating systems included in this experiment.

Procedure 4: Replace a Pre-login Executable File

Procedure	Short description	Operating System	Objective	Prerequisites
Procedure 4	Replace a pre-login executable file	Windows XP, Windows Vista, Windows Server 2008, Windows 7 RC	Password resetting	A guest operating system

Table 5.4: Procedure 4 *Replace a pre-login executable file.*

Description
We got the idea of this procedure from a user with nickname *rearthur2003* at [82]. This procedure serves the objective of replacing an executable file on the host operating system using a live CD containing a guest operating system. The replaced file is executable before a user has logged in. By replacing this file, *System* access to the host system is acquired. *System* access is described in further details in 2.2.1. In this procedure we replace the Sticky Keys program (Windows/System32/sethc.exe), with the Windows command console (Windows/System32/cmd.exe). The Sticky Keys program is created by Microsoft to help people with physical disabilities. Sticky Keys is a feature that makes it possible to press a modifier key, such as the SHIFT-, CTRL-, ALT- or the Windows Logo keys, and have them remain active until another key is pressed. The Windows

command console with *System* account privileges is used to reset the Administrator password.

Prerequisites
For this procedure to be successful you need a guest operating system that is able to mount the file system of the host operating system with write access.

Approach
For this approach we are using Ubuntu 8.10 as the guest operating system. As already mentioned, this guest operating system is used to get access to the host operating system's file system.

1. First you have to perform Procedure 10 up to and including step 11, without inserting the USB device as described in step 4 in the same procedure. Procedure 10 explains how to burn and boot a live CD.

2. You should now have mounted the file system at */mnt/os*.

3. Write *cp /mnt/os/Windows/System32/sethc.exe /mnt/os/Windows/System32/sethc.exe.bak* to backup the file we are going to replace. In our case, this is the *Sticky Keys* program. To retrieve the system as it was before applying this procedure, you have to copy the sethc.exe.bak to sethc.exe by writing *cp /mnt/os/Windows/System32/sethc.exe.bak /mnt/os/Windows/System32/sethc.exe*. Now you can delete the sethc.exe.bak by writing *rm /mnt/os/Windows/System32/sethc.exe.bak*.

4. Type *cp /mnt/os/Windows/System32/cmd.exe /mnt/os/Windows/System32/sethc.exe* to replace the *Sticky Keys* program with the *Windows command console*.

5. Write *shutdown -r now* to reboot.

6. During reboot you will be asked to remove the live CD.

7. When the host system has booted and the login screen appears press the *SHIFT* key 5 times quickly or until the Windows command console window appears. As you can see at the top of the window, the running program is Windows command console with the name *sethc.exe*.

8. Type *control userpasswords2* to enter the *User Accounts* menu.

9. From this menu you have full control over the system's users. You can for example add, remove and activate user accounts and reset their password.

10. To activate the *Administrator* account, click *Advanced* at the top menu and then *Advanced* under the *Advanced user management* menu. You should now have entered the *Local Users and Groups* program. This program can also be reached by directly typing *lusrmgr* from the *Windows command console* in step 8.

11. Double-click on *Users*, right click on *Administrator* and select *Properties*. Uncheck the *Account is disabled* box and press *OK*. This has to be done to enable the account and make it visible at the login screen.

12. Right click again on the *Administrator* account and chose *Set Password...*. *Proceed* when a warning is prompted.

13. Enter a new password and press *OK*. You have now reset the administrator password.

14. Close all the windows and reboot the machine.

15. You should now be able to log in as *Administrator* with your new password.

Variations

- We have chosen to replace the Sticky Keys program with the Windows Command Console, but it is possible to replace other programs with the Windows Command Console as well. Other programs that are available from the login screen are Narrator.exe, Magnify.exe and osk.exe. Narrator and Magnify are, as Sticky Keys, programs created by Microsoft to help people with physical disabilities while osk.exe is an on-screen keyboard. It is also said that it is possible to replace the *Windows/System32/LOGON.SCR*.

- Another variation is regarding the path to reach the programs mentioned above. In some systems the path is */Windows/System32* and for others, such as Windows XP, the path is */WINDOWS/system32/*.

- Even though we are using Ubuntu 8.10 as guest operating system in this approach, you may use other operating systems to obtain file access to the host system as well. One possibility is to use the YALP tool described in Section **??**.

- If you are unfamiliar with Linux, a Windows installation CD can also be used. When the installation CD has loaded, enter the repair mode and select the desired host operating system. Then select to enter Command Prompt, and enter the letter assigned to the drive the host system is located on. When the drive is accessed you can use the *copy* command to overwrite the *sethc.exe* file. In our case *sethc.exe* was located at *D:\Windows\System32\sethc.exe* and after entering the directory *D:\Windows\System32* the file was overwritten with the command *copy cmd.exe sethc.exe*. This variation worked with Windows 7 RC.

Procedure 5: Use Mac OS X Installation Disk

Procedure	**Short description**	**Operating System**	**Objective**	**Prerequisites**
Procedure 5	Use Mac OS X Installation Disk	Mac OS X Leopard	Password resetting	Mac OS X Installation Disk

Table 5.5: Procedure 5 *Use Mac OS X Installation Disk*.

Description
This approach is based on a tutorial you can find on this web page [83]. The goal of this procedure is to reset the *Administrator* password on a machine running Mac OS X Leopard without having to type the old, and unknown, password.

Prerequisites
To perform this procedure you need the Mac OS X Installation disk closest to the version of Mac OS X installed on the computer.

Approach
In this approach we are trying to reset the *Administrator* password on Oscarsborg, the computer running Mac OS X Leopard.

1. Insert the Mac OS X Installation disk and boot the system while holding the *C* key. This will tell the computer to boot from the CD-ROM.
2. When the Mac OS X Installation disk system launches, select the desired language. We chose *Use English for the main language.* Press *Enter* to confirm.
3. Ignore the *Welcome* screen and go to *Utilities* -> *Reset Password* in the top menu.
4. Select your Mac OS X hard disk volume.
5. Then select the *System Administrator (root)* account, enter a new password and click *Save*.
6. You will now get a confirmation that the password is saved. Click *OK*.
7. Close the windows by pressing the red button in the upper left corner.
8. Then press *cmd+q* to quit, where *cmd* is the Apple Command key found at the bottom left of the keyboard. Click on *Restart* in the next window to reboot the computer.
9. You should be able to log in to the *System Administrator (root)* account. When the login screen appears, select *Other*. Then write *root* in the *Name* field and use the newly created password.

Variations

- No variations.

Procedure 6: Windows Password Reset Disk

Procedure	**Short description**	**Operating System**	**Objective**	**Prerequisites**
Procedure 6	Windows Password Reset Disk	Windows XP, Windows Vista, Windows Server 2008, Windows 7 RC	Password resetting	Already created a Windows Password Disk (USB flash drive).

Table 5.6: Procedure 6 *Windows Password Reset Disk.*

Description
This procedure is Microsoft's solution to reset a lost user- or *Administrator* password. This means that Microsoft has built this solution into their operating system. The big disadvantage with this solution is that it requires that the user has created the password reset disk before the password is lost. All of the Windows operating systems we are experimenting with in this thesis have a built in function to create this password reset disk. Although the password reset disk can be called a prerequisite because it has to be created in advance (before the password is lost), we will give a short description on how this disk is created in Windows Vista and how it is used. Microsoft has a great tutorial on how to create a Windows Vista Password Reset Disk at this page [84]. The approach below is based on this. Tutorials also exist for other Windows operating systems, such as [85]. The Password Reset Disk will only work on the system where it has been created.

Prerequisites
The procedure requires a password reset disk, thus you need a USB flash disk or a floppy disk. The disk has to be made in advance, before the password is lost. This disk is specific to the operating system it was created on, meaning that you cannot use it on another Windows operating system, even though it is the same version. The removable media will be dedicated for this purpose, and information already located on the media may be lost. The disk will work for the account you created the disk on.

Approach
In this approach we will describe how to create a Windows Vista Password Reset Disk and how it is applied to reset the Administrator password when it is lost. The approach is therefore divided into two enumerating lists, one describing how to create the Windows

Password Reset Disk and the other describing how to use the newly created reset disk.

How to create a Windows Vista Password Reset Disk using a USB flash drive:

1. Boot Windows Vista and log in as Administrator or a user with *Administrator* privileges.
2. Click *Start* in the bottom left corner of the screen, and type *control userpasswords* in the *Start Search* box.
3. Click on *control userpasswords* in the *Programs* list.
4. Insert the USB flash drive. Wait a few seconds for the USB flash drive to load. Press *Esc* if the *AutoPlay* message appears on the screen.
5. In the *Task* list on the left-hand side, click on *Create a password reset disk.*
6. The Forgotten Password Wizard will appear on the screen, click *Next* to continue.
7. Then you need to specify on which drive you want to create the password reset disk. This will probably be recognized automatically. Click *Next* to continue.
8. Now you have to enter the *Current user account password* for the account you are logged in with. Click *Next* to create the disk.
9. The *Password Reset Disk* will now be created. Wait for the progress to complete and press *Next* to continue.
10. Press Finish to complete the *Forgotten Password Wizard.*
11. Your *Windows Vista Password Reset Disk* should now have been created.

How to use the Windows Vista Password Reset Disk:

1. Boot the Windows Vista machine, where the password is lost.
2. Insert the *Windows Vista Password Reset* USB flash disk created earlier.
3. When the login window appears, click on *Reset Password* located right under the username field.
4. If the *Reset Password* link is not visible, click the arrow at the right side of the password field without typing any password. *The username or password is incorrect* will appear, click *OK* and the *Reset Password* link should now be visible.
5. Click *Next* when the *Password Reset Wizard* appears on the screen.
6. Chose where the *Password Reset* disk is located, usually the default option, and click *Next.*

7. In the new window, type a new password and a new password hint. Leave them blank if no passwords are desired. Click *Next* to continue.
8. Click *Finish* to end the wizard.
9. Your password should now have been reset.
10. You can use the same *Password Reset* disk to reset the password in the feature. You do not need to update this disk.

Variations

- The approach describes how to create a Password Reset disk for a Windows Vista operating system. It is also possible to create a Password Reset disk for other Windows operating systems, but then the procedure might be different. This will not be described in this thesis.

5.2 Supplementary Procedures

In this section supplementary procedures that are used in addition to other procedures and tools to reset or recover a lost password are presented.

In this section supplementary procedures used in the process of password resetting or/and recovery, are presented. These essential procedures are important to present as many of the earlier mentioned password resetting and recovery procedures and tools depend on them.

Procedure 7: Edit Boot Parameters in Unix Systems

Both of the subprocedures presented below were found at [86].

Procedure 7.1: Edit Boot Parameter: single

Procedure	**Short description**	**Operating System**	**Objective**	**Prerequisites**
Procedure 7.1	Edit Boot Parameter: single	Fedora 10	End up in single-user mode	boot loader

Table 5.7: Procedure 7.1 *Edit Boot Parameter: single.*

Description
This procedure serves the objective of entering single-user mode by changing the boot parameters. If the boot parameters can be changed, it is desirable to end up in single-user mode. From this mode various procedures and tools can be applied to change the

administrator/root password.

Prerequisites
In this procedure a possibility to change the boot parameters had to be present. The boot loader has this capability. As mentioned in Section 2.2.2 Fedora 10 uses the GRUB boot loader.

Approach
The list below shows how to edit the GRUB boot loader boot parameters in Fedora 10.

1. Boot system.
2. When GRUB starts loading press *ESC* to enter the menu.
3. Highlight the desired Operating System by using the arrows.
4. Press *e* to edit the commands before booting.
5. Mark the line that begins with Kernel.
6. Press *e* to edit the line.
7. On the end of the line append the word *single*.
8. Press *enter*.
9. Press *b* to boot the edited line.
10. You should now have entered the single-user mode.

Variations

- Instead of writing *single* in the command prompt, as described in the approach above, you can enter *1*. You end up in the same mode when writing *1* as when you write *single*.
- When trying this procedure in Ubuntu Server 8.10 you enter a *Recovery Menu* which requires a password. This is the reason why this procedure is not successful in Ubuntu Server 8.10.

Procedure 7.2: Edit Boot Parameter: *init=/bin/bash*

Procedure	**Short description**	**Operating System**	**Objective**	**Prerequisites**
Procedure 7.2	Edit Boot Parameter: init=/bin/bash	Fedora 10, Ubuntu Server 8.10	Password resetting	boot loader

Table 5.8: Procedure 7.2 *Edit Boot Parameter: init=/bin/bash.*

Description
As mentioned in Section 2.2.2 the program */sbins/init* determines the runlevel and many different startup programs are initiated by the */etc/init.d/rc* program. By setting *init=/bin/bash* at startup, init will start the Bourne shell. [87] defines the Bourne shell as command execution program, often called a command interpreter. It is the original Unix shell developed by AT&T. When this shell is booted, root commands can be executed and the passwords can be changed.

Prerequisites
To easily change the boot parameter, a boot loader such as GRUB is preferable.

Approach
The systems we perform this procedure on use a boot loader called GRUB. The procedure is performed as follows on Ubuntu Server 8.10:

1. Boot the system
2. Press *ESC* to enter the GRUB boot loader menu.
3. When the GRUB boot loader appears on the screen highlight the preferred operating system and press *e* to edit the line.
4. Highlight the line beginning with kernel and press *e* to edit.
5. Append *init=/bin/bash* on the end of the line and press *enter*.
6. Press *b* to boot the changes.
7. The Bourne shell will now appear, and root access is granted without typing any password.

Variations

- In Fedora the approach is the same as for Ubuntu Server 8.10, described above. It should be noted that Fedora 10 does not display the GRUB loading screen in verbose mode so *ESC* have to be pressed rapidly while the system boots. The only difference we experienced with Fedora 10 was that the mounting did not go as planned after the system had booted. At step 7 in this approach, we had to type *mount -o remount, rw /*.

Procedure 8: Reboot and Hold Apple+S

Procedure	**Short description**	**Operating System**	**Objective**	**Prerequisites**
Procedure 8	Reboot and hold Apple+S	Mac OS X Leopard	End up in single-user mode	None

Table 5.9: Procedure 8: *Reboot and hold Apple+S.*

Description
This procedure will demonstrate how to end up in single-user mode in Mac OS X Leopard. We found this procedure described at [88].When this mode is entered a few more steps will give the user the ability to act as root. To reset the password other procedures and tools described above can be applied.

Prerequisites
This procedure has no prerequisites.

Approach
The procedure of putting the Mac OS X system into single-user mode is described below:

1. Boot system.
2. While booting the system hold *Apple+s.*
3. After a few seconds single-user mode is entered.
4. Type *mount -uw /* to enable read and write on an already mounted file system.

Variations

- No variations found.

Procedure 9: Enter Single-user Mode Through Boot-menu

Procedure	**Short description**	**Operating System**	**Objective**	**Prerequisites**
Procedure 9	Enter Single-user Mode Through Boot-menu	FreeBSD	End up in single-user mode and reset password	None

Table 5.10: Procedure 9: *Enter Single-user Mode Through Boot-menu.*

Description
The purpose of this procedure is to end up in single-user mode when the system boots. This will give you *Administrator* rights and then also the possibility to reset the *Administrator* password. We identified parts of this procedure in [89].

Prerequisites
This procedure has no prerequisites.

Approach
This approach is specific to FreeBSD and this approach is tested on FreeBSD 7.1 Release:

1. Boot the host system. Press enter or wait for the system to start when the first screen appears.

2. When the FreeBSD boot menu appears, type *4* to *Boot FreeBSD in single user mode.*

3. You are then asked to *Enter full pathname of shell.* Press *Enter* to choose the default location (/bin/sh). We chose the default location.

4. Type *fsck -y*. fsck is a file system consistency check and interactive repair of the disks.

5. Write *mount -a* to mount all file systems not marked as *noauto*. Type *man mount* or *man fsck* to get more information about these command. This has to be done after the *mount -a* command has been issued.

6. You are now ready to reset the *Administrator* password, and there are a number of procedures to use to perform this action.

7. It can for example be done by entering */usr/bin/passwd root* or *passwd*, which is also mentioned in Procedure 6.

8. Type a new password and confirm by re-typing the password again.

9. The *Administrator* password has now been changed.

Variations

- When the FreeBSD boot menu appears, as described in stage 2 above, an alternative is to type 6 *Escape to loader prompt* and then type *boot -s*. You can now continue from stage 3.

Procedure 10: Using Live CD to Obtain Password File

Tool/ Procedure	**Short description**	**Operating System**	**Objective**	**Prerequisites**
Procedure 10	Using Live CD to Obtain password file	All	Password resetting and recovery	Live CD able to mount file system

Table 5.11: Procedure 10 *Using Live CD to Obtain password file.*

Description
A live CD is a bootable CD with an operating system pre-installed on the CD. This operating system does not need to be installed on the host system, it can boot directly from the CD. Many Linux distributions have a live CD publicly available on the Internet. In this procedure the goal is to obtain the password file of the operating system. When the admin password is lost and the system is locked, the password file is inaccessible from the system itself, except for the case described in 1. To get access to the password file, the file containing the hash value of the password, a live CD can be used. A user can then just use the live CD to copy the file to an external location, or begin the password resetting or password recovery directly. To perform the latter, reset and recovery tools have to be already included in the live CD or some other procedures mentioned below, e.g. 2, have to be carried out.

Prerequisites
This procedure requires a live CD. The live CD must be able to mount the host file system containing the password file. Ubuntu 8.10 has already installed a program called *mount*, capable of mounting the most common file systems. When you are using a bootable CD, the host system must be able to boot from a CD drive. To copy the password file to e.g. a USB device this device has to be mounted. This is done automatically in Ubuntu 8.10.

Approach
In this procedure we use Ubuntu 8.10 as the guest operating system and Windows XP as the host operating system. We used the host computer called *Fredriksholm.* In this system, F12 is pressed during boot to enter the boot menu. This can vary from system to system. A live CD can be obtained from the Ubuntu web page [9]. To accomplish this procedure the following steps can be performed:

1. Download *ubuntu-8.10-desktop-i386.iso* from [9] choosing a location near you.

2. Burn the image file (ubuntu-8.10-desktop-i386.iso) on an empty writable CD using e.g. ImgBurn [90].

3. Boot the host system and enter the boot menu by pressing *F12.*

4. Insert the USB device and live CD, and select *IDE CD-ROM Device*.
5. Select preferred language.
6. Select *Try Ubuntu without any change to your computer*.
7. Open a terminal by selecting Applications->Accessories->Terminal.
8. Write *sudo -i* to become root (Administrator user).
9. Write *fdisk -l* to find out which device the desired partition is recognized on and what file system is used. E.g. in our case the device is recognized as */dev/sda1* and the file system is *HPFS/NTFS*. The USB will also be present in this list.
10. Make directories by writing *mkdir /mnt/os*, to use for the partition.
11. Write *mount -t ntfs-3g /dev/sda1 /mnt/os*.
12. The password file is now located in */mnt/os/WINDOWS/system32/config/SAM*.
13. The system key is also required with some password crackers; this is located at */mnt/os/WINDOWS/system32/config/system*.
14. Write *cp /mnt/os/WINDOWS/system32/config/SAM /mnt/os/WINDOWS/system32/config/system /media/disk/*.
15. The password file *SAM* and the system key *system* are now copied to the USB stick.

Variations

- In this procedure many different guest operating systems could have been used. There are many variants of live CDs available on the Internet. The live CD can be easy and intuitive to use, or can be advanced and more difficult to use. Some live CDs have better hardware support than others, and some may contain more security tools than others. The capacity and functionality of the live CD can be compared to the size of the operating system. Some live CDs are even so big that they only fit on DVDs.
- This procedure also applies to many different host operating systems such as Windows XP and Mac OS X Leopard. The limitations are whether or not the guest operating system is capable of mounting the file system. Guest operating systems, for instance Ubuntu 8.10, can mount various file system types such as ntfs, fat32 and ext3.

Procedure 11: Using Live CD to Change Root Into the Host Operating System

Procedure	**Short description**	**Operating System**	**Objective**	**Prerequisites**
Procedure 11	Using Live CD to Change Root Into Host Operating System	Fedora 10, Ubuntu Server 8.10, FreeBSD 7.1, Mac OS X Leopard	Change root directory	Empty writable CD, same guest and host operating system

Table 5.12: Procedure 11 *Using Live CD to Change Root Into The Host Operating System.*

Description
To easily change the password through a live CD, a user can first mount the file system of the host operating system using a tool called *mount* and then change the root directory using a tool called *chroot*. By running *chroot* from the guest operating system to the host operating system we will be able to interact with files on the host system through programs located on the host system. By changing the root into the host system, the terminal window will act as the host system, and changes to the passwords can be made. For instance Tool 6, can after a *chroot* be issued from the guest operating system towards the host operating system and interact with its password files. It should be noted that, according to our observations, the guest operating system should be of the same operating system type as the underlying host system. We choose to use Fedora 10 both as guest operating system and as host operating system, but it should also work if you use for example Ubuntu as both guest and host operating system. Be aware that the approach then may differ from the approach described below. The approach below is partly based on the tutorial found in [91].

Prerequisites
This procedure requires a live CD. The live CD must be able to mount the host file system containing the password file. Fedora 10 have already installed a program called *mount*, capable of mounting the most common file systems. It also includes a program *chroot* to change the root directory. The guest operating system used should be of the same operating system type as the host system. When you are using a bootable CD, the host operating system must be able to boot from a CD drive.

Approach
In this approach we will describe how to use the *chroot* command from a Fedora 10 live CD, which functions as a guest operating system, to a host system also running Fedora 10. Steinvikholmen is the computer running Fedora 10 in our experiments.

1. Download *F10-i686-Live.iso* from [92].
2. Burn the image file (F10-i686-Live.iso) on an empty writable CD using e.g. ImgBurn [90].
3. Boot the host system and enter the boot menu by pressing *F12*.
4. Insert live CD, and select *IDE CD-ROM Device*.
5. Fedora live CD will now boot.
6. Select preferred language and keyboard layout at the bottom menu. Then click *Log In*.
7. Select Application->System Tools->Terminal to open a terminal window.
8. Write *su* to become root (Administrator user).
9. Type *mkdir /mnt/os* to create a mount directory.
10. Now, type *fdisk -l* to list the Fedora partitions that are available.
11. The guest operating system is now ready to mount the file system of the host operating system Fedora 10 in the */mnt/os* directory.
12. When the *fdisk -l* was issued the Fedora live CD listed the Fedora partitions. The partition we want to mount is a Linux LVM (Logical Volume Manager) partition.
13. To scan your system for LVM volumes and identify the volume name that has your Fedora volume, write: *vgscan*. Our LVM volume was *VolGroup00*.
14. Activate the volume by typing *vgchange -ay*.
15. Now, write *lvs* to find the logical volume that has your Fedora root file system. In our case the logical volume was *LogVol00*.
16. Mount the file system with *mount /dev/VolGroup00/LogVol00 /mnt/os*.
17. Write *chroot /mnt/os* to change the root directory from the guest operating system to the host operating system.
18. Other procedures such as Procedure 2 and tools such as Tool 8 and 6 can be utilized to reset or recover the passwords.
19. To exit the environment you changed the root directory to type *exit* and press *Enter* to exit.
20. Unmount the host operating system by writing *umount /mnt/os*.
21. Type *reboot* to reboot the system

Variations

- It may be sufficient, when for example using Ubuntu as both guest and host operating system, to just mount the partition device with the corresponding partition type that is listed when running the *fdisk -l* command. So, if you are not running a Fedora system or a Linux system using LVM, the approach is slightly different from the one above.

- In this procedure a variation can appear when the user tries to mount file system of the host operating system. The file system type and partition placement can vary from host operating system to host operating system.

Chapter 6

Tools

In this chapter we present step-by-step approaches to reset and recover an Administrator password by using various tools. It should be noted that this chapter as well, does not necessarily have to be read in its entirety, and should instead be used as a reference work when password resetting or recovery is needed. We also present some supplementary tools that can be used together with other procedures and tools to reset or recover the Administrator password.

6.1 Tools for Password Resetting

In this section the different password resetting tools are presented.

Tool 1: PCLoginNow 2.0.1

Tool	**Short description**	**Operating System**	**Objective**	**Prerequisites**
Tool 1	PCLoginNow 2.0.1	Windows XP, Windows Vista, Windows Server 2008, Windows 7	Password resetting	Empty writable CD

Table 6.1: Tool 1 *PCLoginNow 2.0.1*.

Description
This tool also serves the ability to reset the *Administrator* password. Using this tool it is not possible to set a new password but to set the desired user's password blank.

Prerequisites
To use this tool, it has to be burned on an empty writable CD. The tool *PC Login Now 2.0.1* can be downloaded from [93].

Approach
This tool was tested on Windows Vista with the following approach:

1. Download the tool from [93].
2. Burn the image on an empty writable CD using for instance ImgBurn available from [90]
3. Boot the system and press *F9* to enter the boot menu. Insert the boot CD into the CD-ROM and select to boot from CD.
4. When the system has booted, a window displaying the program PC Login Now 2.0 appears.
5. Press *Next* to continue.
6. Highlight the preferred operating system and click *Next*.
7. Select the desired user, and check the box labeled *password is empty*. Make sure that the *is disabled* box is unchecked to make the account enabled.
8. Click on *Next*.
9. A window confirming the reset will appear. Select *no* to exit.
10. On the next window, press *OK* to reboot the computer.
11. The Administrator password should now be cleared and the account should be visible in the login window.
12. Remove the CD from the CD-ROM drive when rebooting.
13. If the computer needs to run a disk check, let it complete or press any key to skip and continue.
14. You should now be able to log in with the *Administrator* account without typing any password.

Variations

- On PCLoginNow's web page [64] a newer version of this software is available. The web page claims that the software is free, but when trying to download, a page displaying *Only PC Disk Tools Customers who has ever purchased any of PC Disk Tools products can download any of the freeware issued by PC Disk Tools* appears.

Tool 2: Offline NT Password & Registry Editor

Tool	**Short description**	**Operating System**	**Objective**	**Prerequisites**
Tool 2	Offline NT Password & Registry Editor	Windows XP, Windows Vista, Windows Server 2008, Windows 7 RC	Password resetting	Empty writable CD

Table 6.2: Tool 2 *Offline NT Password & Registry Editor*.

Description
We chose to use the boot disk available from the *Offline NT Password & Registry Editor* web page [56]. As described in Section 2.4.1.1 at page 49 this tool can reset the password on various Windows operating systems. On the web page the tool is said to be tested on NT 3.51, NT 4 (all versions and SPs), Windows 2000 (all versions & SPs), Windows XP (all versions, also SP2 and SP3), Windows Server 2003 (all SPs), Windows Vista 32 and 64 bit [56]. We will test the tool on operating systems mentioned in table 6.2 mentioned above.

Prerequisites
To use this tool, it has to be burned on an empty writable CD. The tool is also available in a floppy disk version, but this will not be tested in this thesis. The procedure is the same for the CD and the floppy disk version. The tool was downloaded from Offline NT Password & Registry Editor web page [56].

Approach
This approach shows how to apply the tool on a machine running Windows Vista, in our case the *Hegra* machine listed in Table 3.2 on page 64. The approach is very similar and applicable for the other operating systems mentioned in the table above.

1. Download the tool from [56].
2. Unzip the *cd080802.zip* file with for instance WinZip available from [94].
3. Burn the image *cd080802.iso* on the empty writable CD using for instance ImgBurn [90].
4. Boot the host system and press F9 to enter the boot menu.
5. Insert the boot disk into the CD-ROM drive.
6. Select to boot from *CD-ROM*.

7. A menu will appear on the screen, press enter or just wait for the system to boot.
8. A partition table will show the partitions on the machine. Select the correct partition by typing the corresponding number. Usually the default value is the desired value.
9. Other options are also available, but these will not be described.
10. Select the path to the registry directory, again the default value *Windows/system32/config*, is usually the correct one.
11. Select which part of registry to load, in our case choose *Password Reset*, also listed as the default value.
12. You should now have entered the *chntpw Main Interactive Menu.*
13. Select *1* to edit the user data and passwords, which are also listed as default.
14. The user accounts are now listed. Enter the desired username or select the default value *Administrator* by pressing *enter.*
15. The *User Edit Menu* is now displayed and two choices for resetting the password are available.
16. The choices are to either clear or edit the password. Choose *Clear (blank) user password* by pressing *1* and then *enter.*
17. The password should now be cleared.
18. Even though the password is cleared, the administrator account may additionally have to be enabled.
19. Press *enter* to edit the *Administrator* account again.
20. Select *4 Unlock and enable user account [probably locked now].* If the user account is already enabled this stage is unnecessary and instead of [probably locked now] the menu will display [seems unlocked already].
21. The Administrator account should now be enabled.
22. Select *!* and press *enter* to return to the *chntpw Main Interactive Menu.*
23. Select q and press *enter* to quit.
24. The last step is the *Writing back changes* stage. Type *y* and press *enter* to save the changes.
25. The edit is now complete. Select the default value *n* by pressing *enter* when asked for a *New run?* .

26. Remove the CD from the CD-drive and press *CTRL-ALT-DELETE* to reboot.
27. You should now be able to log in to the *Administrator* account without typing any password.

Variations

- As already mentioned above, there are two possible options to reset the administrator password when using this tool. You can either clear or edit the password. We chose to clear the password as it appears that editing the administrator password does not work on Windows XP, Windows Vista, Windows Server 2008 and Windows 7. Even though editing the administrator password did not work for the operating systems included in the experiment, this may not be the case for other Windows operating systems.
- There are also minor variations when it comes to enabling the administrator account, as described in the approach above.
- Some operating systems have the administrator account enabled by default while other operating systems require you to manually enable it. This can be done by using the scripts already included in the Offline NT Password & Registry Editor tool. The account has to be enabled to be displayed on the login screen.

Tool 3: BartPE

Tool	**Short description**	**Operating System**	**Objective**	**Prerequisites**
Tool 3.0	BartPE	All Windows systems	System access	An empty writable CD

Table 6.3: Tool 3.0 *BartPE.*

Description
BartPE (Bart Preinstalled Environment) [58] is a bootable CD-ROM based on the Windows XP installation CD. Even though the BartPE CD is based on the installation files of Windows XP it has shown effective on Windows Vista as well. It should be noted that BartPE is not a Microsoft product. To reset the password a plug-in to BartPE has to be included when producing the BartPE CD and the CD is produced with a tool called PE-Builder [58]. *Password Renew* [62] and *WindowsGate* [63], which are both described below, are plug-ins that makes password resetting possible with the use of BartPE.

Prerequisites
The tool must be burned on an empty writable CD before it can be used. Bart's PE Builder, Windows XP Installation CD and Password Renew are used to build the BartPE bootable CD.

Approach
To prepare the BartPE bootable CD, included in the approach below, we used another machine than the one that is subject to the password resetting. The following approach partly shows how BartPE is constructed. This approach must then be followed by either Tool 1 or Tool 2 to be able to reset a password.

1. Download PE-Builder from Bart Lagerweij's web page [58].
2. Run the *pebuilder3110a.exe* to install PE-Builder. Follow the default installation settings.
3. Insert the Windows XP Installation CD and alternatively copy it to a folder on the hard drive.
4. Start PE Builder and agree to the PE Builder license by pressing *I agree.*
5. The program will automatically prompt *Search for Windows installation files?* Select *Yes.*
6. Hopefully an installation location will be found and press *OK*.
7. Now proceed with Tool 3.1 or Tool 3.2. It should be noted that it is possible to include both of the plug-ins *Password Renew* and *WindowsGate*, which are described in those two tools respectively, on the same BartPE CD.

Variations

- No variations.

Tool 3.1: Password Renew

Tool	**Short description**	**Operating System**	**Objective**	**Prerequisites**
Tool 3.1	Password Renew	Windows XP, Windows Vista, Windows 7 RC	Password resetting	Included on for example the BartPE CD

Table 6.4: Tool 3.1 *Password Renew.*

Description
Password Renew is a tool which in this approach is included as a plug-in on the BartPE CD. The tool has a lot of different features to configure user accounts, which is described in Section 2.4.1.3. In this approach we will concentrate on the password resetting feature. BartPE with Password Renew serves the ability to reset the administrator password. There are also other tools included on the BartPE boot CD that are able to perform other useful tasks.

Prerequisites
Password Renew is a tool that in our case is included as a plug-in on the BartPE CD, so the construction of the BartPE CD is required in this approach.

Approach
This approach shows how *Password Renew* can be included as a plug-in on the BartPE CD and afterwards this approach describes how to reset the password on a Windows XP operating system. We performed the approach on Fredriksholm.

1. First you have to perform the steps described in Tool 3.
2. Then, download *Password Renew* from sala's web page [62].
3. Press *Plug-ins* and then *Add* to add *Password Renew* as a plug-in.
4. Locate the renew_1.1-BETA.cab file and open it by pressing *Open*.
5. Select the default plug-in folder name and press *OK*.
6. Press *Close*. The plug-in should now be installed.
7. If the Windows XP Installation CD files were copied to a folder, as mentioned in step 3 in the approach for Tool 3, you can use PE-Builder to burn the BartPE CD directly by selecting *Burn to CD/DVD* and pressing *Build*. If the Windows XP Installation CD was used directly choose *Create ISO image* and select *Build*.
8. If the latter option was selected press *yes* to automatically create the destination folders.
9. Agree to the license again and the build process should start.
10. An image file located at the destination folder should now have been created.
11. Burn the image with for example ImgBurn available from [90].
12. Boot the newly created BartPE CD on the system with the desired administrator account, and press *F12* to enter the boot menu.
13. Select to boot from *IDE CD-ROM Device*.
14. Press *No* when asked to start network support.

15. From the *GO* menu at the bottom left chose *Programs -> Password Renew.*
16. In the *Password Renew* program, click on *Select a target* in the *Startup* window. The target is the path to where the host system is installed. In our case, *C:\ WINDOWS.*
17. Press *OK* when the correct path is chosen.
18. Now, click on *Renew existing user password* in the left hand menu.
19. In the *Renew existing user password* menu, select a user with administrator privileges as the account and chose a password. Note that the account has to be enabled. This is not automatically done with the *Administrator* account, and cannot, as far as we know, be done with Password Renew.
20. Select *Install* in the left hand menu.
21. A window telling that the password renew is successfully done. Press *OK.*
22. Then press *Quit* in the left hand menu.
23. Press *GO -> Shut down -> Restart* to restart the system.
24. Remove the BartPE CD during reboot.
25. If the computer needs to run a disk check, let it complete or press any key to skip and continue.
26. You should now be able to log in to the host system with the *Administrator* account and the password you selected.

Variations

- This tool is said to work on Windows 2000 and 2003 as well, but this will not be tested in this thesis.

Tool 3.2: WindowsGate

Tool	**Short description**	**Operating System**	**Objective**	**Prerequisites**
Tool 3.2	WindowsGate	Windows XP, Windows Vista, Windows Server 2008, Windows 7 RC	Password resetting	Included on for example the BartPE CD

Table 6.5: Tool 3.2 *WindowsGate.*

Description
WindowsGate is a tool which in this approach is included as a plug-in on the BartPE CD. When using this tool you can either start the Windows User Accounts menu at the login screen, to for example reset the password of the desired account, or you can make WindowsGate disable the logon password validation.

Prerequisites
WindowsGate is a tool that in our case is included as a plug-in on the BartPE CD, so the construction of the BartPE CD is required in this approach.

Approach
This approach shows how *WindowsGate* can be included as a plug-in on the BartPE CD and afterwards this approach describes how to reset the password on a Windows XP operating system. We performed the approach on Fredriksholm.

1. First you have to perform the steps described in Tool 3.
2. Then, download *WindowsGate* from [63].
3. Press *Plug-ins* and then *Add* to add *WindowsGate* as a plug-in.
4. Locate the *windowsgate.cab* file and open it by pressing *Open.*
5. Select the default plug-in folder name and press *OK.*
6. Press *Close.* The plug-in should now be installed.
7. If the Windows XP Installation CD files were copied to a folder, as mentioned in step 3 in the approach for Tool 3, you can use PE-Builder to burn the BartPE CD directly by selecting *Burn to CD/DVD* and pressing *Build.* If the Windows XP Installation CD was used directly choose *Create ISO image* and select *Build.*
8. If the latter option was selected press *yes* to automatically create the destination folders.
9. Agree to the license again and the build process should start.
10. An image file located at the destination folder should now have been created.
11. Burn the image with for example ImgBurn available from [90].
12. Boot the newly created BartPE CD on the system with the desired administrator account, and press *F12* to enter the boot menu.
13. Select to boot from *IDE CD-ROM Device.*
14. Press *No* when asked to start network support.
15. From the *GO* menu at the bottom left chose *Programs -> WindowsGate.*

16. In the *WindowsGate* program, highlight the desired host operating system.
17. Check the *Utilman.exe* box to see WindowsGate at logon when pressing *WinKey+U*.
18. Press *OK* to confirm. Then, close the WindowsGate window.
19. Press *GO -> Shut down -> Restart* to restart the system.
20. At the logon screen press *WinKey+U* to start WindowsGate. From this program you can for instance start the Windows User Accounts menu by typing *Control UserPasswords2* in the *Run process with system privileges* field. Press *Open* to start the process. What you can do from the Windows User Accounts menu is already described in for example Procedure 4 from step 9 and beyond.

Variations

- There is also a possibility to check the *msv1_0.dll patch* box, found in the WindowsGate menu, to disable logon password validation. When the *msv1_0.dll patch* box is checked you should be able to log in to the account without typing any password. When you are logged in it is now possible to run the *control userpasswords2* command, and from there for instance enable the administrator account and/or set a new password on the administrator account.
- To redo the changes performed by WindowsGate you have to start WindowsGate again. You can either access the WindowsGate program by booting the BartPE CD or by using the *WinKey+U* shortcut, if this is enabled. Now, uncheck the options listed and close the window. After a restart the login procedure should be back to normal.

Tool 4: Kon-Boot

Tool	Short description	Operating System	Objective	Prerequisites
Tool 4	Kon-Boot	Windows XP, Windows Vista, Windows Server 2008, Ubuntu Server 8.10	Bypass password authentication	Empty writable CD

Table 6.6: Tool 4 *Kon-Boot.*

Description
Kon-Boot is a tool used to bypass the login procedure. It may look like this tool reset your password automatically but in fact it disables the password authentication instead. In Windows systems, when the Kon-Boot CD is removed from the CD-ROM and the system is rebooted, you will need to type in a password again.

Prerequisites
To use this tool it must be burned on an empty writable CD. *Kon-Boot* can be downloaded from [57].

Approach
This approach shows Kon-Boot tested on Hegra, which has Windows Vista installed. The approach for Linux systems is presented below this list.

1. Download the tool from [57]. We chose the *Kon-Boot Windows & Linux version 1.1-2in1*.

2. Burn the image on an empty writable CD using for instance ImgBurn available from [90]

3. Boot the system and press *F9* to enter the boot menu. Insert the boot CD into the CD-ROM and select to boot from CD.

4. A *kryptos logic security software* splash screen appears. Press *Enter* to continue.

5. Then Kon-Boot loads and the login screen appears. If the computer has more than one account, it is possible to select the desired account during start-up.

6. Leave the password field blank, press *Enter*

7. Now you should be logged in as the default user.

8. Click the Windows button and R simultaneously to enter the *Run* menu.

9. Then type *control userpasswords2* and press *Enter* to enter the *User Accounts* menu.

10. You can now follow Procedure 4 from step 9 to reset the Administrator password, create a new account and also enable an account.

The Linux approach is a little bit different than the Windows approach above. The approach below shows Kon-Boot tested on Kristiansten, which has Ubuntu Server 8.10 installed.

1. Perform the four first steps presented in the approach above. Press *F12* instead of *F9* to boot the CD.

2. After Kon-Boot has loaded; make sure that the text-based login appears, usually by pressing *Alt + F1* or *Alt + Ctrl + F2.* On Kristiansten no graphical interface loads at boot, and text-based login appears automatically, this is because no graphical interface is installed on this machine.
3. Type *kon-usr* as username and press enter. You are now automatically logged in as root. This can be verified by typing *whoami.*
4. You can for instance use Tool 6 and write *passwd* to change the root password.
5. Type *exit* and press *enter* to exit the user.
6. To restore the system type *kon-fix* as a username and leave the password field blank.
7. The system will probably prompt *Login incorrect*, just ignore this message.
8. The *Administrator* (root) account password should now be reset.

Variations

- As you can see in the approaches presented above, this tool is used differently on Windows and Linux systems.
- When you try to change the password with the *passwd* command, there might be some variations on how to run this command. These differences are presented in more detail in Tool 6.

Tool 5: DreamPackPL

Tool	**Short description**	**Operating System**	**Objective**	**Prerequisites**
Tool 5	DreamPackPL	Windows XP	Password resetting	Empty writable CD, Windows XP Installation CD

Table 6.7: Tool 5 *DreamPackPL.*

Description
This tool is based on an approach found in [65]. When the tool is properly configured according to the approach presented below, a menu with a lot of options will be accessible from the login screen. The tool will provide many possibilities to reset the Administrator password.

Prerequisites
To use this tool it must be burned on an empty writable CD. *DreamPackPL* can be downloaded from [65]. You must also keep the Windows XP Installation CD handy, since we are performing this procedure on a computer running Windows XP.

Approach
DreamPackPL was tested on Fredriksholm, which has Windows XP installed.

1. Download the tool from [65]. Use the link labeled *DreamPackPL v2004.06.10.*

2. Unzip the *dreampackpl 20040610.zip* file with for instance WinZip available from [94].

3. Enter the unzipped *dreampackpl 20040610* folder and run the *DreamPackPL* program. Click on *Run* if you are asked *Are you sure you want to run this software?*

4. A *DreamPackPL Installation* window will now appear. Insert the Windows XP Installation CD and click on *Create CD* to create a bootable CD containing DreamPackPL.

5. A new window, *DreamPackPL CD Creator*, will appear.

6. Select the correct CD-ROM drive and click on *Make ISO CD image.*

7. Then select a name for the image file (.iso file) and specify where you want to save it.

8. Wait for the process that creates the image file to finish.

9. A new window telling you that the image file is ready will appear. You may now close all windows related to DreamPackPL.

10. Burn the image file on an empty writable CD using for instance ImgBurn available from [90]

11. Boot the system and press *F12* to enter the boot menu. Insert the boot CD into the CD-ROM and select to boot from CD.

12. Press a key when the *Press any key to boot from CD* message displays on the screen.

13. Then wait while Windows is loading setup files.

14. You should now have entered the *Windows XP Professional Setup* menu.

15. Press *R* to repair a Windows XP installation using the recovery console.

16. *Microsoft Windows XP Recovery Console* is now loaded and the Windows installations are listed. Write the number corresponding to the Windows installation you would like to log in to. We wrote *1*: (C:\WINDOWS), which is quite common for systems with only one Windows installation. Press *Enter* to continue.

17. Type in a password of your choice when asked to *Type the Administrator password.* This is not the password to use when logging in when this process is done, but still this is a required step.

18. Now you have to make a backup of the original *sfcfiles.dll* file by using the following command: *ren C:\Windows\System32\sfcfiles.dll sfcfiles.lld*

19. Overwrite the *sfcfiles.dll* file located in *\Windows\System32* folder on the computer with the *pinball.ex_* from the CD to the by writing: *copy D:\i386\pinball.ex_ C:\Windows\System32\sfcfiles.dll*, where D in our case is the CD drive.

20. Type *exit* and press *Enter* to exit the *Recovery Console.*

21. The computer will now restart. Make sure that Windows boot normally. Remove the CD during reboot if necessary.

22. At the Windows login window, write *dreamon*, which is a DreamPackPL command, in the user name or password field. This will display the *DreamPackPL* menu.

23. Click on the top graphic, and another menu will appear.

24. Select *Commands -> Commands settings.*

25. In the *Commands Settings* menu, make sure that the Good-Password box is checked with for instance *god* as keyword. Now close all DreamPackPL windows. In Figure 2.13 at page 55 the *Command Settings* menu and the *DreamPackPL* menu is displayed.

26. Now, type *god* in the password field in the Windows login screen. This will work for all existing user accounts.

27. You should now be entering the account.

28. When the account is entered, press *Win+R* and enter *control userpasswords2* to enter the *User Accounts* menu. You can follow from step 4 in Procedure 4 to enable the *Administrator* account and change the password.

29. If you want *DreamPackPL* removed, just enter the menu mentioned in step 22 and select *Uninstall DreamPackPL.*

Variations

- DreamPackPL has a lot of different features and some of them can make changes to the user accounts. These will not be presented in this thesis.

Tool 6: passwd

Tool	**Short description**	**Operating System**	**Objective**	**Prerequisites**
Tool 6	passwd	Ubuntu Server, Fedora, FreeBSD, Mac OS X	Password resetting	Already entered single-user mode or after a *chroot* command.

Table 6.8: Tool 6 *passwd*.

Description
This tool serves the objective of resetting the *Administrator* password on a Unix system. The *passwd* tool can also change other user's passwords by applying the username as an argument.

Prerequisites
This procedure requires the use of other procedures in advance. You have to be an *Administrator* to be able to enter the *passwd* command successfully. The command has to be issued on the host system and the system can for example be booted through single-user mode, as described in Procedure 7.1, Procedure 7.2, Procedure 8 and Procedure 9. You can also use the *chroot* command, as described in procedure 11, to enable root privileges on the host system. In that way you can change the root directory of the guest operating system to the directory where you mounted the host operating system.

Approach
This is how you can reset the *Administrator* password when the user has already entered single-user mode. See *Prerequisites* if you wonder how to enter single-user mode.

1. Enter single-user mode by applying either Procedure 7.1, Procedure 7.2, Procedure 8 or Procedure 9 depending on which host operating system used.
2. Write *passwd* to change the Administrator password. You do not need to specify an account because you are already logged in as root.
3. Type a new password and confirm by re-typing the password again.
4. The *Administrator* password has now been changed.
5. Type *shutdown -r now* to reboot the machine.

Variations

- A variation is that you can enter */usr/bin/passwd root* instead of *passwd* when resetting the *Administrator* password, as also mentioned in Procedure 9.

6.2 Tools for Password Recovery

In this section the different password recovery tools are presented.

Tool 7: Ophcrack Live CD 2.1.0

Tool	**Short description**	**Operating System**	**Objective**	**Prerequisites**
Tool 7	Ophcrack Live CD 2.1.0	Windows XP, Windows Vista, Windows Server 2008, Windows 7 RC	Password recovery	Empty writable CD and optional rainbow tables

Table 6.9: Tool 7 *Ophcrack Live CD 2.1.0.*

Description
At Ophcrack's web pages there are two different live CDs available for download, *Ophcrack XP Livecd 2.1.0* and *Ophcrack Vista Live CD 2.1.0.* We experimented with the Vista live CD version of Ophcrack available from [44]. As described earlier in Section 2.4.2.1, this tool can be used to recover a lost administrator password. It has the ability to use rainbow tables, which is explained in Section 2.1.2.3, to recover a password from a hash value. The tool is applicable for both LM and NT hash values. LM and NT hashes are described in detail in Section 2.3.1.2. The live CD we use is designed for Windows Vista, mainly because it contains some free rainbow tables designed to crack the NT hash used in Windows Vista. This means that this tool is also applicable to other Windows operating systems utilizing the NT hash. For this procedure we used two administrator accounts, *Administrator* and *nerblak*, to see if Ophcrack managed to retrieve the passwords.

Prerequisites
The tool has to be burned on an empty writable CD. The tool can be extended with additional rainbow tables. The tool and additional rainbow tables can be found on *Ophcrack's* web page [44].

Approach
This approach demonstrates how *Ophcrack's Vista Live CD* can be used on a machine running Windows Vista to recover a lost password. In this approach *Hegra* was the machine we experimented with. Presently *Ophcrack* delivers two variants of their live CDs. The *Vista Live CD* may also work on other systems utilizing NT hash, like Windows Server 2008 and Windows 7.

1. Download the tool from [44].

2. Burn the image (iso file) on the empty writable CD using for instance ImgBurn [90].
3. Boot the host system and press F9 (this may vary from system to system) to enter the boot menu.
4. Insert the boot disk into the CD-ROM drive.
5. Select to boot from CD-ROM.
6. You should now have entered the *Ophcrack Live CD boot menu*.
7. Select *Ophcrack Graphic mode* and press *enter*, or wait for it to launch automatically. If you prefer to use the text-based version of Ophcrack, this can be selected instead. The text-based version will not be explained in further details.
8. A graphical environment called *Fluxbox* will appear and you are asked to select which partition to crack. In our case *0* (/mnt/sda1/Windows/System32/config) was the correct partition to choose. If only one partition exists, Ophcrack will choose this partition automatically, and no interaction is required at this point.
9. A window presenting the users and their corresponding LM- and/or NT hash are now listed. The same window shows the progress bar for the cracking process.
10. The *Vista free* rainbow tables, which can be found at [44] and that are already included on the Ophcrack Vista Live CD, are then loaded into RAM, before the cracking process begins.
11. Additional rainbow tables can be included by pressing *Stop* in the menu at the top of the window and then pressing the *Tables* button, also in the top menu. Click *Install* to find the location of the preferred rainbow tables, highlight the directory containing the rainbow tables and press *Choose*. Then enable the rainbow table by highlighting the corresponding name, for example *Vista special*, and press the green button located at the bottom left of the window, which will enable the tables. Then press *OK* and you will be back in the original window. The new tables will appear in the *Table* overview, and to start the cracking process press *Crack* in the top menu.
12. Wait for the cracking process to finish.
13. If the password fulfill the requirements of the rainbow tables used, such as for example the maximum password length possible, the password will be displayed. As already described in Section 2.3.1.2 on page 31, Windows Vista use only NT hash as default, and the password should therefore be displayed in the column labeled *NT Pwd*, with the assumption that Ophcrack was able to crack the password. If Ophcrack did not recover the password, *not found* will appear in the *NT Pwd* column. If no password is set for the account the *NT Pwd* column will display *empty*.

14. To exit Ophcrack press *Exit* at the top menu and then press any key when the *Press a key to exit* appears. Then you should be prompted with a question asking if you want to shutdown, type *y* to shutdown.

15. You should now be able to log in to the administrator account on the host system by using the retrieved password.

Variations

- We did also download and run the Ophcrack XP Live CD 2.1.0. This live CD is very similar to the Ophcrack Vista Live CD 2.1.0 described above. The tool serve the same purpose, but this live CD is designed for Windows XP, mainly because it contains some free rainbow tables designed to crack the *LM hash* used in Windows XP. The procedure is therefore the same for the Ophcrack XP Live CD as for the Ophcrack Vista Live CD; the only difference is that the Ophcrack XP Live CD includes the *XP free small* tables instead of the *Vista free* tables, which can be found at Ophcrack's web pages [44]. Another difference is that the password, if Ophcrack was able to crack it, will be displayed in the *LM Pwd 1* and *LM Pwd 2* columns when using the Ophcrack XP Live CD. The password will then be the concatenation of the content in *LM Pwd 1* and *LM Pwd 2* columns for the desired user. The same is the case when using the Ophcrack Vista Live CD on a Windows operating system that uses LM hash, such as Windows XP. We used *Fredriksholm*, the machine with Windows XP installed, when running the Ophcrack XP Live CD.

Tool 8: John the Ripper 1.7.2

Tool	Short description	Operating System	Objective	Prerequisites
Tool 8	John the Ripper 1.7.2	Windows XP, Linux systems with hash values generated with *crypt* (only some algorithms)	Password recovery	Empty writable CD

Table 6.10: Tool 8 *John the Ripper 1.7.2*.

Description
We used John the Ripper 1.7.2 [68], which is a password cracking tool, to recover the password of two different Administrator accounts. The two different Administrator accounts have passwords of different strength. This was carried out on two different systems, as will be described below. In the first approach we choose to use Ubuntu 8.10 live CD as a guest operating system and Fredriksholm, which has Windows XP installed, as a host system. In the second approach we choose to use Ubuntu 8.10 live CD as guest operating system and Kristiansten, which has Ubuntu Server 8.10 installed, as host system. This is to demonstrate both a Windows- and a Linux system. This will require that the necessary password files are fetched and that John the Ripper is installed through the Ubuntu live CD. This means that John the Ripper has to be installed again if you reboot and the password files must be re-located.

Prerequisites
This approach requires an empty writable CD and that you have Internet connection. Internet connection is important because you are required to download a couple of software packages and wordlists in this approach.

Approach
We have included two approaches for John the Ripper, one approach using a Windows XP as the host operating system and one approach using Ubuntu Server 8.10 as the host operating system. In both cases we are using an Ubuntu live CD as a guest operating system.

The first approach describes how John the Ripper is used through an Ubuntu live CD on a computer running Windows XP (Fredriksholm).

1. Apply the first 12 steps described in Procedure 10 without inserting the USB drive mentioned in step 4.
2. Type *mkdir john* to create a working directory.
3. Go to *System -> Administration -> Software Sources* in the top menu of Ubuntu. This will open a window where you now should enable *Community-maintained Open Source software (universe)* by checking the corresponding box. Then press *Close*.
4. Click *Reload* to update the newly selected sources.
5. Write *aptitude install john bkhive* to install John the Ripper and bkhive on the guest operating system and press *Y* and *Enter* to confirm. bkhive dumps the syskey/bootkey from a Windows NT/2K/XP/Vista system hive.
6. Enter the directory you created by writing *cd john/*.
7. Write *bkhive /mnt/os/WINDOWS/system32/config/system key* to dump the syskey to a file named *key*.

8. Enter *samdump2 /mnt/os/WINDOWS/system32/config/SAM key >hashes.txt* to dump the Windows password hashes to a file called *hashes.txt*. *samdump2* is the program performing this action.

9. To start the cracking process type *john hashes.txt.* This will make John the Ripper try all the cracking modes. These modes are described in Section 2.4.2.2. It is also possible to use more specific commands, for example commands to only use one of the modes, and these commands are also described in 2.4.2.2.

10. The passwords for the user accounts should now have been cracked, depending on the strength of the password. The stronger the password is, the longer it may take to crack it, if it is possible to crack at all.

11. To see the cracked passwords in another format you can type *john –show hashes.txt.*

This second approach describes how John the Ripper is used through the Ubuntu live CD on a machine running Ubuntu Server 8.10 (Kristiansten).

1. Apply the first 10 steps described in Procedure 10 without inserting the USB drive mentioned in step 4.

2. Write *mount /dev/sda1 /mnt/os* to mount the ext3 file system used in Kristiansten. *ext3* and *sda1* was recognized in step 9 in Procedure 10.

3. Type *mkdir john* to create a working directory.

4. Write *aptitude install john* to install John the Ripper on the guest operating system and press *Y* and *Enter* to confirm.

5. Enter the directory you created by writing *cd john/.*

6. Write *umask 077* to set the default permission mode on newly created files and directories.

7. Then write *unshadow /mnt/os/etc/passwd /mnt/os/etc/shadow > mypasswd* to combine the information found in the passwd file and the shadow file and dump it to the newly created mypasswd file.

8. To start the cracking process type *john mypasswd.* This will make John the Ripper try all the cracking modes. These modes are described in Section 2.4.2.2. It is also possible to use more specific commands, for example commands to only use one of the modes, and these commands are also described in 2.4.2.2.

9. The passwords for the user accounts should now have been cracked, depending on the strength of the password. The stronger the password is, the longer it may take to crack it, if it is possible to crack at all.

10. We experienced that the hash values created by the *MD5* hash algorithm was tested by *john* but the hash values created with *SHA-512* was never tested by John the Ripper. On John the Ripper's web site [68], it is only specified that several *crypt(3)* password hash types are supported, but we suspect that *SHA-512* is not supported yet.

11. To see the cracked passwords in another format you can type *john –show hashes.txt.*

Variations

- There are three modes and several options available together with the *john* command. You can find an overview of these modes and options by writing *man john.*

Tool 9: Cain & Abel v4.9.30

Tool	Short description	Operating System	Objective	Prerequisites
Tool 9	Cain & Abel v4.9.30	Windows XP, Windows Vista, Windows Server 2008, Windows 7 RC	Password recovery	Password and system file, rainbow tables

Table 6.11: Tool 9 *Cain & Abel v4.9.29.*

Description
Cain & Abel is a password recovery tool with the support of cracking hash values created by many different hash algorithms. The tool is created to run on Windows systems but can crack hash values created on many other systems as well. The tool supports dictionary, brute-force and cryptanalytic attacks, as described in 2.1.2.2.

Prerequisites
In this approach, you first need to obtain the password file that you want to crack with Cain & Abel. We are going to show how to perform a cryptanalytic attack and therefore we need rainbow tables. We downloaded free rainbow tables from [95]. It is recommended that you download the rainbow tables before you start on this approach, since it may take quite a long time. The same rainbow tables can also be generated using Tool 12 described at page 120.

Approach
This approach is carried out on a different machine than the machine containing the target password files. We obtained the password files (SAM and system) that we want to crack from Fredriksholm, which is running Windows XP. We have chosen to import additional rainbow tables which can crack LM hashes.

1. Download and install the tool from [72] on the machine you are going to use for the cracking process. We are using Cain & Abel version 4.9.30.

2. Fetch the Windows password files (SAM and system file) that you want to crack. See Procedure 10. We obtained the password files (SAM and system file) from Fredriksholm.

3. Download the rainbow tables from [95]. The rainbow tables are distributed with the BitTorrent protocol and the tracker can be found at [96]. We suggest that you download these rainbow tables in advance because it takes quite a long time. When we downloaded the files they were packed as lzma. See this page [95] for a more detailed description.

4. Run the Cain & Abel program as Administrator. You can do that by right-clicking on the Cain & Abel program and select *Run as administrator*.

5. Click on the *Cracker* tab in the top menu. Then click on the white sheet in the main window to activate it, and press the *Insert* button on your keyboard to import the password and system file.

6. Select *Import Hashes from a SAM database*.

7. In the *SAM Filename* field, locate the SAM file that you want to crack by using the button to the right of the field.

8. In the *Boot Key (HEX)* field you need to locate the system file. When the button next to the field is pressed a *Syskey Decoder* window appears. From this you need to locate where the desired system file is saved. When you have opened the system file, and have returned to the *Syskey Decoder* window you need to copy (CTRL+C) the HEX value.

9. Then press *Exit* to return to the *Add NT Hashes from* window.

10. Now, paste (CTRL+V) the HEX value in the *Boot Key (HEX)* field in the *Add NT Hashes from* window.

11. Click on *Next*. The available user accounts and their corresponding hash values should now have been loaded.

12. Mark the user accounts that you want to crack the password for. Then right-click and select *Cryptanalysis Attack->LM Hashes->via RainbowTables (RainbowCrack.*

13. To be able to use the rainbow tables that we downloaded earlier, a charset file has to be specified. This file is closely connected to the rainbow tables and it describes to the cracking program which set of characters to perform password cracking with. In our case we created a charset.txt file with the following line: alpha-numeric-symbol32-space = [ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$%^&*()-_+=~‘[ ]{}|\:;”’< >,.?/ whitespace]

14. From the *LM Hashes cryptanalysis* window click on the *Charsets* button on the right hand menu to import the charset.txt file.

15. Click on the *Add Table* button in the right hand menu and locate the rainbow tables. Open all the tables by using the *CTRL+A* command and press *Open*.

16. Click on the *Start* button located in the bottom right to start the cracking process.

17. If the cracking process succeeds, the user accounts and their corresponding passwords will be displayed in clear text which means that the password has been recovered.

Variations

- In this tool other types of attacks can be used. We chose to use the cryptanalysis attack, but there also exists dictionary and brute-force attacks.
- We used rainbow tables designed to crack LM hashes, but other types of rainbow tables can also be used.
- There exist also different rainbow table distributors. We chose to download free LM tables from the Shmoo Group [95].
- This tool can also crack hash values produced by many other hash algorithms than the one presented in this approach.

Tool 10: LCP 5.04

Tool	Short description	Operating System	Objective	Prerequisites
Tool 10	LCP 5.04	Windows XP, Windows Vista, Windows 2008 server, Windows 7 RC	Password recovery	Password file

Table 6.12: Tool 10 *LCP 5.04*.

Description
LCP 5.04 [74] is also a popular password cracker with the support of cracking hash values produced by various hash algorithms. The tool is also created to run on Windows systems. The tool supports dictionary, brute-force and hybrid attacks.

Prerequisites
In this approach, you first need to obtain the password and the corresponding system file that you want to crack with LCP.

Approach
This approach is carried out on a different machine than the machine containing the target password files. We obtained the password files (SAM and system) that we want to crack from Hegra, which is running Windows Vista.

1. Download and install the tool from [74] on the machine you are going to use for the cracking process. When we downloaded the tool version 5.04 was the latest.

2. Fetch the Windows password files (SAM and system file) that you want to crack. See Procedure 10. We obtained the password file (SAM and system file) from Hegra.

3. Run the LCP program.

4. Click *Import->Import from SAM file* in the top menu of the program.

5. Select the desired SAM file, in our case Hegra's SAM file. Select Hegra's system file as *Startup key in a registry.*

6. Press *OK* to import the hash values. Select *Yes* if a window is prompted.

7. To start the cracking process press *F4*. This may take quite a long time since a brute-force approach will begin if not the dictionary and the hybrid approach are able to crack the password.

8. In this approach *password* and *Hax0R* was used as passwords. The approach revealed *password* very fast but *Hax0R* took too long time to crack so we canceled after a while. To be sure to crack the passwords within a reasonable time, we recommend using Tool 9 with rainbow tables.

Variations

- Hash values produced by other hash algorithms can also be cracked with this tool.

6.3 Supplementary Tools

In this section supplementary tools that are used in addition to other procedures and tools to reset or recover a lost password are presented.

In this section supplementary tools used in the process of password resetting or/and recovery, are presented.

Tool 11: fgdump 2.1.0

Tool	**Short description**	**Operating System**	**Objective**	**Prerequisites**
Tool 11	fgdump 2.1.0	Windows XP, Windows Vista, Windows 2008 server, Windows 7	Dump SAM file	None

Table 6.13: Tool 11 *fgdump 2.1.0*.

Description
A tool called *fgdump 2.1.0* can be used to retrieve information obfuscated in a *SAM* file. It is a program executable from most windows system. It also has support for dumping hash information through a network.

Prerequisites
In this approach no prerequisites were required.

Approach
We will in this approach show how to dump the windows *SAM* file, when the user has access to the system.

1. Download the tool from [75] on the machine you are going to dump the password file. When we downloaded the tool version 2.1.0 was the latest.

2. Unzip the downloaded file to a folder on your desktop.

3. Double click on the *fgdump.exe* file and it will fetch the necessary information and dump it on a file placed in the same folder as the executable file (fgdump.exe).

4. A file called *127.0.0.1.pwdump* containing the revealed *SAM* file is available. This can for instance be interpreted with *WordPad*.

Variations

- This tool has many options available. These can be found on the official *fgdump* web page [75] or by running the tool in Windows Command Console with the following command: *fgdump.exe -h*.

Tool 12: Winrtgen 2.8

Tool	Short description	Operating System	Objective	Prerequisites
Tool 12	Winrtgen 2.8	Windows XP, Windows Vista, Windows 2008 server, Windows 7	Generate Rainbow tables	None

Table 6.14: Tool 12 *Winrtgen 2.8*.

Description
This tool is a graphical version of rtgen.exe and rtsort.exe from the software called RainbowCrack [67]. The tool can generate rainbow tables for many different hash algorithms. In this thesis under Tool 9 we use tables that are generated from either *Winrtgen* or *rtgen.exe*. When we used the rainbow tables, we did not create them, as this is very time consuming, but using this tool the same tables can be generated. Winrtgen can be executed on any Windows system, and if large tables are to be generated sufficient amount of storage capacity is needed. A charset file is already included with the tool, but if specific characters are needed, this file should be edited.

Prerequisites
In this approach no prerequisites were required.

Approach
In this approach we will demonstrate how to generate the rainbow tables that was used in Tool 9.

1. Download the tool from [77].
2. Unzip the tool using for instance Winzip [94].
3. Execute the newly extracted *winrtgen.exe*.
4. When the graphical interface opens click on *add table*, and various options can then be changed.

5. To acquire the same tables as described in Tool 9 the following options should be chosen:
    - *Hash*: lm
    - *Min Len*: 1
    - *Max Len*: 7
    - *Index*: 0
    - *Chain Len*: 15200
    - *Chain Count*: 67108864
    - *N° of tables*: 64
    - *Charset*: all-space-DG
6. If the same name is desirable for each of the tables press Edit and make the necessary changes in the charset file.
7. When the correct properties have been chosen, click benchmark to measure factors such as *Table precomputation time* and *Max cryptanalysis time*.
8. To start the generation of tables, press *OK*. When the next window appears, press *OK* again.
9. This process can take a large amount of time depending on the computer the tables are being generated on.

Variations

- Tables for other hashes can be made, such as NT and MD5.
- The tables can be specified to have different properties, so that the overall process of revealing the password can be optimized for the computer handling the password cracking process. If for instance a computer with for instance a slow CPU is used for the cracking process, a large set of tables with short chains is preferable, thus requiring more physical storage capacity. If the cracking process is performed on a supercomputer, it might be clever to trade some of the physical space with time required for cryptanalysis.
- Distributing the generation of tables on many computers, can be smart in terms of saving time.

Chapter 7

Results

This chapter will present the general results found, and includes results regarding the procedures and tools that have already been described and the production of our self made tool named YALP. The results obtained from the empirical password study are presented in Chapter 8, as we have chosen to separate this study from the other results. Note that the results presented in Chapter 7 and Chapter 8 will be further discussed in Chapter 9.

7.1 Procedures and Tools

This section serves the purpose of presenting the results related to the procedures and tools described in Chapter 5 and Chapter 6 respectively. We have separated between procedures and tools used to *reset* the administrator password and procedures and tools used to *recover* the administrator password.

The results are presented in tables below, and in the tables we have included a column labeled *Time*. These times are influenced by the hardware specifications of the computers used in this experiment and these specifications can be found in Section 3.1 in Chapter 3. The execution times are supposed to give an indication of the scope of the specific procedure or tool, and based on that a comparison between the procedures and tools can be made. Note that the times given do not include downloading of files, burning of CDs or other preparations, only from the system is booted to you are logged in as Administrator. For instance in Procedure 6, we will not include the time it takes to create the *Windows Password Reset Disk*. We have also defined an *ease of use* field to give an indication of how difficult the procedure is to perform on a scale from 1 to 3, where 1 is easy and 3 is hard. The last column lists which operating systems that the procedure or tool is applicable to.

In the tables below, under the field *Operating System*, we have chosen to present the operating system tested on in the approach for the specific procedure or tool which was subject to password resetting or password recovering. The procedure or tool may run on other operating systems, and can possibly perform password resetting or password recovering of passwords originating from many other operating systems than the ones presented in the table below. The supported operating system for the specific procedure or tool is described more in detail in Chapter 6 and 5, and in Section 2.4. An overview of operating systems supported by procedures and tools can also be found in Chapter 4.

7.1.1 Password Resetting

In this section two tables are presented. The first table (Table 7.1) contains some results we gained when the procedures related to password resetting, presented in Chapter 5, were carried out.

Procedures	**Time**	**Ease of use**	**Operating System**
Procedure 1: XP Repair Backdoor	23.05 min	3	Windows XP
Procedure 2: Edit Shadow File in Unix Systems	6.43 min	3	Ubuntu Server 8.10
Procedure 3: Shortcut for Bypassing Regular Login Procedure	0.5 min	1	Windows XP
Procedure 4: Replace a Pre-login Executable File	8.03 min	2	Windows Vista
Procedure 5: Use Mac OS X Installation Disk	3.58 min	1	Mac OS X Leopard
Procedure 6: Windows Password Reset Disk	1.5 min	2	Windows Vista

Table 7.1: Key statistics from the password resetting procedures.

Tools	**Time**	**Ease of use**	**Operating System**
Tool 1: PC Login Now 2.0.1	4.53 min	1	Windows Vista
Tool 2: Offline NT Password & Registry Editor	2.14 min	1	Windows Vista
Tool 3.1: BartPE with Password Renew	4.58 min	2	Windows XP
Tool 3.2: BartPE with Windows Gate	5.30 min	2	Windows XP
Tool 4: Kon-Boot	4.18 min	1	Windows Vista
Tool 5: DreamPackPL	7.93 min	3	Windows XP
Tool 6: passwd	2.95 min	2	Ubuntu Server 8.10

Table 7.2: Key statistics from the password resetting tools.

The second table (Table 7.2) presents the results we achieved with password resetting tools presented in Chapter 6. The *Time* field only yield for the operating system and hardware mentioned under the approach of each procedure or tool as presented in Chapter 5 and Chapter 6. This means that if another supported operating system or hardware is experimented with in a recreation of our experiments, the time may vary.

7.1.2 Password Recovery

In this section results we got from the various tools used for password recovery is presented in a table. We described no procedures to recover the passwords, thus no results for this are presented. In Table 7.3 below you may also notice that John the Ripper is mentioned two times. The reason for this is that we described two approaches in Section 8 at page 112. In all of the approaches described in Section 6.2 we experimented with two different passwords, *password* and *Hax0R*. For each password recovery tool we have included a field denoting which of the passwords that were revealed. On the last tool, Tool 10 we chose to stop the cracking process after 15 minutes. This was because the tool began with an exhaustive brute-force search looking for the password *Hax0R*, which could take a very long time.

Tools	**Time**	**Ease of use**	**Operating System**	**Password revealed**
Tool 7: Ophcrack Vista Live CD 2.1.0	11.77 min	1	Windows Vista	password
Tool 8: John the Ripper	1.52 min	2	Windows XP	password, Hax0R
Tool 8: John the Ripper	1.5 min	2	Ubuntu Server 8.10	password
Tool 9: Cain & Abel	41.62 min	2	Windows XP	password, Hax0R
Tool 10: LCP 5.05	15 min	1	Windows Vista	password

Table 7.3: Key statistics from the password recovery tools.

7.2 YALP

In Chapter 5 we identified various procedures which demanded that the users had a lot of patience and some technical skills. Some procedures had requirements that were hard to fulfill and some procedures demanded a lot of time. Many of the procedures were very cumbersome and required a lot of interaction from the users.

As a result of this, we decided to contribute with a tool that simplifies the problems mentioned above. Our contribution resulted in a tool called *Yet Another Local Password (tool) (YALP)*. YALP is actually a bootable live CD containing some tools created by the authors of this thesis, and some other open source tools available on the Internet. We have chosen to refer to YALP as a tool instead of a distribution containing many tools. The main tools included in YALP have the common purpose of handling passwords in various ways. By handling passwords, we mean to set, reset, recover or protect the administrator/user password that is used when the user logs in to an operating system.

The first section below will give a more thoroughly description of YALP. The last section will then describe the structure of the tool and its features.

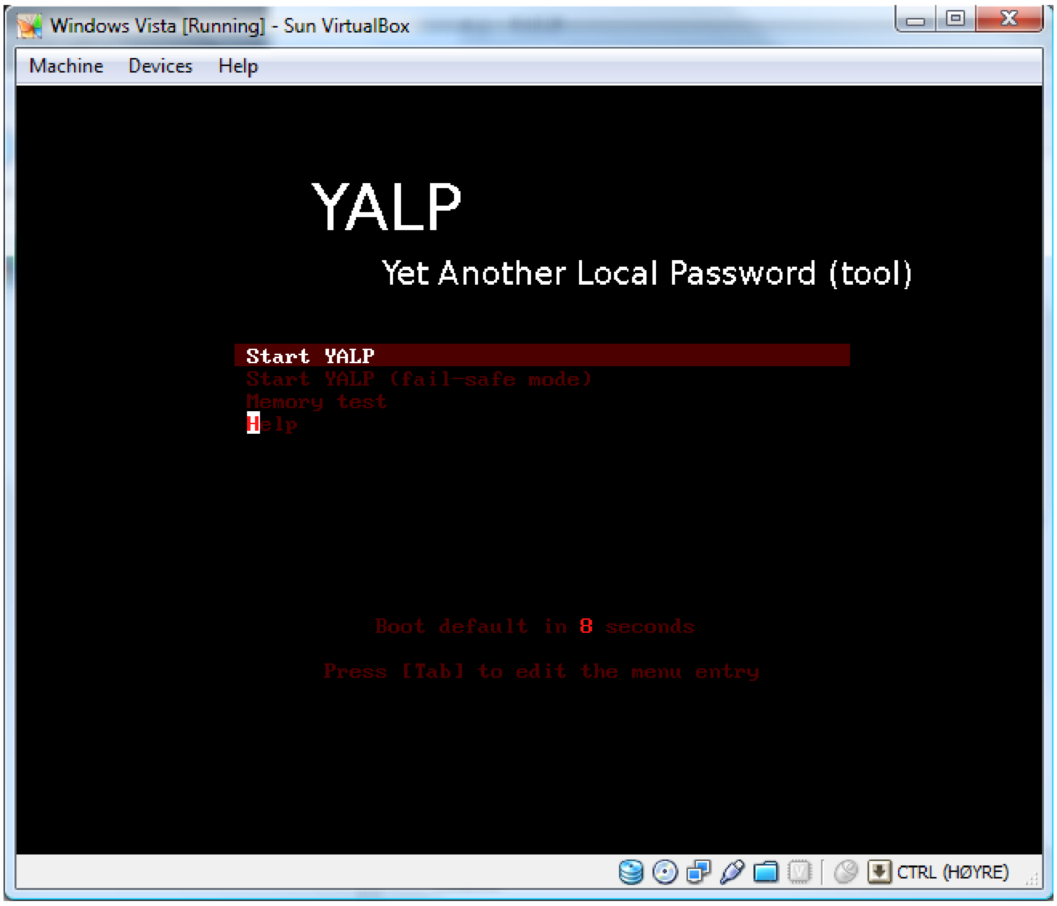
7.2.1 Description of YALP

YALP is a collection of tools mainly designed to handle passwords in various ways. The tool handles only passwords accessed locally and is not designed to handle passwords through a network. Handling passwords through the network may be supported in the future. YALP is based on the operating system *Debian Linux* [97] using the standard Linux kernel [98] as the underlying kernel. To describe the tool more precisely, you can say that YALP is a collection of tools embedded into a Debian Live distribution (Lenny) [99] created with a program called *live-helper* [100]. A more detailed description on how to create YALP is presented in Appendix A. Most of the tools created for YALP are implemented as either *perl* or *bash* scripts. The scripts are included in Appendix B.

In YALP we have added support to mount most of the operating systems described in this thesis. We had to build our own kernel to support mounting of the *UFS* file system used by FreeBSD 7.1. This means that we did not use the pre-compiled kernel that is included by default in live-helper. YALP is a text-based tool, thus no graphical interface is included. A graphical environment can easily be included; however this will make the tool much larger in size. YALP includes the default tools recognizable in most Linux distribution, and because YALP is based on a Debian distribution it has inherited most of the standard Debian/Linux tools. The tool YALP will boot on most x86 architectures, and support for more systems can be realized with customizations during the build of YALP as described in Appendix A.

In the configuration we chose to set the keyboard language to Norwegian, because this is the keyboard language the authors of this thesis use. Some other minor configurations were also applied, and these can be found in the appendix mentioned above.

Figure 7.1 shows YALP booted on top of a Windows Vista operating system, installed in VirtualBox [78].



```
YALP
        Yet Another Local Password (tool)

Start YALP
Start YALP (fail-safe mode)
Memory test
Help




              Boot default in 8 seconds

          Press [Tab] to edit the menu entry
```

Figure 7.1: YALP initial welcome screen.

In Figure 7.1 the Syslinux boot loader menu used by YALP is displayed. The menu enables the possibility to boot YALP in fail-safe mode, perform a memory test, display some help, or boot YALP normally (Start YALP). The Syslinux menu will timeout in 10 seconds, and if no interaction has been done, the default option (Start YALP) will be automatically selected.

7.2.2 Contents of YALP

As mentioned in last section, YALP contains various tools to handle passwords in different ways. All of the scripts created by the authors of this thesis are text based and interacts with the user. The user types in an answer to a question with his/her keyboard and send the answer to the script by pressing *enter*.

The most important script created specifically for YALP is the startup script, *startup.sh*. This script is initiated when all the boot processes have completed. The script is included in Appendix B.1 and is presented in a flow diagram in Figure 7.2.

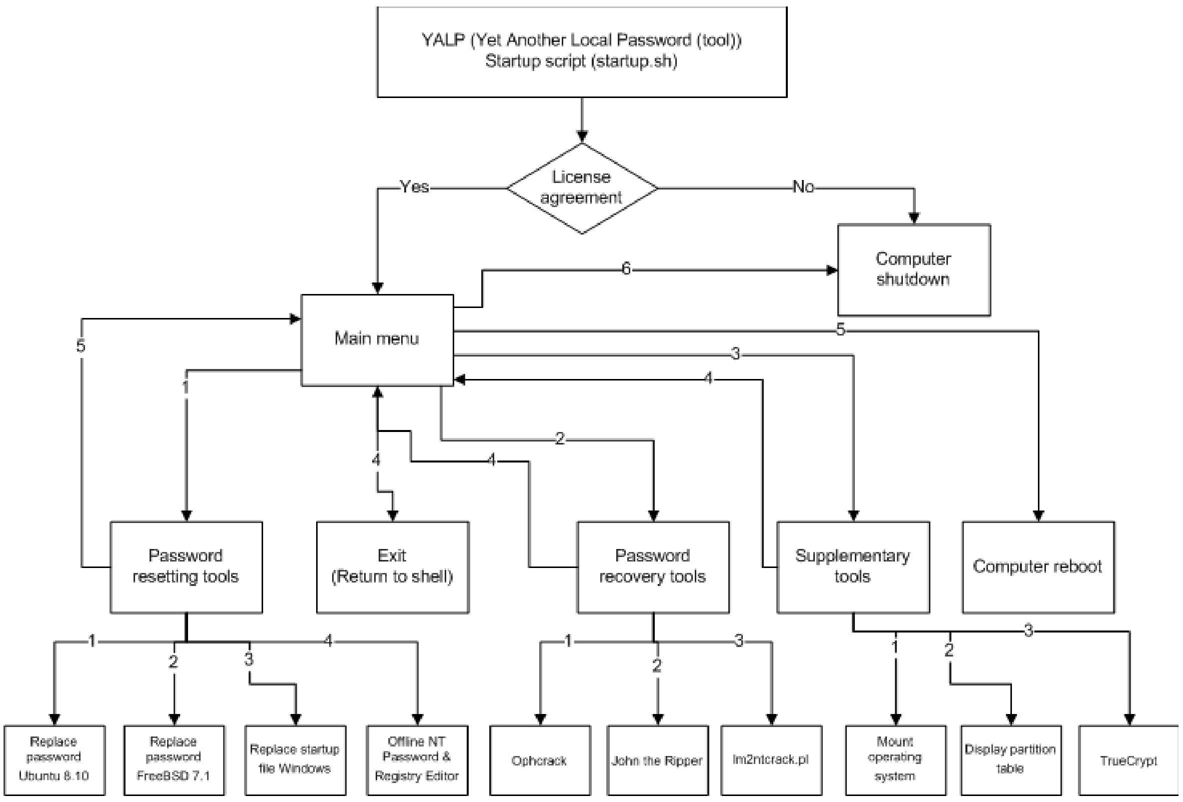
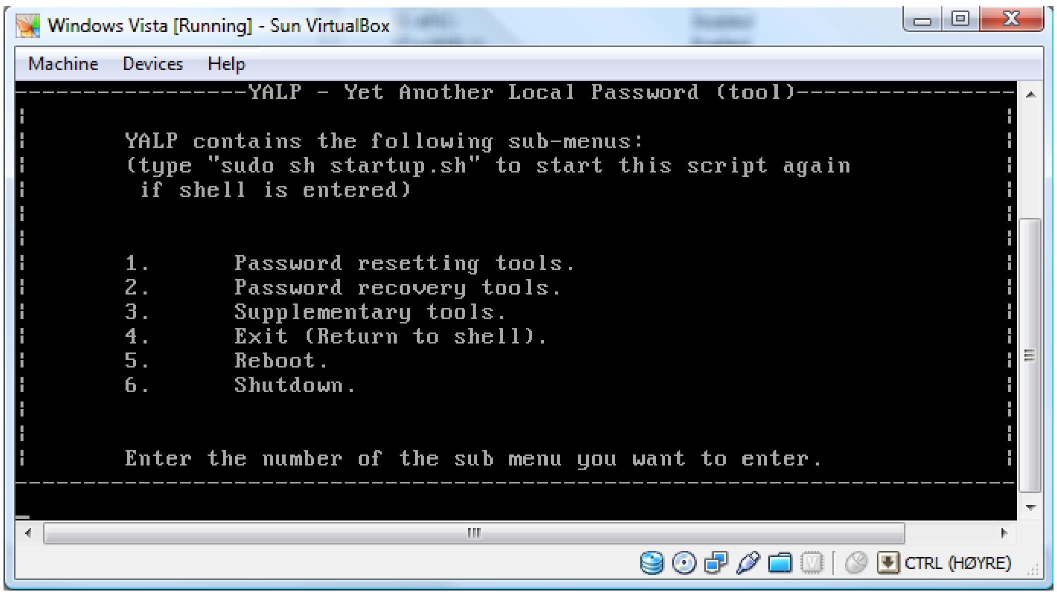


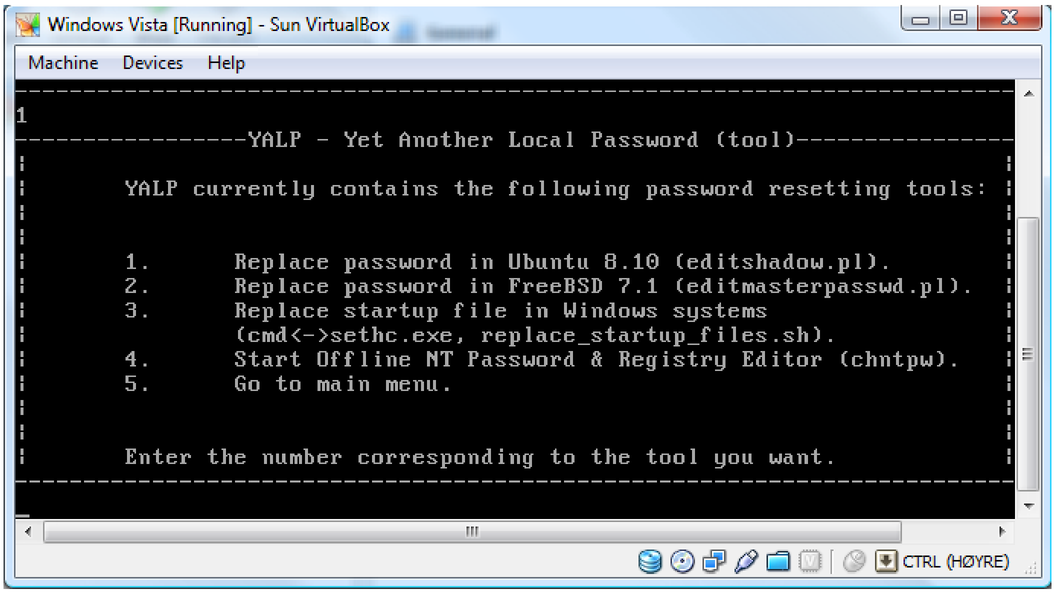
Figure 7.2: A flow chart displaying YALP's startup script.



```
-----------------YALP - Yet Another Local Password (tool)----------------
|                                                                       |
|       YALP contains the following sub-menus:                          |
|       (type "sudo sh startup.sh" to start this script again           |
|        if shell is entered)                                           |
|                                                                       |
|                                                                       |
|       1.      Password resetting tools.                               |
|       2.      Password recovery tools.                                |
|       3.      Supplementary tools.                                    |
|       4.      Exit (Return to shell).                                 |
|       5.      Reboot.                                                 |
|       6.      Shutdown.                                               |
|                                                                       |
|                                                                       |
|       Enter the number of the sub menu you want to enter.             |
-------------------------------------------------------------------------
```

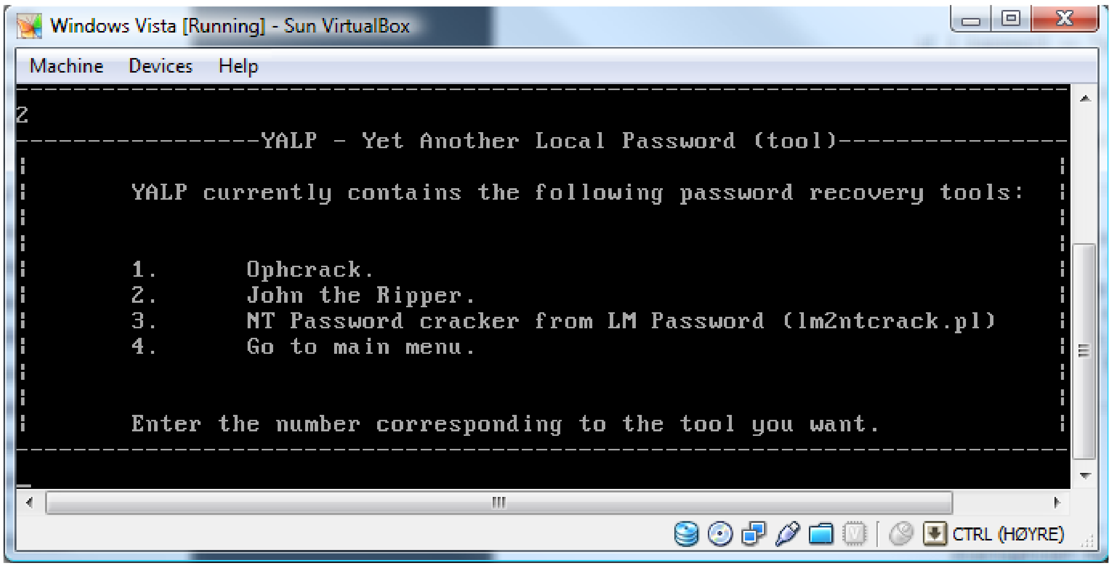
Figure 7.3: YALP's main menu.

At first *startup.sh* presents a license agreement, if the user types anything else than yes, the system will be shut down. If the user types *yes*, the user will be presented with the main menu. From this menu various choices can be made as shown in Figure 7.3.



```
--------------------------------------------------------------------------------
1
-----------------YALP - Yet Another Local Password (tool)----------------
|                                                                        |
|       YALP currently contains the following password resetting tools:  |
|                                                                        |
|                                                                        |
|       1.      Replace password in Ubuntu 8.10 (editshadow.pl).         |
|       2.      Replace password in FreeBSD 7.1 (editmasterpasswd.pl).   |
|       3.      Replace startup file in Windows systems                  |
|               (cmd<->sethc.exe, replace_startup_files.sh).             |
|       4.      Start Offline NT Password & Registry Editor (chntpw).    |
|       5.      Go to main menu.                                         |
|                                                                        |
|                                                                        |
|       Enter the number corresponding to the tool you want.             |
--------------------------------------------------------------------------------
_
```

Figure 7.4: Password resetting tools menu.

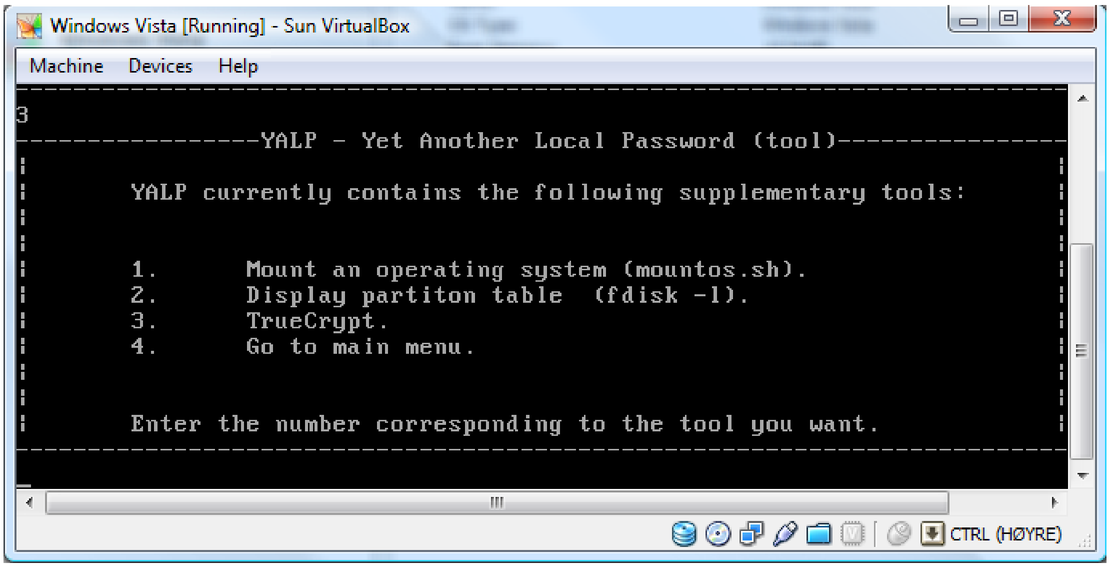


```
--------------------------------------------------------------------------------
2
-----------------YALP - Yet Another Local Password (tool)----------------
|                                                                        |
|       YALP currently contains the following password recovery tools:  |
|                                                                        |
|                                                                        |
|       1.      Ophcrack.                                                |
|       2.      John the Ripper.                                         |
|       3.      NT Password cracker from LM Password (lm2ntcrack.pl)     |
|       4.      Go to main menu.                                         |
|                                                                        |
|                                                                        |
|       Enter the number corresponding to the tool you want.             |
--------------------------------------------------------------------------------
_
```

Figure 7.5: Password recovery tools menu.

The user can type in a number corresponding to the next level of the script. This is presented in the flow diagram where the arrows have the number corresponding to its menu/program. If for instance the user press *1* and *enter* the user will be directed to a new menu. This menu shown in Figure 7.4 displays the various password resetting tools. On the other hand, if the user types *2* from the main menu, the user will be directed to the *password recovery tools* menu. This menu is presented in Figure 7.5.

The last menu available in the startup script is the *supplementary tools* menu. This menu will link to programs that do not fall under the menus previously mentioned. The *supplementary tools* menu is displayed in Figure 7.6.



```
--------------------------------------------------------------------------
3
-----------------YALP - Yet Another Local Password (tool)-----------------
|                                                                        |
|       YALP currently contains the following supplementary tools:       |
|                                                                        |
|                                                                        |
|       1.      Mount an operating system (mountos.sh).                  |
|       2.      Display partiton table  (fdisk -l).                      |
|       3.      TrueCrypt.                                               |
|       4.      Go to main menu.                                         |
|                                                                        |
|                                                                        |
|       Enter the number corresponding to the tool you want.             |
--------------------------------------------------------------------------
_
```

Figure 7.6: Supplementary tools menu.

As you can see from Figure 7.3, YALP can be divided into three subcategories of tools; *Password resetting tools*, *Password recovery tools* and *Supplementary tools*. Under the category *Password resetting tools*, we have created three scripts. The first script can replace the password hash value (MD5) in the *shadow* file in Ubuntu 8.10. This script is an implementation of Procedure 2 described at page 75. This script worked as planned, and is included in Appendix B.3. We also created a similar script for the *master.passwd* file used by FreeBSD, but have not yet succeeded to update the password database files in FreeBSD and this is left to further work. Although the database files are not updated, the script to replace the MD5 hash in *master.passwd* (FreeBSD) is working. This is why we have included this script in YALP.

In the category *Password resetting tools* we also included a script that would ease Procedure 4 described at page 78. This bash script is called *replace_startup_files.sh* and can be viewed in Appendix B.2. The script will replace the *sethc.exe* file with *cmd.exe* and take a backup of *sethc.exe* named *sethc.exe.bak*. After applying this tool changes to the underlying host operating system has been made. When the host system is booted, right before login, the user can press shift key five times and instead of the sticky keys program the user will be presented with the Windows command console. The user can then easily reset a password for any account in the User Accounts menu by typing *control userpasswords2*. The user is granted system privileges before he is logged in. We tested our script on Windows Vista and it worked as planned, the sticky keys program was replaced. The revert function, which restores the sticky keys program that we took a backup of (*sethc.exe.bak*), also worked as planned.

In the category *Password resetting tools* we also included the tool *Offline NT Password & Registry Editor* made by Petter Nordahl-Hagen [56]. This tool is described more in detail in Section 2.4.1.1 and Section 2.

Under the category *Password recovery tools* we included two excellent password recovery tools, Ophcrack [44] and John The Ripper [68]. Since YALP for the time being is text based; only the command line version of Ophcrack is included. The wordlist that is used for the empirical password study, which is presented in the next chapter, is also included in YALP in the same folder as where the scripts are located */home/user*. When John the Ripper is used to crack a LM hash, the password is only revealed in uppercase. This is as mentioned in Section 2.3.1.2, the password is converted to uppercase before it is hashed. With all of the Windows systems we are experimenting with, either just a NT hash is used or both the NT and LM hash is used. The LM hash is included for backward compatibilities with earlier systems. Because of the weaknesses that arrive with the LM hash, John the Ripper attacks this hash. The problem arise when the password is a mixture of upper and lower case letters. Since the output of John the Ripper is uppercase, using the included tool *lm2ntcrack.pl* will quickly reveal the correct case for each letter. It is much smarter to first crack the LM hash and then use lm2ntcrack to reveal the password than trying to crack the NT hash. The set of possible passwords for NT are much larger than the set for LM. When the password is revealed from the LM hash the set of possible password permutations made of different letter cases are not very large. The *lm2ntcrack.pl* script will quickly reveal the correct password with for instance the command displayed in Listing 7.1:

```
sudo perl lm2ntcrack.pl -v -f="<JOHN-THE-RIPPER OUTPUT FILE>"
```

Listing 7.1: How to run lm2ntcrack.pl with a John the Ripper output file.

In category *Supplementary tools* we included a script called *mountos.sh*. This script helps the user to mount the correct file system, and as can be reviewed in Appendix B.1 (method *ismounted()*), this script is also used before some of the other scripts mentioned above are initiated. The *mountos.sh* can be found in Appendix B.4. In this category we also included the tool *TrueCrypt* [101], giving the user a possibility to encrypt/decrypt part of the host system. This tool can provide a protection against many of the tools and procedures presented in this thesis.

Chapter 8

Empirical Password Study

In this section we present an empirical study of the strength of different types of passwords using various tools. We chose to use the already described tools *Ophcrack*, *Cain & Abel* and *John the Ripper* for this experiment. The reason why we wanted to perform a password study was to find out how easily passwords can be cracked. We chose 30 passwords of different strengths and wanted to see how many of these that we were able to crack during a period of 1 working day (8 hours). In this experiment, dictionary attacks and cryptanalysis attacks, which are both password cracking techniques, were applied. Note that since the tools used in this experiment are described earlier in this thesis, a detailed description of how to use these tools will not be given here. This chapter will describe how the experiment was carried out, and also present the results we obtained. A discussion of our findings will be presented in Chapter 9.

This experiment is divided into two phases, *Phase 1* and *Phase 2*. Phase 1 is a time measurement phase with the use of only *one* strong password, while in Phase 2 all of the 30 selected passwords were used (Table 8.3). Phase 2 is the phase where the actual password cracking process was carried out. These two phases are described in detail below. We have included three different types of hash values in this experiment, LM hash, NT hash and MD5 hash, and the two phases are applied to each of the three types of hashes. Recall that LM hash is the type of hash used in for example Windows XP, NT hash is the more secure successor of the LM hash used in for example Windows Vista and Windows 7 RC, and the MD5 hash is used in many Unix systems.

8.1 Laboratory Environment

For this experiment, we used the same laboratory presented in Chapter 3 but we did not use all of the computers. We chose to use only two of them as we wanted to run the cracking processes in parallel but still have computers with the same hardware specifications. The reason for this was to be able to make some comparisons between the cracking processes for the different types of hashes. We chose to use Fredriksholm and Kristiansten because they were the two computers with the best, and identical, specifications. Frediksholm and Kristiansten originally had Windows XP and Ubuntu Server 8.10 installed respectively, as can be seen from Table 3.2 in Chapter 3, but instead Windows Vista and Ubuntu 9.04 were installed on these systems for this specific experiment. As mentioned, we have divided this experiment into two phases, and in Table 8.1 you can see which operating systems that were installed on Fredriksholm and Kristiansten in Phase 1 and Phase 2. Note that the whole disk on each of the two computers was dedicated to the operating system installed. This means that we did not use several partitions. In Phase 1 none of the time-measuring attacks were run in parallel, but in Phase 2 both the dictionary- and the cryptanalysis attacks on the LM- and NT hashes were run in parallel. This was done on the two Windows Vista systems. The dictionary attack that was run on the MD5 hashes in Phase 2 was not run in parallel with any of the other attacks. The MD5 dictionary attack was run on Ubuntu 9.04, which was installed on Kristiansten before we instead installed Windows Vista to run the attacks against the LM- and NT hashes in parallel. Note that we have included Ubuntu 9.04 for this experiment, as this version of Ubuntu was released just before we started the experiment.

As already mentioned, we use three types of hashes in this experiment. The reason why we chose LM is that it is still in use and that we wanted to demonstrate its weak properties, which is described in detail in Section 2.3.1.2. The rainbow tables used to crack the LM hash is much smaller than those for other hashes because these weak properties lead to less combinations to try. Oechslin [Oec03] also used the LM hash as an example when he introduced the concept of rainbow tables. We chose to include the NT hash because this is the new hash type included by Microsoft on newer Windows versions. The NT hash is supposed to solve many of the weaknesses of the LM hash, but still no salt is included. This is why the NT hash also is susceptible to the rainbow table attacks. The MD5 hash was chosen because it is a common type of hash used in Unix systems. Earlier in this thesis we identified that both Ubuntu Server 8.10 and FreeBSD 7.1 utilize the MD5 hash in their password files. MD5 hashes are supported by John the Ripper, which is the tool that we used for the MD5 hashes in this experiment, and they may be cracked with the use of a dictionary attack. At RainbowCrack's web pages [67] rainbow tables for MD5 are available but we did not include these in the experiment due to time constraints. Some of the operating systems described earlier, as for example Fedora 10, use the SHA-512 hash. The reason why we did not include SHA-512 was that the version of John the Ripper used in this experiment did not support it.

Computer name	**Operating system in Phase 1**	**Operating system in Phase 2**
Fredriksholm	Windows Vista	Windows Vista
Kristiansten	Ubuntu 9.04	Windows Vista, Ubuntu 9.04

Table 8.1: The operating systems used in Phase 1 and Phase 2.

8.2 Line of Action

In this section we will describe how this experiment was carried out. As already mentioned, we have chosen to divide the experiment into two phases, Phase 1 and Phase 2. Phase 1 is a timing phase and tends to say whether it is profitable to run a dictionary attack before we start to use rainbow tables to crack the passwords. Phase 2 is the phase where the actual cracking process was carried out. Be aware of that these two phases were carried out on each of the three different types of hashes chosen for this experiment (LM hash, NT hash and MD5 hash). Table 8.2 gives an overview of which techniques (dictionary attack and/or cryptanalysis attack) that were applied to what type of hash in the two phases. The techniques listed were used in both Phase 1 and Phase 2 for the corresponding hash type. For all the dictionary attacks performed we used a wordlist that is already included in the Cain & Abel tool. In the case where John the Ripper was used for the dictionary attack the same wordlist was imported. When it comes to the rainbow tables, which were only used for the LM and NT hashes, we used two different types of rainbow tables. For the LM hashes we used rainbow tables specifically designed to crack LM hashes and these tables, which in all consists of 64 subtables, were in total 64GB in size. It may be added that we installed an extra hard disk to be able to store these rainbow tables. The LM rainbow tables were downloaded for *free* from [95]. The LM hashes were imported into Cain & Abel, and for Cain & Abel to understand which characters these rainbow tables support we had to include a charset (character set). The charset used is named alpha-numeric-symbol32-space, can be found on RainbowCrack's web pages [102] and includes the following characters: [ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$%^&*()-_+=~‘[]{}|\:;”’< >,.?/ whitespace].

For the NT hashes we used rainbow tables that were specifically designed to crack NT hashes, but the tables we used for this purpose were not free. It is possible to obtain free NT rainbow tables from Ophcrack's web pages [44], such as the one named *Vista free*, but these do not include as many characters as the versions you have to pay for. And because we managed to get financial support for these tables we wanted to test them in this experiment. The NT rainbow tables that we bought are called *Vista special*, are 8GB in size and were obtained from Ophcrack's web page as well. An overview of the character sets that is included by the *Vista special* tables is given in Listing 8.1 [44]. Note that there are three different character sets, and which of them to consider depends on how many characters the password consist of. The *Vista special* tables are said to have a success rate of 99%, which means that they are supposed to reveal 99% of the passwords that consist of characters that are within the character set that is used.

```
Passwords of length 6 or less
  Charset: 0123456789
  abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
  !"#$%&'()*+,-./:;<=>?@["]^_`{|}~ (including the space character)
Passwords of length 7
  Charset: 0123456789
  abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
Passwords of length 8
  Charset: 0123456789abcdefghijklmnopqrstuvwxyz
```

Listing 8.1: The character sets used by the Vista special rainbow tables.

The *Vista special* rainbow tables in all consist of 4 rainbow tables. Since the Ophcrack tool manages to understand what characters the rainbow tables support we did not have to make an extra effort to include the charset for the NT rainbow tables. It should be noted that the rainbow tables used were downloaded before we started the experiment. Before we go into detail of the two phases we will mention which passwords we have chosen to use in this experiment, and explain the reason for our choice.

Type of hash	**Techniques**
LM hash	Dictionary attack and cryptanalysis attack
NT hash	Dictionary attack and cryptanalysis attack
MD5	Dictionary attack

Table 8.2: Applied techniques corresponding to their type of hash.

8.2.1 Selection of Passwords

In this experiment we wanted to test passwords of different strength. That is why we created three different password groups, and then added 10 passwords to each group. *Most Popular*, *Mnemonic Passwords* and *Random Passwords* are the names of these groups. The ten passwords added to the *Most Popular* group are taken from Bruce Schneier's web page [16], 8 of the *Mnemonic Passwords* are taken from a mnemonic password generator [103] while 2 are created by us. The *Random Passwords* are collected from a random password generator [104] found on the Internet. The group labeled *Most Popular* contains some of the most common passwords used, and as we can see from Table 8.3, these passwords do not seem very secure. As a contrast, the *Random Passwords* are considered very secure because they are randomly generated with both lowercase- and uppercase letters, numbers and special characters. One disadvantage with the randomly generated passwords is that they are hard to remember, and that is why we have included some mnemonic passwords. A mnemonic password is a password where a user for example chooses a phrase that is easy to remember (mnemonic) and uses a character to represent each word in the phrase. Mnemonic passwords are therefore passwords that seem like random generated passwords but are easier to remember because they are generated from phrases that are easy to remember. As already mentioned, we chose to create two of the mnemonic passwords ourselves (5GCTw4tG@N and tcC!tW84s), and this was to also include mnemonic passwords of nine and ten characters in length. It was desirable to test passwords with different characteristics. As an example of a mnemonic password we can take a look at one of our self-created password: *5GCTw4tG@N* - *Fifth Grade Comm Tech waiting for their Graduation at NTNU.* An overview of the 30 different passwords and to which group they belong is given in Table 8.3.

Most Popular	**Mnemonic Passwords**	**Random Passwords**
password1	hm71li	qaSt4@
abc123	k0fzug	vANe$a
myspace1	fp6jar	!aFu9ut
password	msi89a0	C7e++AV
blink182	%l41pc	w2U$atHe
qwerty1	m.f0vgk	$_Ch6cU5
fuckyou	v*qbt4un	Ke42A2Pe*
123abc	qtdra#	rA3$_ey*c
baseball1	5GCTw4tG@N	b*#D9*7yEG
football1	tcC!tW84s	8rA*_pHa$8

Table 8.3: An overview of the 30 selected passwords, and their corresponding category.

Table 8.4 below lists all the selected *Mnemonic Passwords* and their corresponding phrase.

Mnemonic Passwords	**Phrase**
hm71li	Hefin manages Sven's first lugubrious identity
k0fzug	Kath's nothing frazzles Zoe's unexpected gadget
fp6jar	Frank promotes Zach's jocular absentminded rabbit
msi89a0	Morgan seizes if eighth Nina argues nothings
%l41pc	Percy liberates Fourier's first priceless cabbage
m.f0vgk	Morgan stops for only Vivian glamorizes kilns
v*qbt4un	Victor's handy quarter boosts Tim's fourth unknown number
qtdra#	Quentin traps Dave's reputable aloof hash
5GCTw4tG@N	Fifth Grade Comm Tech waiting for their Graduation at NTNU
tcC!tW84s	The current crisis in the world fights for survival

Table 8.4: 10 mnemonic passwords with their corresponding phrase.

8.2.2 Phase 1: Timing

In Phase 1 the idea was to figure out how long time the dictionary attack and the cryptanalysis attack with the use of rainbow tables took on *one* strong password for each of the three different hashes. The password chosen for this purpose was *8rA*_pHa$8*. The SAM file with the LM hash value of the password was obtained from a Windows XP machine, the SAM file with the NT hash value of the password from a Windows Vista machine and the shadow file with the MD5 hash value of the password from an Ubuntu 9.04 machine. As the selected password did not exist in the wordlist used for the dictionary attack, the dictionary attack would complete without finding the password. This means that the whole wordlist and its possible character combinations, defined by the mangling rules that may be chosen in the tool that is used, will be traversed without a hit. This will therefore give us an indication of how long time the dictionary attack will take when it is not successful.

One of the ideas with Phase 1 was to find out of how much time we would have to invest in this worst case scenario. For the LM and NT hashes we thought it would be desirable to use a dictionary attack to reveal possible weak passwords before we start using rainbow tables in a cryptanalysis attack, and then reduce the running time for the rainbow tables. On the other hand we did not want to spend too much time running the dictionary attack as the idea with this experiment was to see how many of the 30 selected passwords we were able to crack during 1 working day (8 hours). Even though the dictionary attack was the only attack applied to the MD5 hashes, we wanted to get an idea of how long time we could expect this attack to take for all the passwords in Phase 2. For the cryptanalysis attacks we came up with an *estimate* of how long time the process would take. In contrast to the dictionary attacks we did not run the process to completion for the selected password used in this phase, but rather just ran the cryptanalysis attack with the use of one and two rainbow tables. For the LM hash we assumed that the password (8rA*_pHa$8) would not be revealed when using these two rainbow tables. If the password were revealed in any of these tables, other rainbow tables would have to be selected. Because of the limitations of the charset and the restrictions on the password length supported by the rainbow tables used for the NT hash, the password would not be found. Based on these results we calculated how long time it would take in total with the use of all the rainbow tables. These calculations will be explained in more detail. This was done for both the LM hash and the NT hash. The difference was that we used the tool named Cain & Abel for the LM hash and the tool named Ophcrack for the NT hash when using the rainbow tables. Table 8.5 gives an overview of which tools that were used for the different types of hashes. For the LM hash we used Cain & Abel for both the dictionary attacks and the cryptanalysis attacks, for the NT hash we used Cain & Abel for the dictionary attacks and Ophcrack for the cryptanalysis attacks, and for the MD5 hash we used John the Ripper for the dictionary attacks.

The point of Phase 1 was, for the LM and NT hash, to measure how long time both the dictionary attack and the cryptanalysis attack would take on the selected password, and then compare the results to indicate how much time the dictionary attack took

Type of hash	**Tools**
LM hash	Cain & Abel
NT hash	Cain & Abel, Ophcrack
MD5	John the Ripper

Table 8.5: An overview of the tools used in Phase 1 and Phase 2.

compared to when we used the rainbow tables. We wanted to find out how much time we had to use to be able to exclude the weakest passwords by using a dictionary attack before using the rainbow tables. For the MD5 hash we wanted to measure how long time the dictionary attack took on the selected password. It should also be added that we used the built-in timing function in both Cain & Abel and Ophcrack to measure the time for the cryptanalysis attacks on both the LM hash and the NT hash. John the Ripper also displays the elapsed time when you press *Enter* during the dictionary attack. The only case where the running time was not displayed by the tool used, was when the dictionary attack on the LM hash and the NT hash was run with Cain & Abel. Then we had to use the *CPU Time*, which is displayed in the *Windows Task Manager*, for the specific process. We made sure that no other unnecessary processes were running at the same time as the cracking processes to make the processor use all its available resources on this task. It should be noted that this timing function may not be completely precise as other necessary processes will probably influence the counting when these processes need the processor. The time measurements are used to just give an *indication* of how long time the attacks may take. It should be added that we also used an external timing function to be able to check manually if the CPU timing result was reasonable. This was done for each of the hash types.

8.2.3 Phase 2: Password Cracking

Phase 2 is the actual password cracking phase where we first, for the LM and NT hashes, applied a dictionary attack to hopefully exclude some of the weakest passwords before we started to crack the rest of the passwords with the use of rainbow tables. For the MD5 hashes we only applied a dictionary attack. It should be noted that in Phase 1 we only concentrated on *one* password but in Phase 2 we attacked all the 30 selected passwords given in Table 8.3. We created 30 users on a Windows XP machine, 30 users on a Windows Vista machine and 30 users on an Ubuntu 9.04 machine to be able to attack all of the three different types of hashes. It is also worth mentioning that we also created 30 users on a Windows 7 RC (Release Candidate) machine, which also utilizes the NT hash, as an experiment on the side. This was because Windows 7 RC was released during our experiments, and we wanted to find out if Windows had changed the way it handles the NT hash values stored in a Windows 7 RC SAM file. Possible differences between the NT hashes generated on the Windows Vista and the Windows 7 RC machine will be highlighted. The user accounts, and thus also the corresponding password files, were created on virtual machines using a program called VirtualBox [78].

We used VirtualBox because the actions on a virtual machine are easy to revert. To create this many users we made scripts to automate this process, and these scripts can be found in Appendix D. The script for adding users in a Unix system adds 30 users with their passwords hashed with MD5. The result was a shadow file created in Ubuntu 9.04 containing 30 MD5 hash values. The MD5 algorithm is utilized for the user that is added when installing Ubuntu 8.10.

As already mentioned, the cracking processes was done on different computers than the target machines, and instead the password files were imported to the machine running the cracking process. When we planned this experiment, and before we knew how long time the dictionary and the cryptanalysis attack would take, we agreed to stop the dictionary attacks on the LM hashes and the NT hashes after maximum 2 hours and then use the remaining 6 hours for the rainbow tables (8 hours in total). The reason why we planned with such time constraints was that we did not expect all the passwords to be revealed and that we therefore had to cancel the jobs before completion, as the idea was to only use *1 working day*. We also wanted the cryptanalysis attack to get most of the available time as we thought that the chances were bigger that the cryptanalysis attack would crack the more difficult passwords. When it comes to the MD5 hashes the dictionary attack was supposed to run for 8 hours continuously, and then be aborted if it had not completed by then. Note that even though the case with the Windows 7 RC machine was an experiment on the side, we also performed a full dictionary attack and a full cryptanalysis attack on the SAM file that originated from the Windows 7 RC machine.

8.3 Results of The Study

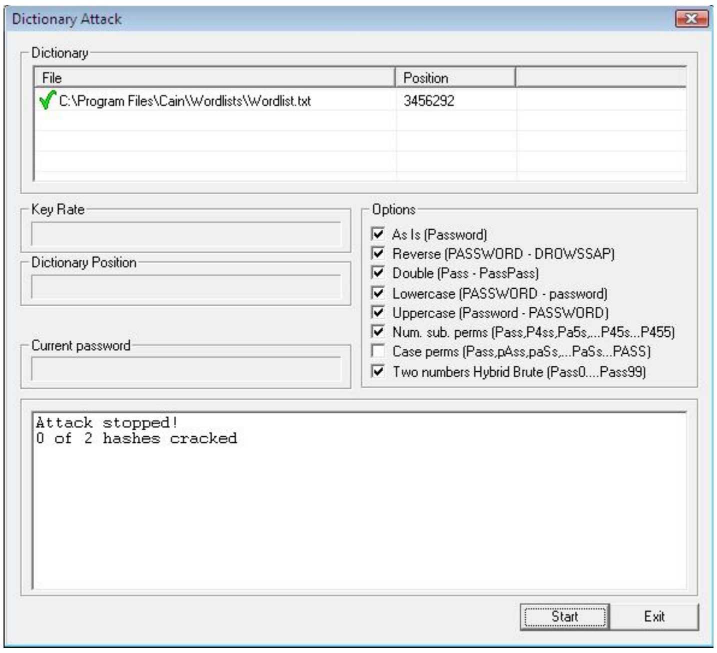
In this section the achieved results regarding the empirical password study will be presented. The results presented here will be further discussed in Section 9.2 in Chapter 9. Keep in mind that this experiment was supposed to reveal how many of the 30 selected passwords that were possible to disclose during 1 working day (8 hours).

8.3.1 Phase 1: Timing

Phase 1 of this experiment was a timing phase. We used this phase to measure how long time the dictionary attacks and the cryptanalysis attacks would take when trying to crack the LM hash and NT hash of the selected password. This phase also measured the time of the dictionary attack on the MD5 hash. The results from this timing phase are presented below.

8.3.1.1 LM hash

First the dictionary attack on the LM hash will be considered. As can be seen in the upper part of the window shown in Figure 8.1 we used a wordlist (Wordlist.txt) for the dictionary attack. This wordlist is included by default in the Cain & Abel tool. The screen shot was taken after the dictionary attack on the selected password had finished, and as can be seen from the figure the password was not revealed. The figure also shows which options, or mangling rules, that could be selected for the dictionary attack. We used the default options shown in the figure. Also note the line saying *0 of 2 hashes cracked* in the lower half of the window. Even though we only used one password, Cain & Abel says that it has found 2 hash values. This is, as described in detail in Section 2.3.1.2, a consequence of the properties of the construction of the LM hash. When the LM hash is constructed, the password is first divided into two segments with maximum 7 characters in each segment. The two segments are then hashed separately. The password used in this phase consists of 10 characters and therefore 2 LM hash values are constructed, one for the first 7 characters and one for the remaining 3 characters. As expected, none of these two hash values was cracked after running the dictionary attack, and thus the password was not revealed.


Dictionary Attack

Dictionary

File	Position
C:\Program Files\Cain\Wordlists\Wordlist.txt	3456292

Key Rate

Dictionary Position

Current password

Options
- [✓] As Is (Password)
- [✓] Reverse (PASSWORD - DROWSSAP)
- [✓] Double (Pass - PassPass)
- [✓] Lowercase (PASSWORD - password)
- [✓] Uppercase (Password - PASSWORD)
- [✓] Num. sub. perms (Pass,P4ss,Pa5s,...P45s...P455)
- [ ] Case perms (Pass,pAss,paSs,...PaSs...PASS)
- [✓] Two numbers Hybrid Brute (Pass0....Pass99)

```
Attack stopped!
0 of 2 hashes cracked
```

Figure 8.1: Phase 1: Settings used for the dictionary attack (LM hash).

For the dictionary attack we had to open the *Windows Task Manager* and use the *CPU Time* for the dictionary attack to measure the time. As already mentioned, this is because the Cain & Abel tool does not show the elapsed time for the dictionary attack. If we take a look at the *CPU Time* column for the Cain.exe process in Figure 8.2, it says that the dictionary attack took 1 minute and 14 seconds. This is not entirely true as the time starts to run when you start the Cain & Abel program and not from when you start the dictionary attack. This is why we deduct 6 seconds from this time, as this was the time Cain & Abel used to start up, making the total time for the dictionary attack *1 minute and 8 seconds* (68 seconds). This time measurements show how long time it may take if the password is not found during the dictionary attack.

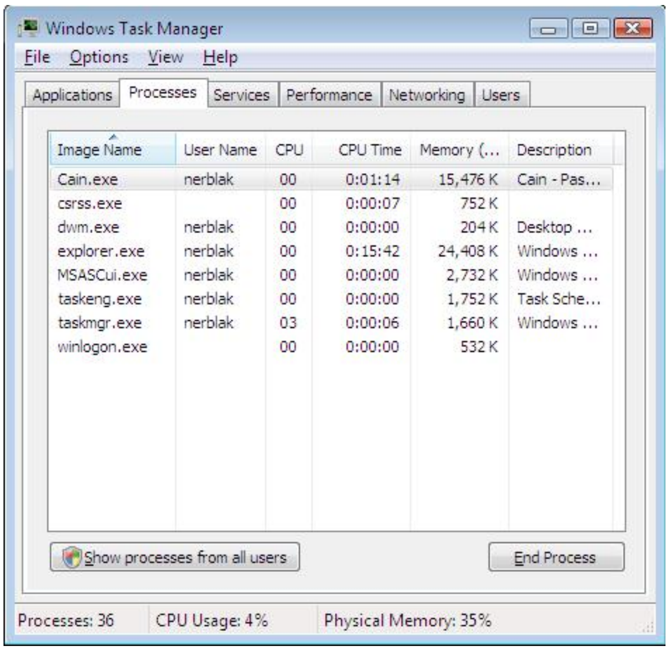


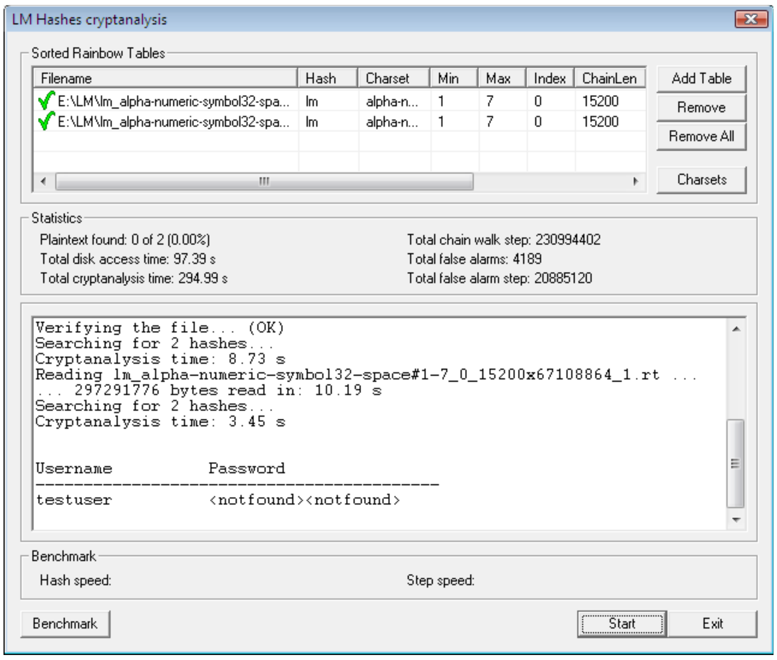
Image Name	User Name	CPU	CPU Time	Memory (...	Description
Cain.exe	nerblak	00	0:01:14	15,476 K	Cain - Pas...
csrss.exe		00	0:00:07	752 K	
dwm.exe	nerblak	00	0:00:00	204 K	Desktop ...
explorer.exe	nerblak	00	0:15:42	24,408 K	Windows ...
MSASCui.exe	nerblak	00	0:00:00	2,732 K	Windows ...
taskeng.exe	nerblak	00	0:00:00	1,752 K	Task Sche...
taskmgr.exe	nerblak	03	0:00:06	1,660 K	Windows ...
winlogon.exe		00	0:00:00	532 K	

Processes: 36 CPU Usage: 4% Physical Memory: 35%

Figure 8.2: Phase 1: Timing of the dictionary attack on the LM hash.

Now let us take a look at the timing of the *cryptanalysis attack* on the LM hash. Figure 8.3 shows the result of the cryptanalysis attack on the selected password for the created *testuser* account, using the first two tables of the downloaded rainbow tables. We used only two tables to be able to create an estimate instead of running the whole attack. It should be added that we in addition ran the cryptanalysis attack using just one rainbow table. We had to run both to be able to estimate how long time the cryptanalysis attack would take when using all the rainbow tables. The reason why we had to use both the first and the second rainbow table was because it takes longer time to run the first rainbow table than the following tables. Rainbow chains and all its intermediate values are calculated for the first table, and then reused for the succeeding tables. The use of the first rainbow table may be considered as an initial process. That is why we needed to consider the second table, and then multiply the time it took to run this table with the total amount of tables minus the initial table. The result is the time it would take to run the last 63 rainbow tables. Then we had to add the time it took to complete the first table (initial process).

As can be seen from Figure 8.3, using the first two tables took 294.99 + 97.39 = 392.38 seconds = 6 minutes 32 seconds when the password was not revealed. These numbers can be found in the middle section named *Statistics* in Figure 8.3. When calculating the total time we included *Total disk access time* and *Total cryptanalysis time*. As already mentioned, we also measured the time when only the first rainbow table was used to be able to estimate the total time. It took 282.81 + 56.83 = 339.64 seconds = 5 minutes 39 seconds to complete the first rainbow table. This means that it took only 392.38 - 339.64 = 52.74 seconds to run the second table. Now it is time to make an estimate for the maximum time the cryptanalysis attack might take when including all the 64 rainbow tables: 339.64 seconds + (52.74 seconds * 63 tables) = *1 hour 1 minute 2 seconds* (3662.26 seconds).



Filename	Hash	Charset	Min	Max	Index	ChainLen
✓ E:\LM\lm_alpha-numeric-symbol32-spa...	lm	alpha-n...	1	7	0	15200
✓ E:\LM\lm_alpha-numeric-symbol32-spa...	lm	alpha-n...	1	7	0	15200

Plaintext found: 0 of 2 (0.00%)
Total disk access time: 97.39 s
Total cryptanalysis time: 294.99 s
Total chain walk step: 230994402
Total false alarms: 4189
Total false alarm step: 20885120

```
Verifying the file... (OK)
Searching for 2 hashes...
Cryptanalysis time: 8.73 s
Reading lm_alpha-numeric-symbol32-space#1-7_0_15200x67108864_1.rt ...
... 297291776 bytes read in: 10.19 s
Searching for 2 hashes...
Cryptanalysis time: 3.45 s


Username          Password
------------------------------------------
testuser          <notfound><notfound>
```

Figure 8.3: Phase 1: Timing of the cryptanalysis attack on the LM hash.

Table 8.6 gives an overview of the timing results from the dictionary attack and from the estimate of the cryptanalysis attack. The latter is based on the accomplishment of two of the 64 rainbow tables. Both these attacks were performed on one password.

Technique	**Time**
Dictionary attack	1 minute 8 seconds (68 seconds)
Cryptanalysis attack	1 hour 1 minute 2 seconds (3662.26 seconds)
Total	1 hour 2 minutes 10 seconds (3730 seconds)

Table 8.6: Phase 1: Time measurements for the LM hash.

The idea with the timing phase was to find out how long time the dictionary attack and the cryptanalysis attack might take, and if it would be advantageous to include a dictionary attack to reveal the easiest passwords before running the cryptanalysis attack with the rainbow tables. Table 8.6 gives an indication of how much time the different attacks might take, and Figure 8.4 gives an overview of how much time this constitute of the 8 hours period available. The dictionary attack is labeled *Dictionary attack* and the cryptanalysis attack where the rainbow tables were used is labeled *Cryptanalysis attack*. The unused time is named *Remaining time*. As can be seen from the figure, the dictionary attack only constitutes 0.24% of the available time and the cryptanalysis attack is estimated to constitute 12.72% of the time. This leave us with as much as 87.04% unused time.

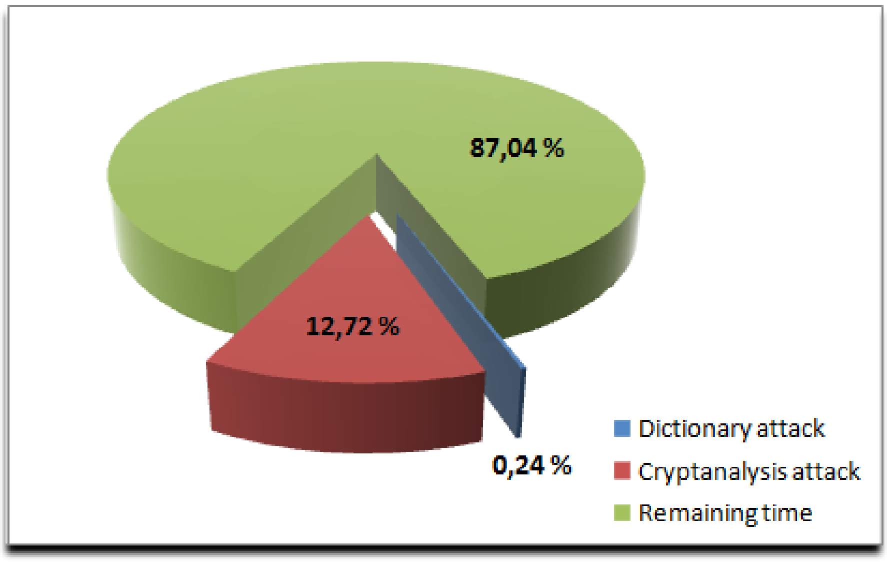
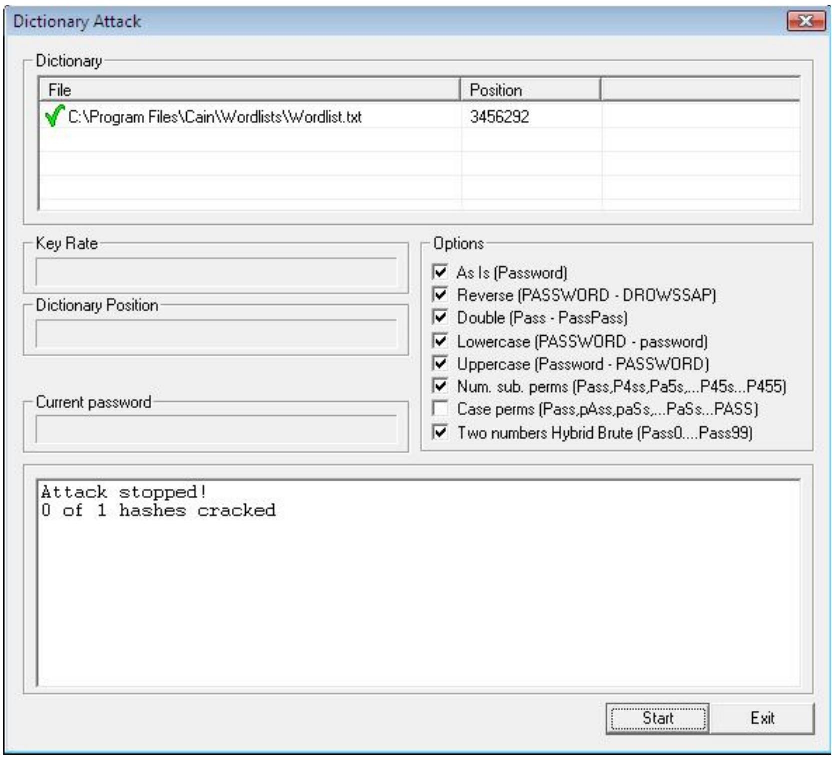


Figure 8.4: Distribution of time between the techniques applied to the LM hash.

8.3.1.2 NT hash

The screen shot in Figure 8.5, which is almost identical as the screen shot shown in Figure 8.1 for the LM hash, shows the default settings, or mangling rules, that were used for the dictionary attack on the NT hash. It can also be verified that we used the same wordlist (Wordlist.txt) as for the LM hash. The only difference between those figures is that only *one* hash value is found by Cain & Abel when cracking an NT hash in contrast to the two hash values found when cracking the LM hash. In Figure 8.5 this can be verified from the line saying *0 out of 1 hashes cracked*. The reason is that when the NT hash value is constructed, the password used is not divided into two segments before the hash is generated, as is the case when generating the LM hash. As a result of this, the password has only one corresponding hash value.

For the NT hash we also had to open the *Windows Task Manager* and use the *CPU Time* to measure the time for the dictionary attack, due to the lack of timing capabilities in Cain & Abel. The dictionary attack was measured to take 54 seconds, but from this we had to deduct 5 seconds as this was the time it took to start Cain & Abel. The reason why had to deduct less time when considering the NT hash compared to the LM hash, was because the processor took longer time to start Cain & Abel in the first case. This makes the total time for the dictionary attack *49 seconds*. This gives an indication of how long time it might take if the password is not found during the dictionary attack.



C:\Program Files\Cain\Wordlists\Wordlist.txt 3456292

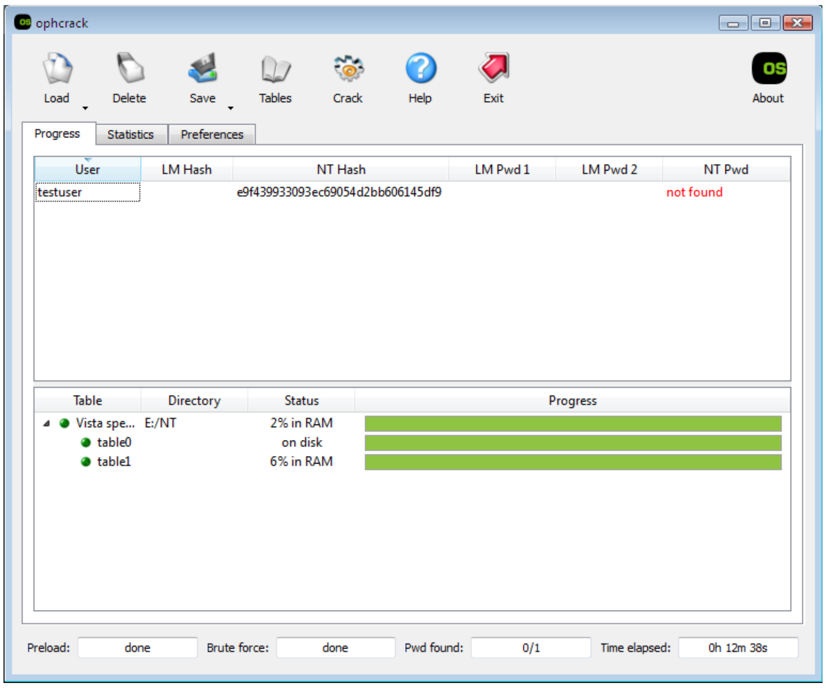
- [✓] As Is (Password)
- [✓] Reverse (PASSWORD - DROWSSAP)
- [✓] Double (Pass - PassPass)
- [✓] Lowercase (PASSWORD - password)
- [✓] Uppercase (Password - PASSWORD)
- [✓] Num. sub. perms (Pass,P4ss,Pa5s,...P45s...P455)
- [ ] Case perms (Pass,pAss,paSs,...PaSs...PASS)
- [✓] Two numbers Hybrid Brute (Pass0....Pass99)

```
Attack stopped!
0 of 1 hashes cracked
```

Figure 8.5: Phase 1: Settings used for the dictionary attack (NT hash).

Now let us consider the timing of the *cryptanalysis attack* for the NT hash. Figure 8.6 shows the result of the cryptanalysis attack on the selected password for the user called *testuser*. As for the LM hash, only two of the rainbow tables that normally belong together were used in this phase to get an estimate of how long time the whole cryptanalysis attack might take when the password is not found. Also in this case we additionally ran the cryptanalysis attack using just one rainbow table to be able to estimate the time it would take when using all the rainbow tables. To estimate the total time we need to multiply the time it took to run the second table with the total amount of tables minus the first table (63 tables all together). The time it takes for the first rainbow table is then added.

As can be seen from the bottom right corner of Figure 8.6 the cryptanalysis attack on the NT hash, using two of the tables, took *12 minutes 38 seconds* (758 seconds). The field displaying the time is labeled *Time elapsed*. Now we have to make an estimate of how long time it might take when all the 4 rainbow tables are included. The 4 NT rainbow tables mentioned here are the tables that together form the non-free *Vista special* tables. It should be added that it took 7 minutes and 38 seconds (458 seconds) when we only used the first rainbow table. This means that it took 758 - 458 = 300 seconds = 5 minutes to run the second rainbow table. The total estimate for the use of rainbow tables on the NT hash is then: 458 seconds + (300 seconds * 3 tables) = *22 minutes 38 seconds* (1358 seconds). This illustrates the maximum time the cryptanalysis attack



User	LM Hash	NT Hash	LM Pwd 1	LM Pwd 2	NT Pwd
testuser		e9f439933093ec69054d2bb606145df9			not found

Table	Directory	Status	Progress
Vista spe...	E:/NT	2% in RAM	
table0		on disk	
table1		6% in RAM	

Preload: done Brute force: done Pwd found: 0/1 Time elapsed: 0h 12m 38s

Figure 8.6: Timing of the cryptanalysis attack on the NT hash.

might take when including all the 4 rainbow tables. What can also be seen from Figure 8.6 is that one NT hash value has been loaded, and is listed in the *NT Hash* column. The corresponding password has, as expected, not been revealed as it says *not found* in the *NT Pwd* column or *0/1* in the *Pwd found* field at the bottom menu of the window. From the *Table* column you can also see that both table0 and table1, which are both part of the *Vista special* rainbow tables, were loaded.

Table 8.7 gives an overview of the timing results from the dictionary attack and from the estimate of the cryptanalysis attack performed on the NT hash value. Both these attacks were performed on one password.

Technique	**Time**
Dictionary attack	49 seconds
Cryptanalysis attack	22 minutes 38 seconds (1358 seconds)
Total	23 minutes 27 seconds (1407 seconds)

Table 8.7: Phase 1: Time measurements for the NT hash.

Figure 8.7 gives, based on the result given in Table 8.7, an overview of how much these results constitute of the 8 hours period available. The dictionary attack constitute only 0.17% of the available time and the cryptanalysis attack constitute 4.72%. As much as 95.11% of the available time is unused.

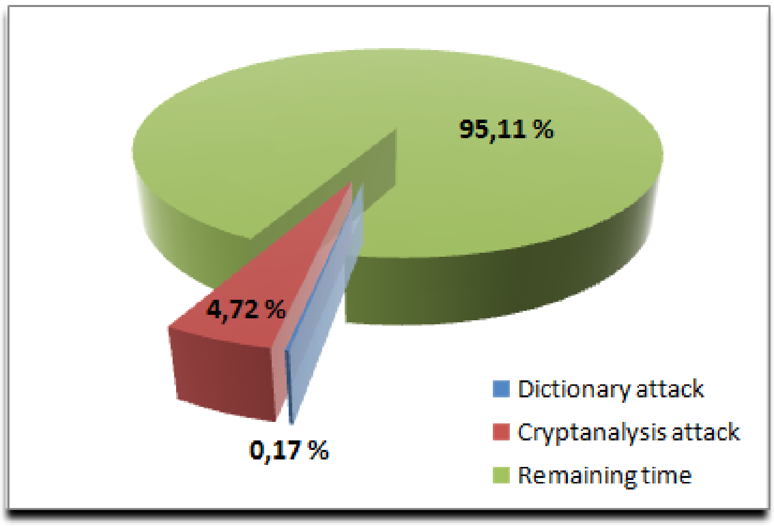


Figure 8.7: Distribution of time between the techniques applied to the NT hash.

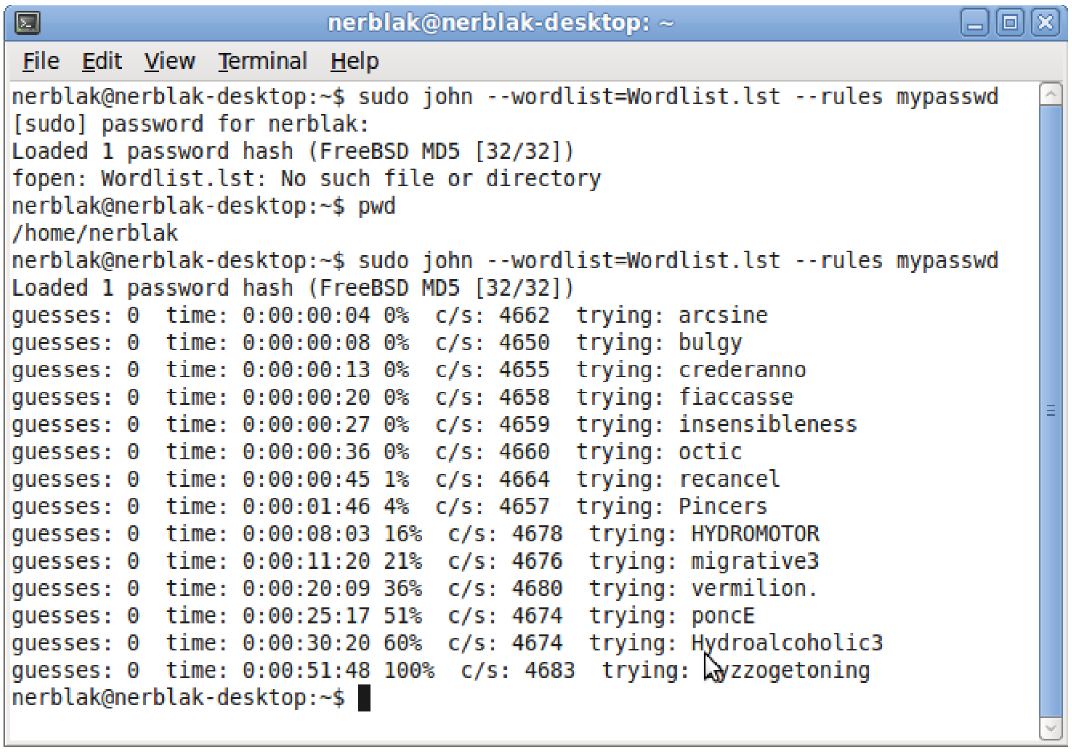
8.3.1.3 MD5 hash

For the MD5 hash we only performed a dictionary attack, and we used John the Ripper to find out how long time the dictionary attack would take on the selected password (8rA*_pHa$8). To start the dictionary attack we had to run a couple of commands:

1. umask 077
2. sudo unshadow /etc/passwd /etc/shadow > mypasswd
3. sudo john −−wordlist=Wordlist.lst −−rules mypasswd

First, the *umask* command was used to set adequate permission rights. Next we used the *unshadow* command to obtain the content of the passwd file and the shadow file, and then store the content in a new file named *mypasswd* in this case. The third command, *john*, was used to actually start the dictionary attack using John the Ripper. As can be seen from the third command we used the same wordlist as for both the LM hash and the NT hash. By using −−*rules* in the same command we also specified that we wanted to use certain mangling rules for the dictionary attack. Cain & Abel, which was used for the dictionary attack on both the LM and NT hash, also used certain mangling rules. The *sudo* command was used to get administrator privileges. Note that these commands also have to be applied in Phase 2 to run the dictionary attack on the MD5 hashes of all the 30 passwords.

The result from the dictionary attack is shown in Figure 8.8. We used the timing function included in the John the Ripper tool, and the figure shows that the dictionary attack, including the word mangling, took *51 minutes and 48 seconds* (3108 seconds) to complete on the selected password. Note the line in Figure 8.8 that, just before the password guessing begins, displays *Loaded 1 password hash*. This confirms that we only used one password, as there are not any division of the password before the hash is generated when using MD5. As expected, even though the dictionary completed the password was not revealed.



```
nerblak@nerblak-desktop:~$ sudo john --wordlist=Wordlist.lst --rules mypasswd
[sudo] password for nerblak:
Loaded 1 password hash (FreeBSD MD5 [32/32])
fopen: Wordlist.lst: No such file or directory
nerblak@nerblak-desktop:~$ pwd
/home/nerblak
nerblak@nerblak-desktop:~$ sudo john --wordlist=Wordlist.lst --rules mypasswd
Loaded 1 password hash (FreeBSD MD5 [32/32])
guesses: 0  time: 0:00:00:04 0%  c/s: 4662  trying: arcsine
guesses: 0  time: 0:00:00:08 0%  c/s: 4650  trying: bulgy
guesses: 0  time: 0:00:00:13 0%  c/s: 4655  trying: crederanno
guesses: 0  time: 0:00:00:20 0%  c/s: 4658  trying: fiaccasse
guesses: 0  time: 0:00:00:27 0%  c/s: 4659  trying: insensibleness
guesses: 0  time: 0:00:00:36 0%  c/s: 4660  trying: octic
guesses: 0  time: 0:00:00:45 1%  c/s: 4664  trying: recancel
guesses: 0  time: 0:00:01:46 4%  c/s: 4657  trying: Pincers
guesses: 0  time: 0:00:08:03 16%  c/s: 4678  trying: HYDROMOTOR
guesses: 0  time: 0:00:11:20 21%  c/s: 4676  trying: migrative3
guesses: 0  time: 0:00:20:09 36%  c/s: 4680  trying: vermilion.
guesses: 0  time: 0:00:25:17 51%  c/s: 4674  trying: poncE
guesses: 0  time: 0:00:30:20 60%  c/s: 4674  trying: Hydroalcoholic3
guesses: 0  time: 0:00:51:48 100%  c/s: 4683  trying: yzzogetoning
nerblak@nerblak-desktop:~$
```

Figure 8.8: Result from the dictionary attack on the MD5 hash.

Table 8.8 gives an overview of the timing results from the dictionary attack performed on the MD5 hash value.

Technique	**Time**
Dictionary attack	51 minutes 48 seconds (3108 seconds)

Table 8.8: Time measurement for the MD5 hash.

Based on the result given in Table 8.8, Figure 8.9 gives an overview of how much this result would constitute of the 8 hours period available. The dictionary constitutes 10.79% of the available time and as much as 89.21% of the available time is unused.

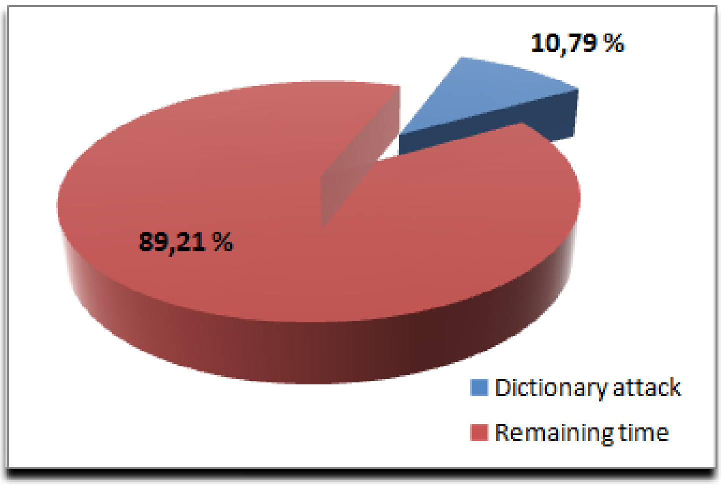


Figure 8.9: Distribution of used time for the dictionary attack on the MD5 hash.

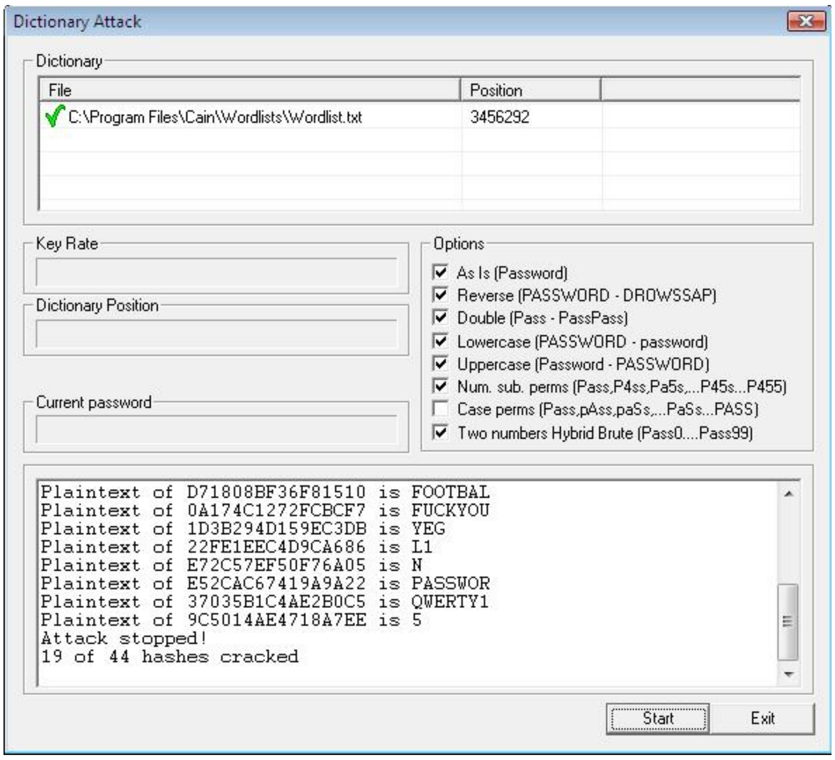
8.3.2 Phase 2: Password Cracking

Phase 2 is where the actual cracking process was carried out. We added 30 users with different passwords and wanted to examine how many of the 30 passwords we managed to crack during a period of 1 working day (8 hours). This was done for each of the three different types of hashes (LM hash, NT hash, and MD5 hash). Recall that we performed an experiment on the side for the NT hashes, with the use of a machine running the newly released Windows 7 RC operating system. This was to be able to highlight possible improvements that Microsoft had done regarding the handling of the login password. A full outline of the selected passwords can either be found in Table 8.3 or from the results presented below. After the completion of Phase 2 it would be possible to compare Phase 1 and Phase 2 to see how long time the cracking process would take for all the 30 passwords compared to when just one password was used, which is an important difference between Phase 1 and Phase 2. Another difference is that the full set of rainbow tables were used for both the LM hash and the NT hash in Phase 2.

8.3.2.1 LM hash

As for the LM hash in Phase 1, we also applied a dictionary attack and a cryptanalysis attack with rainbow tables in this phase. The only difference is that this time the password file contained 30 LM hash values instead of one LM hash value.

In Figure 8.10 you can verify that the same wordlist (Wordlist.txt) was also used in Phase 2, and that also the same options, or mangling rules, used in Phase 1 are applied. The screen shot is taken after the dictionary has completed, and in the bottom of the window it says *19 of 44 hashes cracked*. This does *not* mean that 19 passwords are revealed. Recall that the password is split into two segments when the LM hash is generated. It could mean that the whole password is found but it is also possible that only one of the two possible hash values that constitute the password is cracked. The latter case will reveal only parts of the password. Note that a password consisting of 7 characters or less will only have one LM hash value.



File	Position
✓ C:\Program Files\Cain\Wordlists\Wordlist.txt	3456292

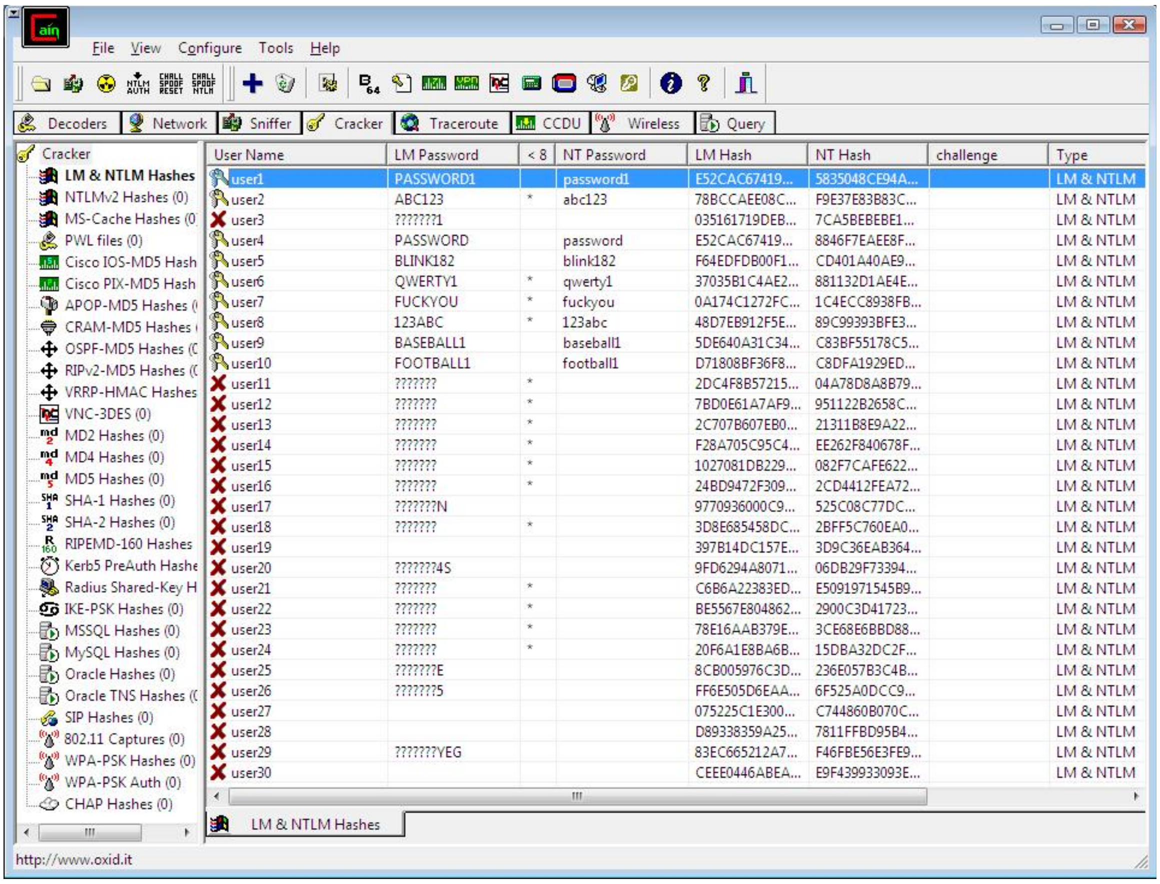
- [✓] As Is (Password)
- [✓] Reverse (PASSWORD - DROWSSAP)
- [✓] Double (Pass - PassPass)
- [✓] Lowercase (PASSWORD - password)
- [✓] Uppercase (Password - PASSWORD)
- [✓] Num. sub. perms (Pass,P4ss,Pa5s,...P45s...P455)
- [ ] Case perms (Pass,pAss,paSs,...PaSs...PASS)
- [✓] Two numbers Hybrid Brute (Pass0....Pass99)

```
Plaintext of D71808BF36F81510 is FOOTBAL
Plaintext of 0A174C1272FCBCF7 is FUCKYOU
Plaintext of 1D3B294D159EC3DB is YEG
Plaintext of 22FE1EEC4D9CA686 is L1
Plaintext of E72C57EF50F76A05 is N
Plaintext of E52CAC67419A9A22 is PASSWOR
Plaintext of 37035B1C4AE2B0C5 is QWERTY1
Plaintext of 9C5014AE4718A7EE is 5
Attack stopped!
19 of 44 hashes cracked
```

Figure 8.10: Phase 2: Settings used for the dictionary attack (LM hash).

Even though some of the revealed passwords can be seen in Figure 8.10, a complete list of all the 30 users and their possible revealed password are listed in Figure 8.11. From the overview is it also possible to see what type of hash that is used. This can be seen from which column the hash value itself is listed in. The columns are labeled *LM Hash* and *NT hash* in the top menu. Since the password file used at this point originate from a Windows XP machine, both the LM hash and NT hash values are listed. Also note the column labeled *<8*, where the passwords that contain less than 8 characters are marked.

Figure 8.11 shows that *9 passwords* were revealed from the dictionary attack. The entry with the revealed passwords is marked with a key symbol in the beginning of the column labeled *User Name*, while the ones that are not cracked are marked with a red cross. The 9 passwords that were revealed during this dictionary attack are listed in Table 8.9, and how much these passwords constitute of all the selected passwords are illustrated in Figure 8.12. Note that the numbering used in Table 8.9 are not universal for this experiment, and are just included to point out that there are 9 revealed passwords at this point. If you take a closer look at Figure 8.11 you will see that some of the passwords are partly cracked, but since not the whole password is revealed they could obviously not be included in the list of revealed passwords. Another observation is that some of



User Name	LM Password	< 8	NT Password	LM Hash	NT Hash	challenge	Type
user1	PASSWORD1		password1	E52CAC67419...	5835048CE94A...		LM & NTLM
user2	ABC123	*	abc123	78BCCAEE08C...	F9E37E83B83C...		LM & NTLM
user3	???????1			035161719DEB...	7CA5BEBEBE1...		LM & NTLM
user4	PASSWORD		password	E52CAC67419...	8846F7EAEE8F...		LM & NTLM
user5	BLINK182		blink182	F64EDFDB00F1...	CD401A40AE9...		LM & NTLM
user6	QWERTY1	*	qwerty1	37035B1C4AE2...	881132D1AE4E...		LM & NTLM
user7	FUCKYOU	*	fuckyou	0A174C1272FC...	1C4ECC8938FB...		LM & NTLM
user8	123ABC	*	123abc	48D7EB912F5E...	89C99393BFE3...		LM & NTLM
user9	BASEBALL1		baseball1	5DE640A31C34...	C83BF55178C5...		LM & NTLM
user10	FOOTBALL1		football1	D71808BF36F8...	C8DFA1929ED...		LM & NTLM
user11	???????	*		2DC4F8B57215...	04A78D8A8B79...		LM & NTLM
user12	???????	*		7BD0E61A7AF9...	951122B2658C...		LM & NTLM
user13	???????	*		2C707B607EB0...	21311B8E9A22...		LM & NTLM
user14	???????	*		F28A705C95C4...	EE262F840678F...		LM & NTLM
user15	???????	*		1027081DB229...	082F7CAFE622...		LM & NTLM
user16	???????	*		24BD9472F309...	2CD4412FEA72...		LM & NTLM
user17	???????N			9770936000C9...	525C08C77DC...		LM & NTLM
user18	???????	*		3D8E685458DC...	2BFF5C760EA0...		LM & NTLM
user19				397B14DC157E...	3D9C36EAB364...		LM & NTLM
user20	???????4S			9FD6294A8071...	06DB29F73394...		LM & NTLM
user21	???????	*		C6B6A22383ED...	E5091971545B9...		LM & NTLM
user22	???????	*		BE5567E804862...	2900C3D41723...		LM & NTLM
user23	???????	*		78E16AAB379E...	3CE68E6BBD88...		LM & NTLM
user24	???????	*		20F6A1E8BA6B...	15DBA32DC2F...		LM & NTLM
user25	???????E			8CB005976C3D...	236E057B3C4B...		LM & NTLM
user26	???????5			FF6E505D6EAA...	6F525A0DCC9...		LM & NTLM
user27				075225C1E300...	C744860B070C...		LM & NTLM
user28				D89338359A25...	7811FFBD95B4...		LM & NTLM
user29	???????YEG			83EC665212A7...	F46FBE56E3FE9...		LM & NTLM
user30				CEEE0446ABEA...	E9F439933093E...		LM & NTLM

Figure 8.11: Passwords revealed from the dictionary attack (LM hash).

the users listed in Figure 8.11 have question marks written in their corresponding *LM Password* field, and some do not. The users without question marks have passwords that are longer than 7 characters, meaning that their password consist of two hash values, and the dictionary attack has not been able to find any of the hash values. The users with question marks written in their password field are either less than 8 characters long, meaning that the password consist of only one hash value, or that the password is longer than 7 characters and that one of the hash values is found. The passwords that are not revealed or just partly revealed will be a subject to the cryptanalysis attack applied next. As Figure 8.12 illustrates, 30% of the passwords were revealed from the dictionary attack, while 70% of them will be a subject for the cryptanalysis attack.

Number	User name	Password
1	user1	password1
2	user2	abc123
3	user4	password
4	user5	blink182
5	user6	qwerty1
6	user7	fuckyou
7	user8	123abc
8	user9	baseball1
9	user10	football1

Table 8.9: Revealed passwords.

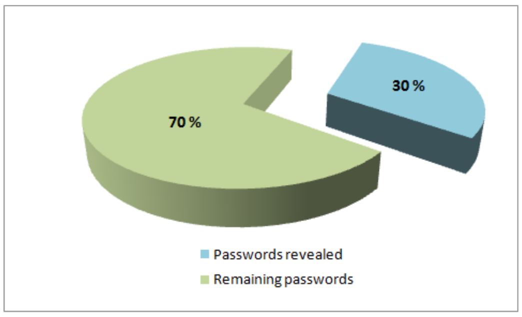
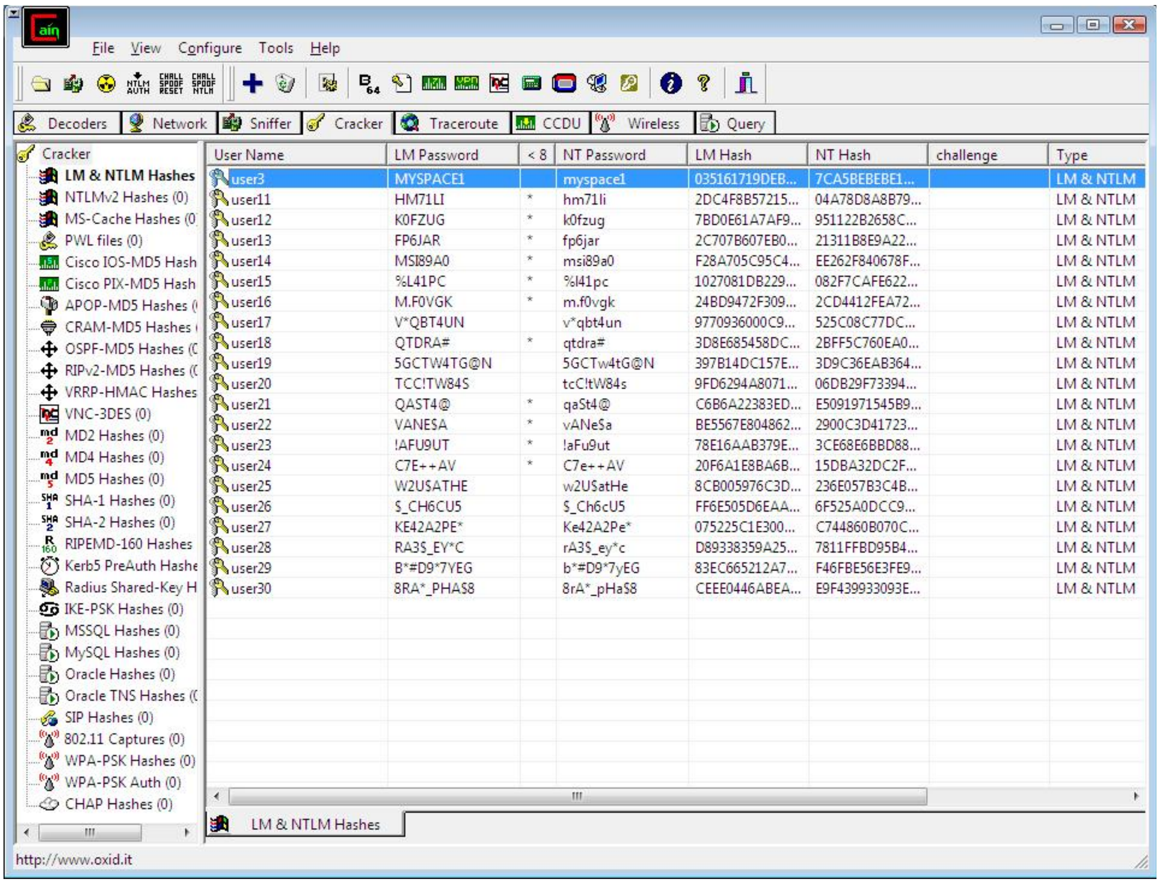


Figure 8.12: Share of revealed passwords.

In Phase 1 we estimated the time for the dictionary attack on the selected password to be 1 minute and 8 seconds. The dictionary attack carried out in this phase with all the 30 selected passwords was measured to take 1 minute and 9 seconds. But also in this case we had to deduct the time it took for Cain & Abel to start as we used the *CPU Time* displayed in *Windows Task Manager*. After deducting the 6 seconds that Cain & Abel used to start, the dictionary attack was estimated to take *1 minute 3 seconds*. This shows that it does not take longer time to run a dictionary attack on a password file containing several LM hashes than a password file containing just one LM hash. In fact, the estimate indicates that the case with several LM hashes takes shorter time than cracking just one LM hash, but be aware of that minor inaccuracy with the timing measurements may influence the result.

Now let us consider the *cryptanalysis attack* with the use of the LM rainbow tables. It should be noted that the users with a password that was revealed during the dictionary attack were removed from the list of passwords we wanted to reveal during the cryptanalysis attack. This can be verified in Figure 8.13 were all the users listed in Table 8.9 are omitted. As Figure 8.13 gives the result of the cryptanalysis attack, we can see that all the remaining *21 passwords* were revealed.

As both the dictionary attack and the cryptanalysis attack now are presented it is time to sum up the results from the cracking process on the LM hashes. All the 21 passwords that were revealed during the cryptanalysis attack can be found in Table 8.11. The 9 passwords that were revealed during the dictionary attack are also included in this table but they have *italic* font. The table also includes the user name, length of the password, to which category the password belongs, and whether the password was revealed or not.



User Name	LM Password	< 8	NT Password	LM Hash	NT Hash	challenge	Type
user3	MYSPACE1		myspace1	035161719DEB...	7CA5BEBEBE1...		LM & NTLM
user11	HM71LI	*	hm71li	2DC4F8B57215...	04A78D8A8B79...		LM & NTLM
user12	K0FZUG	*	k0fzug	7BD0E61A7AF9...	951122B2658C...		LM & NTLM
user13	FP6JAR	*	fp6jar	2C707B607EB0...	21311B8E9A22...		LM & NTLM
user14	MSI89A0	*	msi89a0	F28A705C95C4...	EE262F840678F...		LM & NTLM
user15	%L41PC	*	%l41pc	1027081DB229...	082F7CAFE622...		LM & NTLM
user16	M.F0VGK	*	m.f0vgk	24BD9472F309...	2CD4412FEA72...		LM & NTLM
user17	V*QBT4UN		v*qbt4un	9770936000C9...	525C08C77DC...		LM & NTLM
user18	QTDRA#	*	qtdra#	3D8E685458DC...	2BFF5C760EA0...		LM & NTLM
user19	5GCTW4TG@N		5GCTw4tG@N	397B14DC157E...	3D9C36EAB364...		LM & NTLM
user20	TCC!TW84S		tcC!tW84s	9FD6294A8071...	06DB29F73394...		LM & NTLM
user21	QAST4@	*	qaSt4@	C6B6A22383ED...	E5091971545B9...		LM & NTLM
user22	VANE$A	*	vANe$a	BE5567E804862...	2900C3D41723...		LM & NTLM
user23	!AFU9UT	*	!aFu9ut	78E16AAB379E...	3CE68E6BBD88...		LM & NTLM
user24	C7E++AV	*	C7e++AV	20F6A1E8BA6B...	15DBA32DC2F...		LM & NTLM
user25	W2U$ATHE		w2U$atHe	8CB005976C3D...	236E057B3C4B...		LM & NTLM
user26	$_CH6CU5		$_Ch6cU5	FF6E505D6EAA...	6F525A0DCC9...		LM & NTLM
user27	KE42A2PE*		Ke42A2Pe*	075225C1E300...	C744860B070C...		LM & NTLM
user28	RA3$_EY*C		rA3$_ey*c	D89338359A25...	7811FFBD95B4...		LM & NTLM
user29	B*#D9*7YEG		b*#D9*7yEG	83EC665212A7...	F46FBE56E3FE9...		LM & NTLM
user30	8RA*_PHA$8		8rA*_pHa$8	CEEE0446ABEA...	E9F439933093E...		LM & NTLM

Figure 8.13: Passwords revealed from the cryptanalysis attack (LM hash).

Technique	Passwords Revealed
Dictionary attack	9
Cryptanalysis attack	21
Total	30

Table 8.10: Revealed passwords.

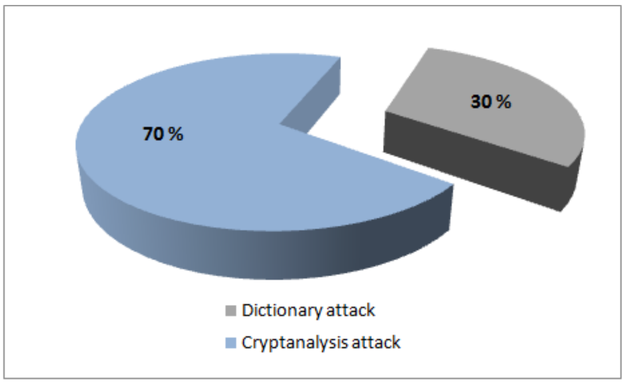


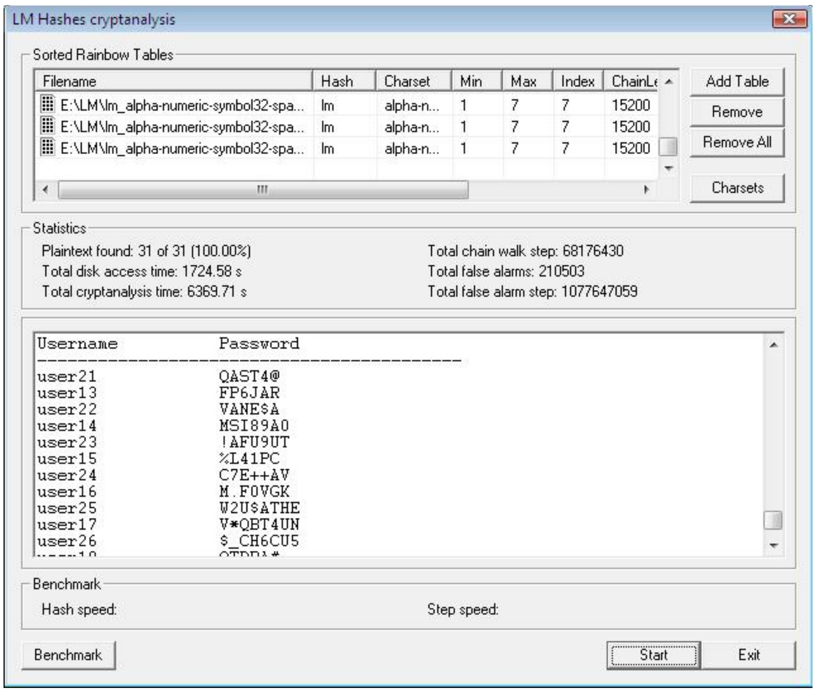
Figure 8.14: Share of revealed passwords.

Number	**User name**	**Password**	**Length**	**Category**	**Revealed?**
1	*user1*	*password1*	*9*	*Most Popular*	*Yes*
2	*user2*	*abc123*	*6*	*Most Popular*	*Yes*
3	user3	myspace1	8	Most Popular	Yes
4	*user4*	*password*	*8*	*Most Popular*	*Yes*
5	*user5*	*blink182*	*8*	*Most Popular*	*Yes*
6	*user6*	*qwerty1*	*7*	*Most Popular*	*Yes*
7	*user7*	*fuckyou*	*7*	*Most Popular*	*Yes*
8	*user8*	*123abc*	*6*	*Most Popular*	*Yes*
9	*user9*	*baseball1*	*9*	*Most Popular*	*Yes*
10	*user10*	*football1*	*9*	*Most Popular*	*Yes*
11	user11	hm71li	6	Mnemonic Passwords	Yes
12	user12	k0fzug	6	Mnemonic Passwords	Yes
13	user13	fp6jar	6	Mnemonic Passwords	Yes
14	user14	msi89a0	7	Mnemonic Passwords	Yes
15	user15	%l41pc	6	Mnemonic Passwords	Yes
16	user16	m.f0vgk	7	Mnemonic Passwords	Yes
17	user17	v*qbt4un	8	Mnemonic Passwords	Yes
18	user18	qtdra#	6	Mnemonic Passwords	Yes
19	user19	5GCTw4tG@N	10	Mnemonic Passwords	Yes
20	user20	tcC!tW84s	9	Mnemonic Passwords	Yes
21	user21	qaSt4@	6	Random Passwords	Yes
22	user22	vANe$a	6	Random Passwords	Yes
23	user23	!aFu9ut	7	Random Passwords	Yes
24	user24	C7e++AV	7	Random Passwords	Yes
25	user25	w2U$atHe	8	Random Passwords	Yes
26	user26	$_Ch6cU5	8	Random Passwords	Yes
27	user27	Ke42A2Pe*	9	Random Passwords	Yes
28	user28	rA3$_ey*c	9	Random Passwords	Yes
29	user29	b*#D9*7yEG	10	Random Passwords	Yes
30	user30	8rA*_pHa$8	10	Random Passwords	Yes

Table 8.11: The results from both the attacks (LM hash).

This means that we have revealed *all* the 30 selected passwords by using both a dictionary attack and a cryptanalysis attack on the LM hashes, and this is illustrated in Table 8.10 and Figure 8.14 above. 30% of the passwords were revealed during the dictionary attack and 70% during the cryptanalysis attack.

Even though we managed to reveal the 21 remaining passwords we still have to calculate how much time the cryptanalysis attack took to complete. To calculate the time we included *Total disk access time* and *Total cryptanalysis time* found in the middle section called *Statistics* in Figure 8.15. As can be seen from the figure the cryptanalysis attack, using all the rainbow tables, took 1724.58 + 6369.71 = 8094.29 seconds = *2 hours 14 minutes 54 seconds*. From the same figure we can also see some of the revealed passwords, which are already presented, and from the *Plaintext found* field it is confirmed that 100% of the LM hashes are found.



Filename	Hash	Charset	Min	Max	Index	ChainLe
E:\LM\lm_alpha-numeric-symbol32-spa...	lm	alpha-n...	1	7	7	15200
E:\LM\lm_alpha-numeric-symbol32-spa...	lm	alpha-n...	1	7	7	15200
E:\LM\lm_alpha-numeric-symbol32-spa...	lm	alpha-n...	1	7	7	15200

Plaintext found: 31 of 31 (100.00%)
Total disk access time: 1724.58 s
Total cryptanalysis time: 6369.71 s
Total chain walk step: 68176430
Total false alarms: 210503
Total false alarm step: 1077647059

```
Username            Password
------------------------------------------
user21              QAST4@
user13              FP6JAR
user22              VANE$A
user14              MSI89A0
user23              !AFU9UT
user15              %L41PC
user24              C7E++AV
user16              M.F0VGK
user25              W2U$ATHE
user17              V*QBT4UN
user26              $_CH6CU5
```

Hash speed:
Step speed:

Figure 8.15: Phase 2: Timing of the cryptanalysis attack on the LM hashes.

Table 8.12 gives an overview of the timing results from the dictionary attack and from the cryptanalysis attack. The dictionary attack performed in this phase was carried out on all the 30 selected passwords, while the remaining 21 hash values were subject to the cryptanalysis attack.

Technique	**Time**
Dictionary attack	1 minute 3 seconds (63 seconds)
Cryptanalysis attack	2 hours 14 minutes 54 seconds (8094 seconds)
Total	2 hours 15 minutes 57 seconds (8157 seconds)

Table 8.12: Phase 2: Time measurements for the LM hashes.

Based on the results given in Table 8.12, Figure 8.16 gives an overview of how much time the different attacks constitute of the 8 hours period available. This shows that we are far within the time limit. As can be seen from the figure, the dictionary attack constitutes 0.22% of the available time and the cryptanalysis attack constitutes 28.10% of the time. This leaves us with 71.68% unused time.

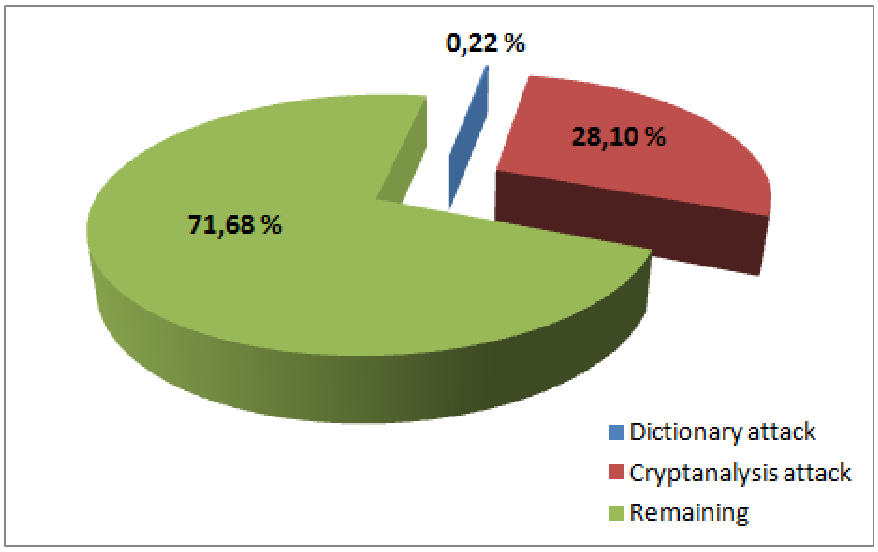


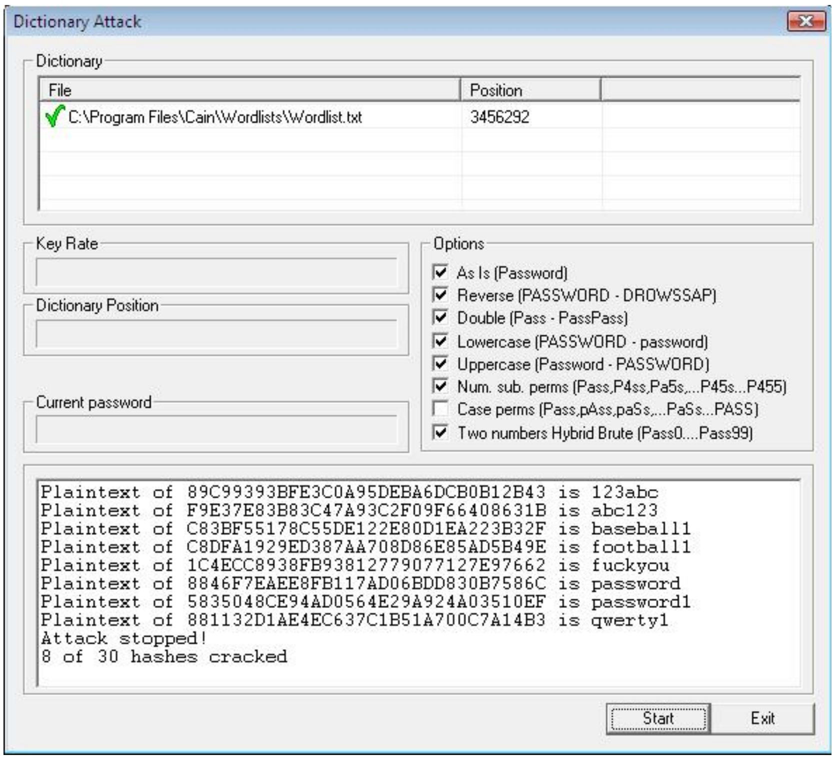
Figure 8.16: Distribution of time between the techniques applied to the LM hashes.

8.3.2.2 NT hash

Like we did for the NT hash in Phase 1, we also applied a dictionary attack and a cryptanalysis attack in this phase. The only difference was that this time the password file contained 30 NT hash values instead of one NT hash value.

First the dictionary attack was performed, and from Figure 8.17 it can be seen that the same wordlist (Wordlist.txt) and the same settings are still being used. The screen shot in the figure was taken after the dictionary attack on the NT hashes had completed, and at the bottom left of the window a message saying *8 out of 30 hashes cracked* is displayed. This message implied that 8 passwords had been revealed, and as we shall see below this was correct. For the LM hashes these numbers did not match the passwords that really had been revealed. The reason why these numbers made sense in this case was because the NT does not split the password into segments before the NT hash is generated. This results in just one hash value for each password.

Some of the passwords that were revealed through the dictionary attack can be seen in Figure 8.17, but a complete list of all the revealed passwords from the dictionary attack is given in Figure 8.18. The window shown in Figure 8.18 is explained earlier, when we presented the results from the cracking of the LM hashes (Figure 8.11). We are still going to point out some differences from the case with the LM hashes. Firstly, note that *8 passwords* were revealed from the dictionary attack. Also be aware that the *LM Password* column is empty and that the whole *LM Hash* column contains the same hash values. The reason is that the password file used in this case is obtained from a system running Windows Vista, and therefore only contains NT hash values, as LM hash is disabled by default in Windows Vista. This is also the case for Windows 7 RC. The characters used in the *LM Hash* column are just the value used when there does not exist a hash value. This means that the NT hashes are the only possible target.



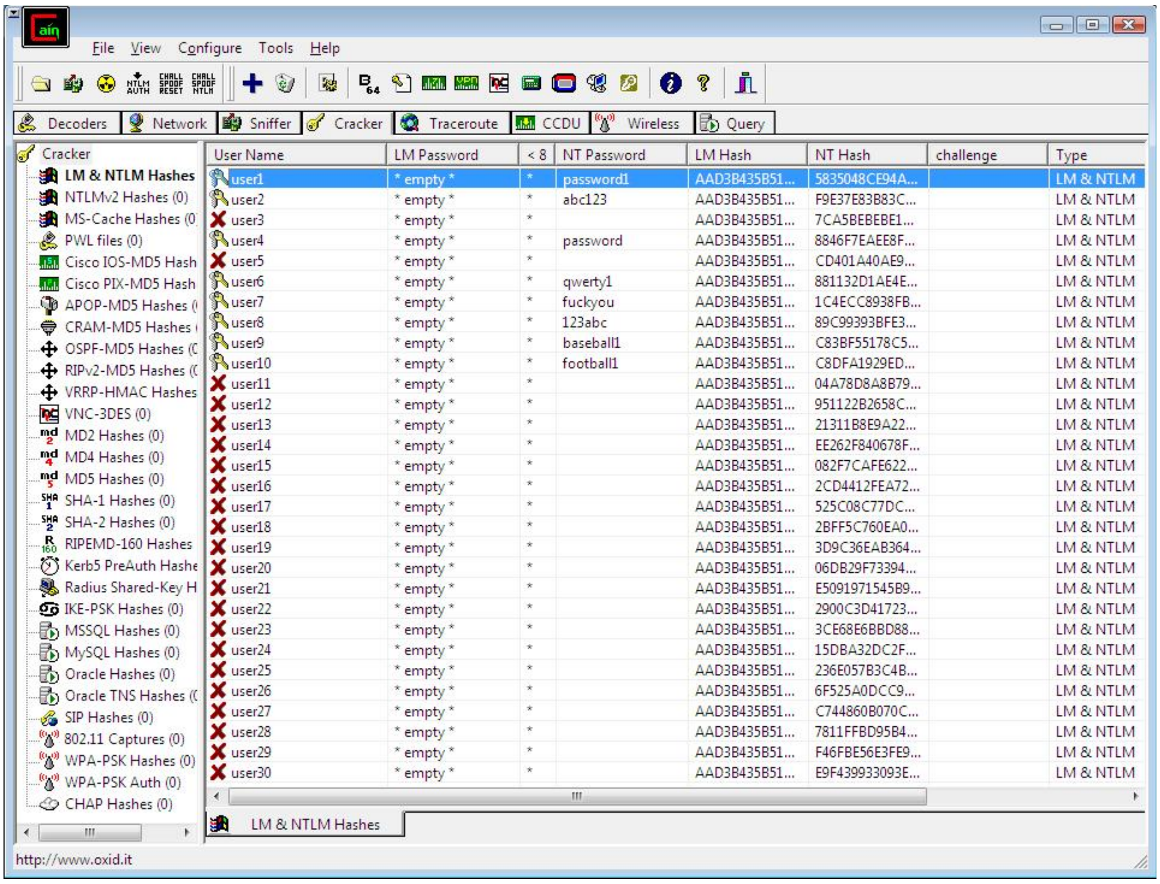
File	Position
✓ C:\Program Files\Cain\Wordlists\Wordlist.txt	3456292

- [✓] As Is (Password)
- [✓] Reverse (PASSWORD - DROWSSAP)
- [✓] Double (Pass - PassPass)
- [✓] Lowercase (PASSWORD - password)
- [✓] Uppercase (Password - PASSWORD)
- [✓] Num. sub. perms (Pass,P4ss,Pa5s,...P45s...P455)
- [ ] Case perms (Pass,pAss,paSs,...PaSs...PASS)
- [✓] Two numbers Hybrid Brute (Pass0....Pass99)

```
Plaintext of 89C99393BFE3C0A95DEBA6DCB0B12B43 is 123abc
Plaintext of F9E37E83B83C47A93C2F09F66408631B is abc123
Plaintext of C83BF55178C55DE122E80D1EA223B32F is baseball1
Plaintext of C8DFA1929ED387AA708D86E85AD5B49E is football1
Plaintext of 1C4ECC8938FB93812779077127E97662 is fuckyou
Plaintext of 8846F7EAEE8FB117AD06BDD830B7586C is password
Plaintext of 5835048CE94AD0564E29A924A03510EF is password1
Plaintext of 881132D1AE4EC637C1B51A700C7A14B3 is qwerty1
Attack stopped!
8 of 30 hashes cracked
```

Figure 8.17: Phase 2: Settings used for the dictionary attack (NT hash).

At this point we experienced a minor difference between the SAM file obtained from the Windows Vista machine and the SAM file from the machine running Windows 7 RC. When we imported the SAM file that originated from Windows 7 RC we noticed that the loaded NT hash values were different than those loaded when using the Windows Vista SAM file, even though the same user account passwords were utilized. And when we tried to run the dictionary attack no passwords were revealed. As we can see from Figure 8.18, the dictionary attack revealed 8 passwords from the *Windows Vista* SAM file. At this point we wondered if Microsoft had improved their password handling for Windows 7 RC by for instance including a salt when generating the NT hash value. This turned out to not be the case. We tried to import the Windows 7 RC SAM file into Ophcrack to see if the hash values still were different from the hash values in the Windows Vista SAM file, and in Ophcrack they were the same. This set aside our theory about improved password security, and just shows that Cain & Abel had problems reading the Windows 7 RC SAM file. When we had loaded the Windows 7 RC SAM file in Ophcrack we chose *Save* from the top menu and then *Save to file*. The saved file was then imported into Cain & Abel, which was the tool used for the dictionary attacks. Now, the hash values were the same for both the Windows 7 RC SAM file and Windows Vista SAM file. The only difference was the window shown in Figure 8.18. The *LM Password* column was still empty, as it was when using the Windows Vista SAM file, but the *LM hash* column did not contain any values, in contrast to the Windows Vista case.



User Name	LM Password	< 8	NT Password	LM Hash	NT Hash	challenge	Type
user1	* empty *	*	password1	AAD3B435B51...	5835048CE94A...		LM & NTLM
user2	* empty *	*	abc123	AAD3B435B51...	F9E37E83B83C...		LM & NTLM
user3	* empty *	*		AAD3B435B51...	7CA5BEBEBE1...		LM & NTLM
user4	* empty *	*	password	AAD3B435B51...	8846F7EAEE8F...		LM & NTLM
user5	* empty *	*		AAD3B435B51...	CD401A40AE9...		LM & NTLM
user6	* empty *	*	qwerty1	AAD3B435B51...	881132D1AE4E...		LM & NTLM
user7	* empty *	*	fuckyou	AAD3B435B51...	1C4ECC8938FB...		LM & NTLM
user8	* empty *	*	123abc	AAD3B435B51...	89C99393BFE3...		LM & NTLM
user9	* empty *	*	baseball1	AAD3B435B51...	C83BF55178C5...		LM & NTLM
user10	* empty *	*	football1	AAD3B435B51...	C8DFA1929ED...		LM & NTLM
user11	* empty *	*		AAD3B435B51...	04A78D8A8B79...		LM & NTLM
user12	* empty *	*		AAD3B435B51...	951122B2658C...		LM & NTLM
user13	* empty *	*		AAD3B435B51...	21311B8E9A22...		LM & NTLM
user14	* empty *	*		AAD3B435B51...	EE262F840678F...		LM & NTLM
user15	* empty *	*		AAD3B435B51...	082F7CAFE622...		LM & NTLM
user16	* empty *	*		AAD3B435B51...	2CD4412FEA72...		LM & NTLM
user17	* empty *	*		AAD3B435B51...	525C08C77DC...		LM & NTLM
user18	* empty *	*		AAD3B435B51...	2BFF5C760EA0...		LM & NTLM
user19	* empty *	*		AAD3B435B51...	3D9C36EAB364...		LM & NTLM
user20	* empty *	*		AAD3B435B51...	06DB29F73394...		LM & NTLM
user21	* empty *	*		AAD3B435B51...	E5091971545B9...		LM & NTLM
user22	* empty *	*		AAD3B435B51...	2900C3D41723...		LM & NTLM
user23	* empty *	*		AAD3B435B51...	3CE68E6BBD88...		LM & NTLM
user24	* empty *	*		AAD3B435B51...	15DBA32DC2F...		LM & NTLM
user25	* empty *	*		AAD3B435B51...	236E057B3C4B...		LM & NTLM
user26	* empty *	*		AAD3B435B51...	6F525A0DCC9...		LM & NTLM
user27	* empty *	*		AAD3B435B51...	C744860B070C...		LM & NTLM
user28	* empty *	*		AAD3B435B51...	7811FFBD95B4...		LM & NTLM
user29	* empty *	*		AAD3B435B51...	F46FBE56E3FE9...		LM & NTLM
user30	* empty *	*		AAD3B435B51...	E9F439933093E...		LM & NTLM

Figure 8.18: Passwords revealed from the dictionary attack (NT hash).

The 8 passwords that were revealed during this dictionary attack are listed in Table 8.13. And as can be seen from Figure 8.19 these 8 passwords constitute 27% of all the selected passwords. This leaves us with 22 remaining passwords (73%) which will be a subject to the cryptanalysis attack performed next. Note that the numbering used in Table 8.13 are just included to point out that there are 8 revealed passwords at this point. We got the same results when using the SAM file obtained from the machine running Windows 7 RC.

In Phase 1 we estimated the time for the dictionary attack on the NT hash for the selected password to be 49 seconds. The dictionary attack carried out in this phase with all the 30 selected passwords was measured to take 58 seconds. After deducting the 5 seconds it took for Cain & Abel to start, as we used the *CPU Time* displayed in *Windows Task Manager* to measure the time, the dictionary attack was measured to take *53 seconds*. As for the LM hashes, this shows that it does not necessarily take significant longer time to run a dictionary attack on a password file containing several NT hash values than it does with a password file containing just one NT hash value.

Number	User name	Password
1	user1	password1
2	user2	abc123
3	user4	password
4	user6	qwerty1
5	user7	fuckyou
6	user8	123abc
7	user9	baseball1
8	user10	football1

Table 8.13: Revealed passwords.

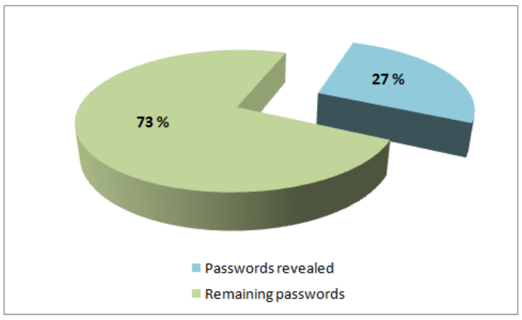
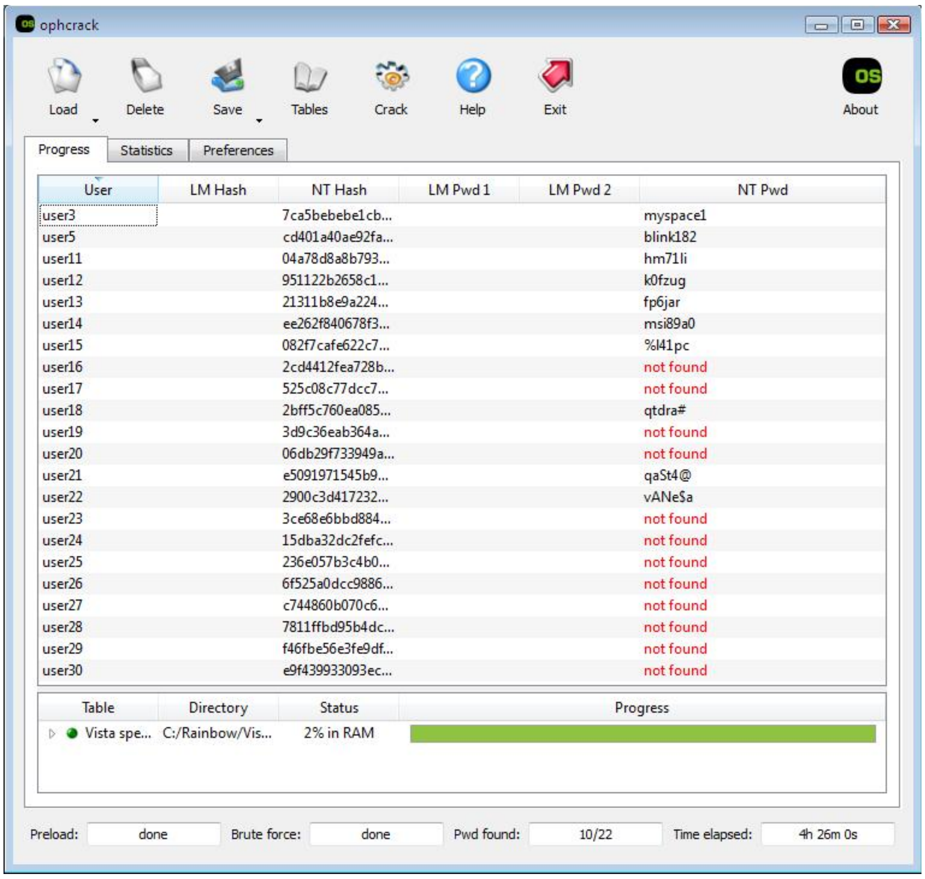


Figure 8.19: Share of revealed passwords.



User	LM Hash	NT Hash	LM Pwd 1	LM Pwd 2	NT Pwd
user3		7ca5bebebe1cb...			myspace1
user5		cd401a40ae92fa...			blink182
user11		04a78d8a8b793...			hm71li
user12		951122b2658c1...			k0fzug
user13		21311b8e9a224...			fp6jar
user14		ee262f840678f3...			msi89a0
user15		082f7cafe622c7...			%l41pc
user16		2cd4412fea728b...			not found
user17		525c08c77dcc7...			not found
user18		2bff5c760ea085...			qtdra#
user19		3d9c36eab364a...			not found
user20		06db29f733949a...			not found
user21		e5091971545b9...			qaSt4@
user22		2900c3d417232...			vANe$a
user23		3ce68e6bbd884...			not found
user24		15dba32dc2fefc...			not found
user25		236e057b3c4b0...			not found
user26		6f525a0dcc9886...			not found
user27		c744860b070c6...			not found
user28		7811ffbd95b4dc...			not found
user29		f46fbe56e3fe9df...			not found
user30		e9f439933093ec...			not found

Table	Directory	Status	Progress
Vista spe...	C:/Rainbow/Vis...	2% in RAM	

Preload: done Brute force: done Pwd found: 10/22 Time elapsed: 4h 26m 0s

Figure 8.20: Passwords revealed from the cryptanalysis attack (NT hash).

Let us now consider the *cryptanalysis attack* with the use of the NT rainbow tables. The users with a password that was revealed during the dictionary attack were moved from the list of passwords we wanted to reveal during the cryptanalysis attack. This can be seen from Figure 8.20 as the users listed in Table 8.13, which lists the users with an already revealed password, are omitted. As can be seen from the bottom menu in Figure 8.20 *10 passwords* of the 22 remaining passwords were revealed from the cryptanalysis attack. In other words, the cryptanalysis attack with the NT rainbow tables was able to reveal 45.5% of the 22 passwords that were subject to the cryptanalysis attack. From the same figure it can be verified that we used the tables named *Vista special*. The NT hashes that were not cracked during the attack are marked with *not found* in the *NT Pwd* column.

Number	**User name**	**Password**	**Length**	**Category**	**Revealed?**
1	*user1*	*password1*	*9*	*Most Popular*	*Yes*
2	*user2*	*abc123*	*6*	*Most Popular*	*Yes*
3	user3	myspace1	8	Most Popular	Yes
4	*user4*	*password*	*8*	*Most Popular*	*Yes*
5	user5	blink182	8	Most Popular	Yes
6	*user6*	*qwerty1*	*7*	*Most Popular*	*Yes*
7	*user7*	*fuckyou*	*7*	*Most Popular*	*Yes*
8	*user8*	*123abc*	*6*	*Most Popular*	*Yes*
9	*user9*	*baseball1*	*9*	*Most Popular*	*Yes*
10	*user10*	*football1*	*9*	*Most Popular*	*Yes*
11	user11	hm71li	6	Mnemonic Passwords	Yes
12	user12	k0fzug	6	Mnemonic Passwords	Yes
13	user13	fp6jar	6	Mnemonic Passwords	Yes
14	user14	msi89a0	7	Mnemonic Passwords	Yes
15	user15	%l41pc	6	Mnemonic Passwords	Yes
16	user16	m.f0vgk	7	Mnemonic Passwords	No
17	user17	v*qbt4un	8	Mnemonic Passwords	No
18	user18	qtdra#	6	Mnemonic Passwords	Yes
19	user19	5GCTw4tG@N	10	Mnemonic Passwords	No
20	user20	tcC!tW84s	9	Mnemonic Passwords	No
21	user21	qaSt4@	6	Random Passwords	Yes
22	user22	vANe$a	6	Random Passwords	Yes
23	user23	!aFu9ut	7	Random Passwords	No
24	user24	C7e++AV	7	Random Passwords	No
25	user25	w2U$atHe	8	Random Passwords	No
26	user26	$_Ch6cU5	8	Random Passwords	No
27	user27	Ke42A2Pe*	9	Random Passwords	No
28	user28	rA3$_ey*c	9	Random Passwords	No
29	user29	b*#D9*7yEG	10	Random Passwords	No
30	user30	8rA*_pHa$8	10	Random Passwords	No

Table 8.14: The results from both the attacks (NT hash).

The results presented above show that 8 passwords were revealed during the dictionary attack and 10 passwords were revealed during the cryptanalysis attack, meaning that 18 of the 30 selected passwords were revealed in total. Table 8.14 lists all the 30 passwords that were revealed. The user name, length of the password and to which category the password belongs is also included in the table. The 18 passwords that were found are listed with *Yes* in the *Revealed?* column. Note that each row in the table containing a password revealed during the dictionary attack has italic fonts. This means that the passwords revealed during the cryptanalysis attack is marked with a Yes in the *Revealed?* column but do **not** have their font in *italic*. 12 of the 30 passwords were not revealed.

Table 8.15 and Figure 8.21 illustrate the share of revealed passwords in the dictionary attack and the cryptanalysis attack. Figure 8.21 shows that 27% of the 30 selected passwords were revealed during the dictionary attack while 33% of the passwords were revealed during the cryptanalysis attack. This means that 40% of the NT hash values were not cracked.

Technique	**Passwords Revealed**
Dictionary attack	8
Cryptanalysis attack	10
Total	18

Table 8.15: Revealed passwords.

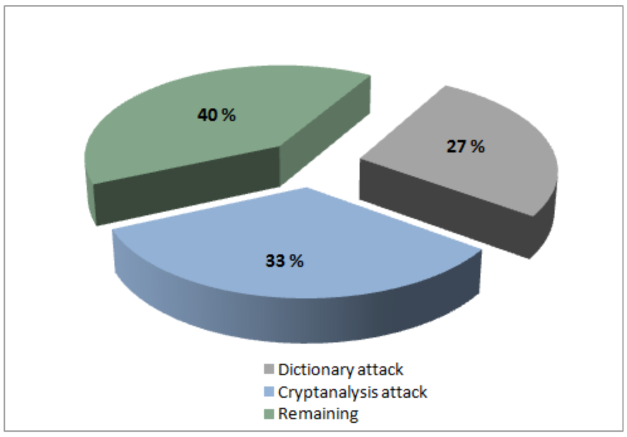


Figure 8.21: Share of revealed passwords.

To figure out how long time the cryptanalysis took to complete we had to take a look at the bottom right corner of Figure 8.20. According the *Time elapsed* field, the cryptanalysis attack took *4 hours 26 minutes*. Recall from Phase 1 that the cryptanalysis attack using the NT rainbow tables was estimated to take 22 minutes and 38 seconds. Table 8.16 gives an overview of the timing results from the dictionary attack and from the cryptanalysis attack. The dictionary attack performed in this phase was carried out on all the 30 selected passwords, while the remaining 22 hash values were subject to the cryptanalysis attack.

Technique	**Time**
Dictionary attack	53 seconds
Cryptanalysis attack	4 hours 26 minutes (15960 seconds)
Total	4 hour 26 minutes 53 seconds (16013 seconds)

Table 8.16: Phase 2: Time measurements for the NT hashes.

An illustrative overview of how much time the different attacks constitute of the 8 hours period available is given in Figure 8.22. As can be seen from the figure, the dictionary attack constitutes 0.18% of the available time and the cryptanalysis attack constitutes 55.42% of the time. This leave us with 44.40% unused time. This shows that we are within the time limit of 8 hours.

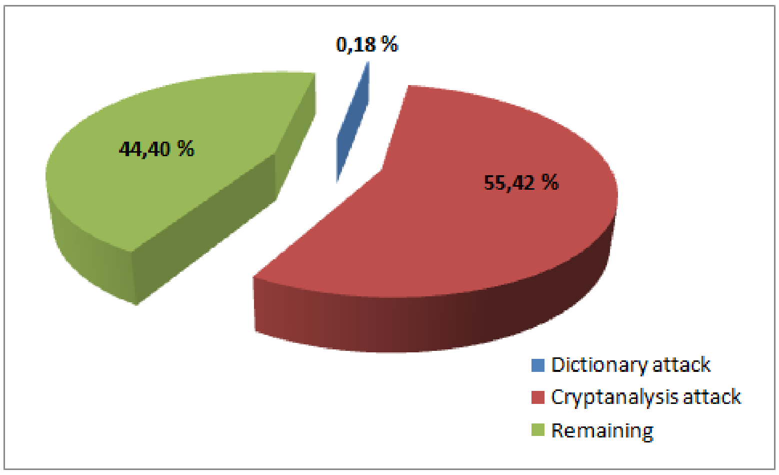
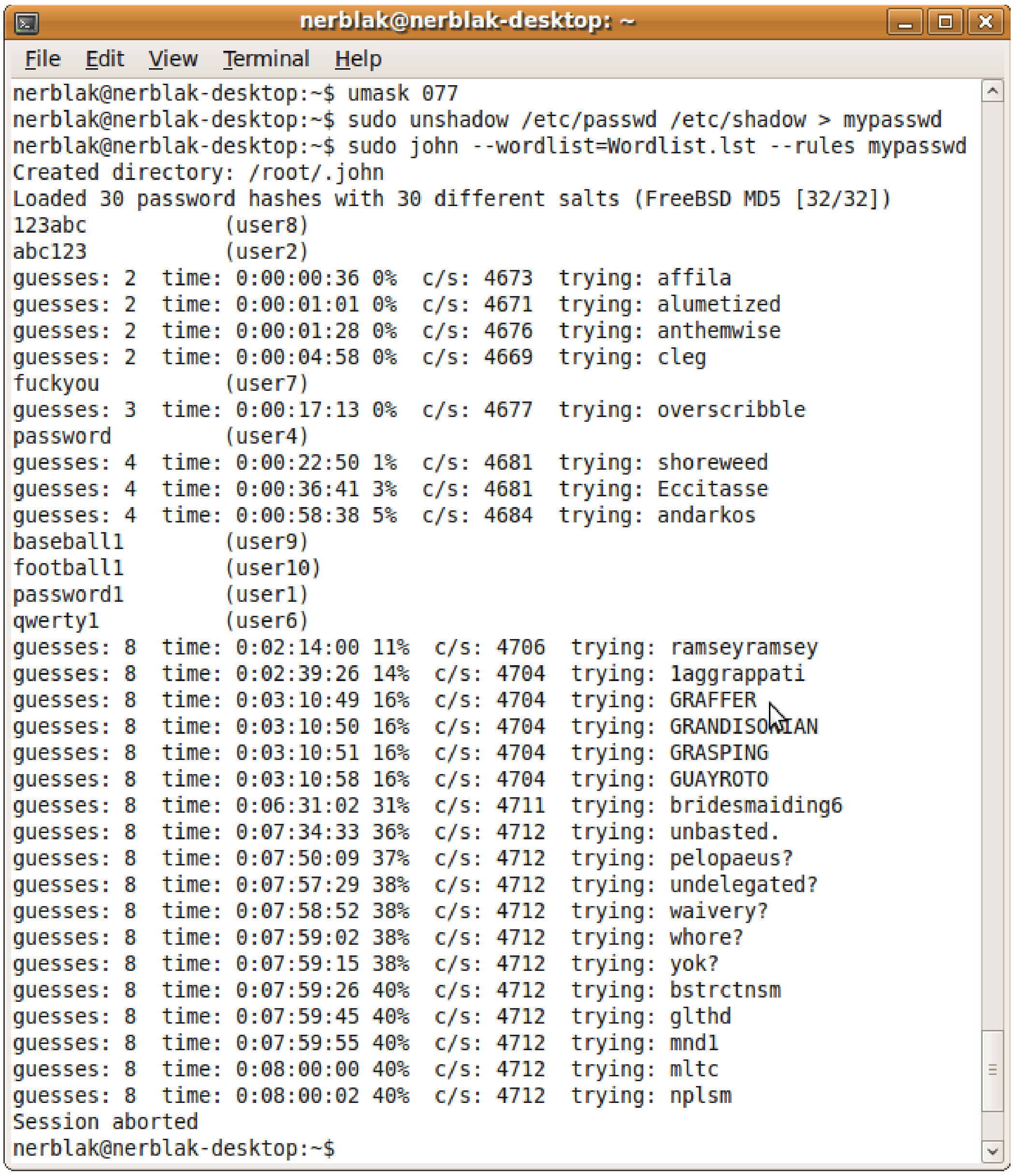


Figure 8.22: Distribution of time between the techniques applied to the NT hashes.

It should be added that we got almost the same results when using the Windows 7 RC SAM file as we did when we used the Windows Vista SAM file. The results from the dictionary attack was identical, and the only difference was that the cryptanalysis attack performed on the Windows 7 RC SAM file took 5 hours 15 minutes and 28 seconds to complete while, as already presented, the same attack took 4hours and 26 minutes on the Windows Vista SAM file. The exact same passwords were revealed in both cases. This means that the NT hash values stored in the Windows Vista SAM file and the Windows 7 RC file are identical, and that Microsoft has not increased the security to the newly released Windows 7 RC operating system regarding the login passwords that are hashed and stored in the SAM file. Microsoft has for example still not included a salt when generating the NT hash values.

8.3.2.3 MD5 hash

For the MD5 hash a dictionary attack was the only attack that was applied. The only difference between the dictionary attack performed in Phase 1 and in this phase is that the password file contained 30 MD5 hash values in this phase instead of one MD5 hash value, which was the case in Phase 1. John the Ripper was the tool used for the dictionary attack, and the dictionary attack was carried out during the whole 8 hours period.

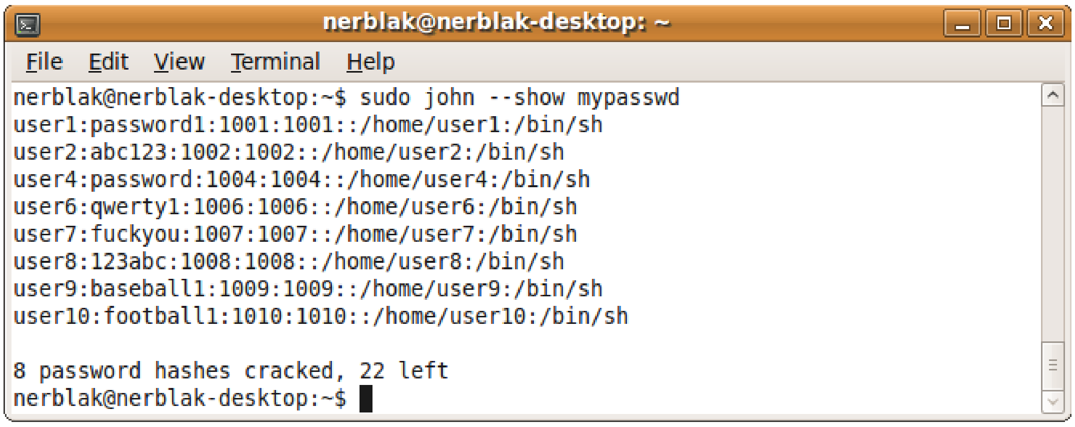


```
nerblak@nerblak-desktop:~$ umask 077
nerblak@nerblak-desktop:~$ sudo unshadow /etc/passwd /etc/shadow > mypasswd
nerblak@nerblak-desktop:~$ sudo john --wordlist=Wordlist.lst --rules mypasswd
Created directory: /root/.john
Loaded 30 password hashes with 30 different salts (FreeBSD MD5 [32/32])
123abc           (user8)
abc123           (user2)
guesses: 2  time: 0:00:00:36 0%  c/s: 4673  trying: affila
guesses: 2  time: 0:00:01:01 0%  c/s: 4671  trying: alumetized
guesses: 2  time: 0:00:01:28 0%  c/s: 4676  trying: anthemwise
guesses: 2  time: 0:00:04:58 0%  c/s: 4669  trying: cleg
fuckyou          (user7)
guesses: 3  time: 0:00:17:13 0%  c/s: 4677  trying: overscribble
password         (user4)
guesses: 4  time: 0:00:22:50 1%  c/s: 4681  trying: shoreweed
guesses: 4  time: 0:00:36:41 3%  c/s: 4681  trying: Eccitasse
guesses: 4  time: 0:00:58:38 5%  c/s: 4684  trying: andarkos
baseball1        (user9)
football1        (user10)
password1        (user1)
qwerty1          (user6)
guesses: 8  time: 0:02:14:00 11%  c/s: 4706  trying: ramseyramsey
guesses: 8  time: 0:02:39:26 14%  c/s: 4704  trying: laggrappati
guesses: 8  time: 0:03:10:49 16%  c/s: 4704  trying: GRAFFER
guesses: 8  time: 0:03:10:50 16%  c/s: 4704  trying: GRANDISONIAN
guesses: 8  time: 0:03:10:51 16%  c/s: 4704  trying: GRASPING
guesses: 8  time: 0:03:10:58 16%  c/s: 4704  trying: GUAYROTO
guesses: 8  time: 0:06:31:02 31%  c/s: 4711  trying: bridesmaiding6
guesses: 8  time: 0:07:34:33 36%  c/s: 4712  trying: unbasted.
guesses: 8  time: 0:07:50:09 37%  c/s: 4712  trying: pelopaeus?
guesses: 8  time: 0:07:57:29 38%  c/s: 4712  trying: undelegated?
guesses: 8  time: 0:07:58:52 38%  c/s: 4712  trying: waivery?
guesses: 8  time: 0:07:59:02 38%  c/s: 4712  trying: whore?
guesses: 8  time: 0:07:59:15 38%  c/s: 4712  trying: yok?
guesses: 8  time: 0:07:59:26 40%  c/s: 4712  trying: bstrctnsm
guesses: 8  time: 0:07:59:45 40%  c/s: 4712  trying: glthd
guesses: 8  time: 0:07:59:55 40%  c/s: 4712  trying: mndl
guesses: 8  time: 0:08:00:00 40%  c/s: 4712  trying: mltc
guesses: 8  time: 0:08:00:02 40%  c/s: 4712  trying: nplsm
Session aborted
nerblak@nerblak-desktop:~$
```

Figure 8.23: Passwords revealed from the dictionary attack (MD5 hash).

From Figure 8.23 it can be seen which of the 30 selected passwords that were revealed during the dictionary attack, and also in which order they were found. From the *john* command in the third line in the figure it can be seen that the same wordlist (Wordlist.lst) is still used, and that mangling rules are applied (−−rules). As John the Ripper has a built-in timing function, it can also be verified that the attack was stopped after 8 hours as planned. The line saying *Session aborted* confirms that the attack was aborted before it was able to complete, and as we can see from the same figure, about 40% of the dictionary attack completed. During the 8 hours the dictionary attack revealed *8 passwords* in total. From line 5 in the figure saying *Loaded 30 password hashes with 30 different salts*, it can be seen that all the 30 password hashes were loaded and that each of the 30 MD5 hash values were generated by using different salt values. And if you take a look at the first 3 lines of the figure you will also recognize the *umask*, the *unshadow* and the *john* command which is already described. The field named *c/s* in the figure gives the number of combinations of usernames and passwords per second.

Another overview of the revealed passwords has been included in Figure 8.24. It presents the same findings as found in Figure 8.23 but in a more easy-to-follow way. You can get this overview by writing *sudo john −−show mypasswd*, where mypasswd is the name of the file containing the content of both the passwd and the shadow file. The command has to be run after the dictionary attack has completed or been aborted. In addition, the figure explicitly informs us about the number of passwords found: *8 password hashes cracked, 22 left.*



```
nerblak@nerblak-desktop:~$ sudo john --show mypasswd
user1:password1:1001:1001::/home/user1:/bin/sh
user2:abc123:1002:1002::/home/user2:/bin/sh
user4:password:1004:1004::/home/user4:/bin/sh
user6:qwerty1:1006:1006::/home/user6:/bin/sh
user7:fuckyou:1007:1007::/home/user7:/bin/sh
user8:123abc:1008:1008::/home/user8:/bin/sh
user9:baseball1:1009:1009::/home/user9:/bin/sh
user10:football1:1010:1010::/home/user10:/bin/sh

8 password hashes cracked, 22 left
nerblak@nerblak-desktop:~$ █
```

Figure 8.24: A list of the revealed passwords sorted by username.

The 8 passwords that were revealed during the dictionary attack are listed in Table 8.17. Note that the passwords are sorted on the user name, meaning they are not listed in the order they were revealed, and the numbering in the left column shows that there are 8 revealed passwords. As Figure 8.25 points out, these 8 passwords constitute 27% of all the 30 selected passwords. This means that the remaining 22 passwords, which constitute the remaining 73%, was not found since the dictionary attack was the only attack performed on the MD5 hash values.

Table 8.18 lists all the 30 passwords to give an overview of which of the selected passwords that were revealed, and which did not. This can be seen from the column labeled *Revealed?*

Number	User name	Password
1	user1	password1
2	user2	abc123
3	user4	password
4	user6	qwerty1
5	user7	fuckyou
6	user8	123abc
7	user9	baseball1
8	user10	football1

Table 8.17: Revealed passwords.

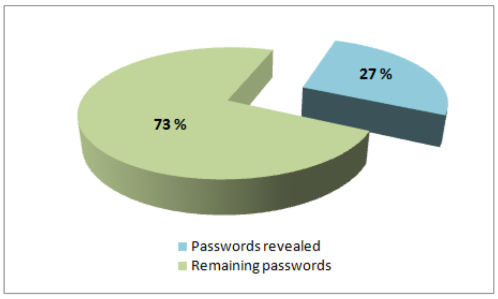


Figure 8.25: Share of revealed passwords.

Number	User name	Password	Length	Category	Revealed?
1	user1	password1	9	Most Popular	Yes
2	user2	abc123	6	Most Popular	Yes
3	user3	myspace1	8	Most Popular	No
4	user4	password	8	Most Popular	Yes
5	user5	blink182	8	Most Popular	No
6	user6	qwerty1	7	Most Popular	Yes
7	user7	fuckyou	7	Most Popular	Yes
8	user8	123abc	6	Most Popular	Yes
9	user9	baseball1	9	Most Popular	Yes
10	user10	football1	9	Most Popular	Yes
11	user11	hm71li	6	Mnemonic Passwords	No
12	user12	k0fzug	6	Mnemonic Passwords	No
13	user13	fp6jar	6	Mnemonic Passwords	No
14	user14	msi89a0	7	Mnemonic Passwords	No
15	user15	%l41pc	6	Mnemonic Passwords	No
16	user16	m.f0vgk	7	Mnemonic Passwords	No
17	user17	v*qbt4un	8	Mnemonic Passwords	No
18	user18	qtdra#	6	Mnemonic Passwords	No
19	user19	5GCTw4tG@N	10	Mnemonic Passwords	No
20	user20	tcC!tW84s	9	Mnemonic Passwords	No
21	user21	qaSt4@	6	Random Passwords	No
22	user22	vANe$a	6	Random Passwords	No
23	user23	!aFu9ut	7	Random Passwords	No
24	user24	C7e++AV	7	Random Passwords	No
25	user25	w2U$atHe	8	Random Passwords	No
26	user26	$_Ch6cU5	8	Random Passwords	No
27	user27	Ke42A2Pe*	9	Random Passwords	No
28	user28	rA3$_ey*c	9	Random Passwords	No
29	user29	b*#D9*7yEG	10	Random Passwords	No
30	user30	8rA*_pHa$8	10	Random Passwords	No

Table 8.18: The results from the dictionary attack (MD5 hash).

In Phase 1 we estimated the time for the dictionary attack on the MD5 hash for the selected password to be 51 minutes and 48 seconds. The dictionary attack carried out in this phase with all the 30 selected passwords was aborted after 8 hours, as this was the time limit set for the experiment. This means that the attack did not complete. Table 8.19 gives an overview of the time measurement for the dictionary attack on the MD5 hash values. In other words, the dictionary attack took 100% of the available time.

Technique	**Time**
Dictionary attack	8 hours (28800 seconds)

Table 8.19: Phase 2: Time measurement for the MD5 hashes.

Chapter 9

Discussion

In this thesis we have presented various procedures and tools to reset or recover an administrator's login password. We also performed an empirical password study to test if a set of passwords, retrieved in hidden form (hashed) from various operating systems, were possible to recover. In this thesis we also created a tool based on the experience we gained through all of the previously mentioned experiments. We have tested and experimented with our tool and other procedures and tools on many popular operating systems and have accomplished results we never expected before we began writing this thesis.

9.1 Procedures and Tools

In this thesis we have presented 6 procedures in Chapter 5, and 10 tools in Chapter 6 that can be used to reset or recover the login password for Administrator accounts on various Windows and Unix-based operating system. Some of the procedures and tools were easier to use than others, and in Table 7.1, 7.2 and 7.3 in Section 7.1 we presented an *ease of use* field where we gave the procedure or tool a score representing how easy it was to use. A procedure or a tool had three factors that we observed when the experiments were carried out. These factors were how much technical skills that were needed to carry out the procedure or tool, how many prerequisites were needed, and how much time the procedure or tool used. The basis of the *easy to use* score was generated from a combination of these three factors. To be mentioned in this thesis, the procedure or tool had to have at least one successful approach on at least one operating system.

During our research and during the experiments we only found *procedures* concerning password resetting. Because all of the operating systems we tested with hide their login passwords, usually as a hash value, no *procedure* could easily reveal these passwords. If the passwords had been stored in clear text, or if the hash algorithm would have been easily reversible, a simple procedure could be to find these passwords using for instance a live CD. Fortunately the passwords were stored much more securely with hash algorithms that were hard to reverse. On the other hand, a big problem with some of the hash algorithms had weaknesses exploitable by well known password cracking *tools*. You could manually compute a hash value and compare it with the stored one, although this might be very time consuming and inefficient. The tools we presented in Chapter 6 and 8 automate this process and utilize the power of a computer. The tools have various password cracking techniques, as described in Section 2.1.2.2 and presented in Section 2.4.2.

In Table 7.1, 7.2 and 7.3 found on page 124 the result of the experiments with procedures and tools to reset or recover a password was presented. In these tables we included a field called *Operating System*. This field presented what operating systems worked for the specific procedure or tool when the approach was carried out. What operating systems the particular procedure or tool supported is listed in Chapter 4. The tables presented in Chapter 4 may be considered a result in itself, describing where each procedure and tool can be accomplished successfully. We performed two different approaches with Tool 8, John the Ripper as presented in Table 7.3, which is the reason why the tool is displayed twice.

9.1.1 Evaluation of Procedures

The first procedure presented in this thesis, Procedure 1: *Windows XP Repair Backdoor*, we gave the score 3. This procedure was very time consuming and required a lot of intervention from the user. The procedure also required that the user had an installation CD present. The procedure only worked for that specific operating system (Windows XP) and was therefore not very flexible. As there are many other procedures to choose for this operating system, we would advice the user to rather use one of these.

Procedure 2 *Edit Shadow File in Unix Systems* also got the score 3. The reason for this was that this procedure had some prerequisites that were hard to fulfill. The procedure required that the user had a precomputed hash, and this had to be in the same format as the hash value being replaced. Procedure 2 was also quite difficult to perform. The positive thing with this procedure was that it was not as time consuming as the first procedure mentioned. Procedure 3, *Shortcut for Bypassing Regular Login Procedure*, on the other hand was a very practical and simple procedure to perform. If Windows XP is the operating system where the password is lost, this procedure should be the first choice for the user trying to get access. The procedure had no prerequisites and was performed in less than a minute. The only problem with this procedure was that it only worked on Windows XP. In addition it is very common to set the Administrator password, and then the procedure is useless. The procedure is worth a try if you do not know if the Administrator password is set.

Procedure 4, *Replace a Pre-login Executable File*, had the big advantage that it worked on every Windows version we tested it on, even the new release candidate of Windows 7. The procedure required a bit technical skills, but if the approach described at page 78 were to be followed perfectly, it should not be that hard to complete successfully. This procedure had a prerequisite, but this was not a very difficult task to obtain. The only requirement was that the user had write access to the file system of the operating system subject to the password resetting. Getting file access can be done with almost every live CD that is able to mount the NTFS partition the operating system is installed on. The procedure can also be performed with a Windows installation CD. Based on this we gave this procedure the score 2.

Procedure 5, *Use Mac OS X Installation Disk*, was specific to Mac OS X. The procedure was easily accomplished as long as the prerequisite, having a Mac OS X installation disk, was fulfilled. The procedure is also suggested procedure by Apple, and as a result of this we gave the procedure the score 1.

The last resetting procedure described in this thesis was Procedure 6, *Windows Password Reset Disk*. This procedure is Microsoft's own answer to the problem when a user has lost the login password. When we tested the procedure it worked for all the Windows operating systems described in our thesis, and was very easily carried out as long as the prerequisite were met. In Procedure 6 the user had to create the password reset disk before the password was lost. The problem with this was that users will usually not create this disk, and when the disk is needed it is too late. The prerequisite required for this procedure was the reason why we gave this procedure score 2. Note that the *Windows Password Reset Disk* only works for the specific operating system it is created on.

9.1.2 Evaluation of Tools

The first tool we presented in this thesis was Tool 1, PCLoginNow 2.0.1. This tool was a very good tool if the goal was to set the Administrator password to blank (password resetting). Although setting the password to blank was the only option concerning password handling that this tool could offer, we chose to give this tool score 1 because it was so simple and worked on all Windows systems we experimented with. If the goal was to set a new password, this could have easily been done afterwards, when the user had logged in as Administrator with a blank password. The tool was quick and had a simple interface that among other things gave the user the possibility to enable the Administrator account if this was disabled. Enabling the Administrator account is an important feature because this account is often disabled by default.

Tool 2, *Offline NT Password & Registry Editor*, was one of the best tools we experimented with in this thesis. It had the possibility reset or blank the password on any account on all of the Windows systems we experimented with. It booted the system very fast, but to some people it might be a bit difficult to understand. The tool did not have a graphical interface such as Tool 1 but had support for enabling the Administrator account. We chose to give the tool score 1 because of its speed and functionality.

Tool 3 consists of two possible approaches, and both Tool 3.1 and Tool 3.2 had much similarities and the prerequisites were the nearly same. Tool 3.1, Password Renew, was a bit quicker than Tool 3.2, Windows Gate. For some reason we experienced that Password Renew did not work for Windows Server 2008 when we tested it. Windows Gate had some more features than Password Renew, but none of relevance for the topic in this thesis. We chose to give both of the tools score 2 because of the prerequisites needed for these tools. The process of creating the BartPE bootable CD with these tools was very time consuming and required a Windows XP installation disk.

Tool 4, Kon-Boot, was one of the best tools we tested in this thesis. It was very small in size and only required the user to *press any key* for it to work. It worked on both Windows and Unix-based systems but of some reason it did not work on the latest release candidate of Windows 7. There can be many possible reasons for this, one might be that Windows has fixed the backdoor utilized by Kon-Boot. Because of the simplicity and because the tool worked on many different operating systems we chose to give this tool score 1. Tool 5, DreamPackPL, on the other hand, we chose to give the score 3. This was because it was one of the most difficult tools to handle, and we only got it to work on Windows XP. The positive thing about this tool was that it had much functionality once it was installed.

The last password resetting tool we presented in this thesis was the built in Unix tool *passwd*. This tool, Tool 6 *passwd*, required no preparation in itself as it was already installed on the system. In other words, the tool had no prerequisites, and could be initiated without any physical media such as a CD. The problem was that the user had to have root or Administrator access to use it. This was nevertheless easily acquired by performing some of the supplementary procedures; 7.1, 7.2, 8 or 9. We chose to give this tool score 2 and not 1, because of its dependency of the supplementary procedures mentioned.

In addition we presented 4 *password recovery* tools in this thesis. In all of the approaches described in Section 7.1.2 (Password Recovery) the goal was to reveal two passwords, *password* and *Hax0R*. The first password recovery tool, Tool 7 Ophcrack, was in our opinion a very good tool, exploiting the weakness of hash algorithms that lack the use of salt. We gave the tool score 1. This tool worked for both LM and NT hash values, thus capable of recovering passwords on all the Windows operating systems we experimented with. Tool 7 had a nice graphical interface, and started immediately when the system had booted. It was very easy to use, but yet very powerful when it came to password cracking. The results of the approach described at page 110 would have been much better if we had used supplementary rainbow tables. With the free rainbow tables that were included on Ophcrack Vista Live CD 2.1.0 we were able to reveal *password* but not *Hax0R*. This was because the character set for the rainbow tables included did not cover the characters of the latter password, leading to a password cracking process with rainbow tables that was completed in less than 12 minutes. In Chapter 8 we demonstrate the real power of Ophcrack using larger rainbow tables.

With Tool 8, John The Ripper, we carried out two approaches. When we used the tool in the first approach with Windows XP as the host operating system, we managed to reveal both of the passwords. Because of the structure of the LM hash, the passwords were revealed in only uppercase letters by John the Ripper. With patches of Tool 8, the correct lower and uppercase password can be revealed. In the tool YALP, described in Section 7.2, which the authors of this thesis created, we included a tool called *lm2ntcrack*. This tool is able to try all combinations of the revealed LM password in such a way that when the correct mix of uppercase and lowercase letters generates the correct NT hash value, the correct password is found. In the second approach we used John the Ripper with Ubuntu server 8.10 as the host operating system. Ubuntu Server 8.10 used the hash algorithm MD5 to hide *password* and SHA-512 hash algorithm to hide *Hax0R*. As the password cracking tool did not support SHA-512, this password was not revealed. It is said that patches can be found for John the Ripper to support both NT and SHA-512 algorithms, but this was out of scope for this thesis. Tool 8 is a text based tool, and may therefore seem deterrent to some people. The tool supports many different hashes from various operating systems. It has powerful cracking modes, and is one of the most popular password recovery tools available. We chose to give this tool a score 2 because carrying out both of the approaches may be a bit difficult, yet the tool revealed both of the passwords on one of the approaches very fast.

Tool 9, Cain & Abel, was in our opinion an excellent password recovery tool. It had support for many password cracking techniques and support for many different types of rainbow tables. The tool could reveal passwords from many different operating systems. In the approach described in detail on page 115, we use rainbow tables that are capable of revealing every password consisting of less 15 characters in Windows XP. By using rainbow tables the approach was harder to carry out as the tables had to be downloaded in advance. The tool was easy to use, but because of the prerequisites (rainbow tables) we gave this tool a score 2. By using this tool and this approach, we managed to reveal both of the passwords. This approach was the only approach that revealed the passwords correctly, even with mixed-case letters.

The last tool we presented at page 117 was Tool 10, LCP 5.05. This tool was a simple password recovery tool, with the support of revealing most Windows operating system passwords. When the approach for this tool was carried out we chose to stop the password recovery process after 15 minutes. This was because the tool did not find the password *Hax0R* in any dictionaries, and began using brute-force as password cracking technique. Because of the simplicity of the tool and the support for revealing both LM and NT hashed passwords, we chose to give the tool score 1. Tool 10 revealed *password* but not *Hax0R*. As with some of the other tools mentioned, this password was revealed easily because it was present in the dictionary used by the tool.

9.1.3 General Discussion

We experienced that some of the procedures was hard to carry out and that each tool often support separate systems. We identified that a live CD containing many of the tools and an implementation of some of the procedures were desirable. As a reaction to this we began creating a tool called YALP described more in detail in Section 7.2 and 9.3. By collecting many well known tools and implementing some of the procedures mentioned earlier, YALP is able to reset or recover most of the passwords on most of the systems presented in this thesis.

At first glance, one may think that it is much better to recover the password than to reset it. Both of the approaches have their pros and cons. A big problem with password recovery is that you usually do not know much about the strength of the password. Since the recovery process can take a large amount of time, resetting is often performed in just minutes. A negative side of revealing a lost password is that this will often not require changes to the system, and attackers may reveal the password unnoticed to the owner of the system. Another negative side of an attacker revealing a users password, is that users tend to use the same passwords on many different systems or services and if one is revealed, the attacker might try this password on other systems as well. A revealed password may lead to a disaster for both private persons and enterprises.

In this thesis we describe both procedures and tools. People may argue that a procedure is better than using a tool or the other way around. Both have their positive and negative sides. Using a tool may simplify the resetting of passwords, and make recovery of passwords possible. On the other hand, some of the resetting procedures might be clever to check before a tool is used, because they usually require little prerequisites and some are very quickly carried out. If it was our choice we would first identify the operating system subject to password recovery or resetting. If the system was Windows XP, choosing Procedure 3 (Shortcut for Bypassing Regular Login Procedure) taking only half a minute, would be a good start. If this procedure did not work but the system was a Windows system, of course checking for a password reset disk described in Procedure 6 could be a simple solution to the problem. If it was desirable that the system should be unchanged, using a password recovery tool such as Tool 7, 8, 9 and 10 would be a good idea. When it comes to the Unix-based systems, using either Procedure 5 or one of the supplementary procedures 7, 8 or 9 in a combination with Tool 6, the password could easily be reset. A weak password may also be revealed using for instance Tool 8 (John the Ripper). If the resetting of the password was desirable, Tool 1 or 2 would be a wise choice. One of the tools that came as the biggest surprises to the authors of this thesis was Tool 4, Kon-Boot. This tool is very simple, and when loaded there is no need for a login password. You can log in in with any accounts available without typing a password, and make the necessary changes, such as changing the password.

If a person is in possession of a super computer or a large set of distributed computers for instance via a botnet, much time can be saved on rainbow table generation or password cracking. Even if the password is hashed with a salt, the rainbow tables could be generated in real time, and at the same time password cracking could take place. The use of salt will then just be a small hindrance in revealing the password.

In the poster mentioned in Section 2.4.3.2, and included as Appendix F, we present an approach on generating NT hash rainbow tables with the use of Winrtgen. In the approach we performed a benchmark using *Fredriksholm*. The hardware specifications can be found in Section 3.1. The approach presented in this poster is unfeasible to carry out because of the time it will take to generate the rainbow tables (5670 years). We therefore propose that the use of one or more supercomputers would drastically reduce both the time used for the generation of rainbow tables and the time used for cryptanalysis to reveal the password.

To sum up, choosing what procedure or tool to use depends on how much time that is available, what kind of system that is used and how much technical skill a person has and is willing to involve in the process.

9.2 Empirical Password Study

This section serves the purpose of discussing what can be extracted from the results presented in Section 8.3 in the preceding chapter, regarding the empirical password study. We will try to answer some of the questions asked, for example if it was advantageous to run a dictionary attack before the cryptanalysis attack. Comments on how many and which of the passwords that were revealed will also be given, as well as a comparison of the time measurements that were performed. We have chosen to divide this section into four parts. The first part discusses results related to the time measurements for the cracking of both the LM and NT hash, and the second part discuss the results regarding the password cracking of the LM and NT hash. The third part concentrate on both the time measurements and password cracking related to the *MD5* hashes. The reason why we chose to separate the MD5 hash from the LM and NT hashes in this discussion was that no cryptanalysis attack was performed on the MD5 hashes. The LM and NT hashes are much more related to each other. The last part discusses other general results obtained in this study.

Figure 9.1 presents the main results from the password cracking phase in the Empirical Password Study. As can be seen, 100% of the LM hashes were cracked during a working day (8 hours). This is a disturbing result because the LM hash, which is used in Windows XP, is still widely in use. From Figure 9.1 in Chapter 1 it can be seen that Windows XP constitute about 64% of the operating system market share. When it comes to the NT hashes, 60% were cracked during the same period. The NT hash is the successor of the LM hash and is used in for instance Windows Vista. 27% of the MD5 hashes were cracked, but it should be noted that the MD5 hashes were only exposed to dictionary attacks.

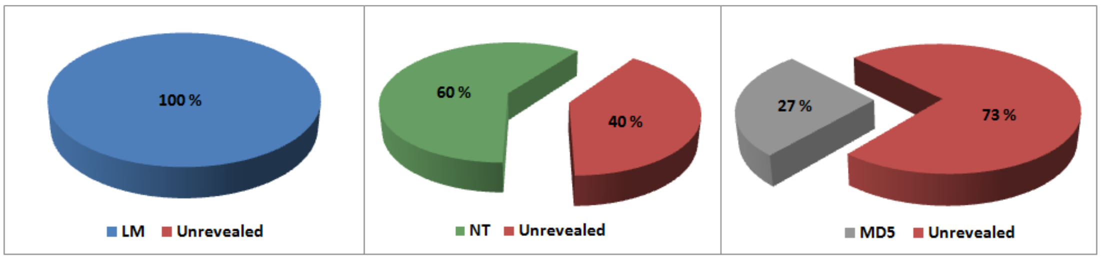


Figure 9.1: Results from the empirical password study.

9.2.1 Time Measurements for LM and NT

In Phase 1 the dictionary attack on the *LM hash* of the selected password (8rA*_pHa$8) was measured to take 1 minute and 8 seconds, and the cryptanalysis attack was estimated to take 1 hour 1 minute and 2 seconds. This can be seen from Figure 8.6 in Section 8.3.1.1. In other words, the dictionary attack constituted 0.24% and the cryptanalysis attack constituted 12.72% of the available time. The time measurements from Phase 2 showed that the dictionary attack on the LM hash took 1 minute and 3 seconds (0.22%) and the cryptanalysis attack took 2 hours and 14 minutes (28.10%) when using all the 30 hash values. This shows that the timing result for the dictionary attack in Phase 1 was about the same as the result obtained from the dictionary attack in Phase 2, even though we only used one hash value in Phase 1 and 30 hash values in Phase 2. Thus it does not necessarily take longer time when performing a dictionary attack on a password file containing many hash values than it does if the password file contains only one hash value. The reason is that the dictionary attack performs parallel operations on several of the 30 hash values. This implies that if you shall crack several password hashes, you should make a password file containing all the hash values instead of running repeatedly attacks on different password files, each containing one password hash. At least our experience was that cracking multiple hash values simultaneously was much more *time-saving* than cracking each hash value one at the time. Additionally we experienced that it was more *practical* to store all the password hashes in one file instead of having a lot of passwords files containing one hash value each. These timing results also show that the dictionary attack constitute just a small portion of the total time used in our experiment when trying to crack the LM hash. As will be discussed below, the timing results of a dictionary attack depends on factors as how comprehensive mangling rules you use and also the size of the wordlist. This is regardless of what type of hash you run the dictionary attack on.

The dictionary attack performed on the *NT hash* of the selected password in Phase 1 was measured to take 49 seconds, and the cryptanalysis attack was estimated to take 22 minutes and 38 seconds. These results can be seen from Figure 8.7 in Section 8.3.1.2. This shows that the dictionary attack constituted 0.17% of the 8 hours available, and the cryptanalysis attack constituted 4.72%. To be able to do some comparison, we need to consider the actual results obtained on all the 30 password hashes in Phase 2. In Phase 2 the dictionary attack took 53 seconds (0.18%), which shows that the estimate (49 seconds) was quite good, even though the estimate in Phase 1 was based on the use of only one NT hash value. This was also the case for the LM hash, as described above, and the same reasoning apply to the NT hash as for the LM hash.

For the cryptanalysis attack on the other hand, there was quite big difference between the estimate and the actual result for both the LM hash and the NT hash. As mentioned, for the LM hash we estimated the cryptanalysis attack to constitute 12.72% of the available time, but instead it constituted 28.10%. For the cryptanalysis attack on the NT hash the difference between the estimated time and the actual result was even bigger. In Phase 1 the estimated time for the cryptanalysis attack was 22 minutes and 38 seconds (4.72%), while the actual result showed that it took 4 hours and 26 minutes (55.42%). Recall that for the cryptanalysis attack in Phase 1 we did not use all of the rainbow tables, but instead calculated an estimate based on the result we got when using just parts of them. Together with the fact that Phase 2 considered all of the 30 hash values, this could be the reason for the big difference between the estimate from Phase 1 and the actual result from Phase 2. This concerns both the LM and the NT hash.

To be able to give a good explanation of the differences mentioned above, we need to take a look at how the rainbow tables work again. For the first table a rainbow chain with its intermediate values are generated. And if not a match is found when using the first table, the chain may be reused when the other tables are loaded into memory gradually. The size of the rainbow tables used is among other aspects dependent on the length of the rainbow chains. Short chains give larger rainbow tables as more starting points and endpoints need to be stored. This again leads to frequent lookups in the rainbow tables. Larger chains give smaller tables, as fewer starting points and endpoints need to be stored in the rainbow tables. The case with small chains requires a lot of storage capacity, frequent lookups and fewer processing resources for the chains, while the case with large chains requires less storage capacity, less lookups but a lot of processing resources for the chain generation. If we take a look at the LM rainbow tables they consist of 64 rainbow tables with a total size of 64GB, thus a lot of storage capacity. The results presented above showed that it took much longer time to complete the cryptanalysis attack on the NT hashes than it did on the LM hashes. This is because the larger LM rainbow tables requires a lot of rainbow table lookups as shorter chains are used, and the lookup process is less time consuming than the rainbow chain generation. The NT rainbow tables on the other hand consist of 4 tables with a total size of 8GB, thus less storage requirements but longer chains. And as already mentioned, long chains require more processing resources, which is more time consuming than frequent rainbow tables lockup. This is the reason why it took much longer to complete the cryptanalysis attack on the NT hashes than it did on the LM hashes.

The reason why we got a difference between the estimation and the actual result for each of the hash types at all was because we only used one hash value for the estimation while we used 30 hashes when performing the cracking process in Phase 2. Both the lookup process and the chain generation take longer time when we consider 30 passwords than it takes for just the single password. But why was the difference between the estimate and the actual result so much bigger for the NT hashes than it was for the LM hashes? The main reason is that it is required to generate more chains when considering a lot of passwords than when considering only one password as was the case in Phase 1. This influence both the cracking process of the LM hashes and the NT hashes. But as the rainbow chain generation takes longer time than the lookup process, and the chains are longer for the NT rainbow tables than for the LM rainbow tables, the difference will be greater for the NT hashes than for the LM hashes. In other words, this implies that the LM rainbow tables are more effective than the NT rainbow tables when considering many passwords. It may be added that even though we start out with 30 password hashes, this number will decrease as the cracking process reveals passwords. The cryptanalysis will therefore perform faster as passwords are revealed, but still the chain generation will be time consuming because of the big amount of passwords. The passwords will not be revealed simultaneously even though the cracking process considers all the password hashes in parallel, and not one by one. And because the cracking process is not a sequential process where only one password is considered at the time, you cannot multiply 22 minutes and 38 seconds with 30 passwords. 22 minutes and 38 seconds was the estimated time for the cryptanalysis attack on the NT hash for the single password in Phase 1.

It may be argued that it was also 30 password hashes to consider for the dictionary attacks, but still there was almost no difference between the estimate and the actual result. This is because there is for example no computation of rainbow chains for the dictionary attack, and therefore it will not be much difference whether you use several passwords or just one password.

Recall that we chose to use two rainbow tables when we estimated the time for the cryptanalysis attack on both the LM hash and the NT hash. The reason why we chose to use two rainbow tables in Phase 1 was, as already mentioned, that the completion of the first table would take longer time than the following tables. This is because the rainbow chains with its intermediate values are calculated and then reused for the following rainbow tables. This was learned from our own experience as we tried to estimate with the use of only one table the first time, but then became aware of that the use of the first rainbow table was more time consuming than the others. We still had to use the timing results obtained from the use of only the first table when we estimated how long time the second table would take to complete. These calculations are described earlier, in Section 8.3 in Chapter 8. These calculations showed that for the LM hash it took only about 53 seconds to complete the second rainbow table. When we used the first two LM rainbow tables it took 6 minutes and 32 seconds. This confirms the theory that running the first rainbow table is more time consuming than the second rainbow table.

The unused time for the LM hash cracking was estimated to about 87% in Phase 1, while the actual unused time turned out to be about 72%. For the NT hash the unused time was estimated to about 95%, while in Phase 2 it turned out to be about 44%. It should be noted that even though Figure 8.4 in Section 8.3.1.1 indicates that for the LM hash about 87% of the available time was unused after we had performed both the dictionary attack and the cryptanalysis attack, it should be highlighted that the attacks in Phase 1 were only carried out on *one* password hash. The idea with the 8 hour limit was to reveal as many passwords of the 30 selected passwords as possible. And as already discussed, the cryptanalysis attack for both the LM hashes and the NT hashes took longer time than estimated because that we used more passwords in Phase 2 than we did when performing the estimate in Phase 1. This is the reason why the estimated unused time is larger than the actual unused time for both the LM and the NT hash. The difference is smaller between the estimated unused time and the actual unused time for the LM hash than for the NT hash. This is the result of what have already been discussed, that the NT rainbow tables uses smaller tables but larger chains than the LM rainbow tables. The main idea with Figure 8.4 was to illustrate how much the different techniques might constitute of the available time using only a single password hash, and then be able to compare this result with how much the same techniques constituted when we used all the 30 password hashes in Phase 2. It was not much difference between the dictionary attacks, but quite big difference for the cryptanalysis attacks.

9.2.2 Password Cracking of LM and NT

We have now considered the timing measurements for both the LM hash and the NT hash. Now it is time to consider how many passwords and also which of the 30 passwords that were revealed during the dictionary and the cryptanalysis attacks. The results from the dictionary attack on the *LM hash* in Phase 2, which can be seen in Table 8.9 in Section 8.3.2.1, shows that 9 of the 30 passwords (30%) were revealed. And if we take a closer look at Table 8.11 in the same section, it can be seen that all the 9 revealed passwords belong to the category named *Most Popular*, which is considered the weakest passwords included in this experiment. The only password from this category that the dictionary attack was not able to reveal was *myspace1*. Actually, as can be seen from the *LM Password* column in Figure 8.11 in Section 8.3.2.1, the dictionary attack was able to find the last character, which is the number *1*. This is because *myspace1* consists of 8 characters, and *1* is therefore the only character that the second hash value is generated from. The reason why the rest of the password was not revealed is that *myspace*, which the first hash value is generated from, is a fairly new word because myspace has become popular in recent years. The word *myspace* was therefore not included in the wordlist used in the dictionary attack. And as also can be seen from Figure 8.11 is that the dictionary attack revealed passwords consisting of 6, 7, 8 and 9 characters. It should be noted that for the dictionary attack on the LM hashes the length of the password does not matter as long as the password length is 14 characters or less. This is because passwords that consist of more than 7 characters and less than 15 characters are divided in two segments, as seen in the case with the *myspace1* password. What matters is that the password represented by the specific hash value is either listed in the wordlist that is being used, or that some parts of that password is listed and that the whole password is found by the use of the mangling rules. An example of a mangling rule that is included in this experiment is the *Two numbers Hybrid Brute* which append the numbers ranging from 0 to 99 to every word in the wordlist to see if the hash of it match the hash of the password that is used. The use of this mangling rule can for example be verified from Figure 8.10 in Section 8.3.2.1. And since the LM password may generate two LM hashes, it is no guarantee that both the LM hashes are revealed since both the hashes must be cracked to find the whole password. This is why only the second part of some of the passwords that consist of more than 7 characters in Figure 8.11 is shown. The 21 password hashes (70%) that were not cracked during the dictionary attack became subject to the cryptanalysis attack.

The results from the cryptanalysis attack on the LM hash, which can be seen both in Figure 8.13 and in Table 8.11 in Section 8.3.2.1, show that all the remaining 21 passwords (70%) were revealed during the cryptanalysis attack. The reason why all the remaining passwords were revealed, regardless of which category they belonged to, was that the character set used for the cryptanalysis included the characters that the remaining passwords consisted of. It should be noted that the cryptanalysis attack revealed passwords that consist of 6, 7, 8, 9 and 10 characters. The LM rainbow tables that were used for this cryptanalysis attack supported passwords with length up to and including 7 characters. A complete list of all the supported characters (the charset) is listed in Section 8.2. So why did the cryptanalysis attack reveal passwords of more than 7 characters in length? The reason is the weaknesses when constructing the LM hash. Because passwords that consist of more than 7 characters are divided into two segments with at most 7 characters in each segment, and in that case produce two hashes that represent that single password, it is sufficient to use rainbow tables that covers passwords up to and including 7 characters.

These results show that in this case it did not matter to which category the password belongs because all the passwords were revealed after the completion of both attacks. According to our intention, almost all of the weakest passwords were revealed during the dictionary attack. As this was the purpose of running the dictionary attack, this part of the experiment has to be considered successful. Despite the fact the dictionary attack fulfilled our expectations, it is important to add that in this case it was not really necessary to run the dictionary attack at all since the character set used for the cryptanalysis attack included every character that is possible to employ in the password. It is possible to include so many characters in the character set and still be able to reveal the password in relatively short time because of the weaknesses related to the construction of the LM hash, like that the password is converted to uppercase and split into two segments before it is hashed. This reduces the number of combinations that has to be tried to reveal the password. In other words, this means that the cryptanalysis attack would have revealed all the 30 passwords.

The results from the dictionary attack on the *NT hash* in Phase 2, which can be seen in Table 8.13 in Section 8.3.2.2, shows that 8 of the 30 passwords (27%) were revealed from the dictionary attack. This is one password less than the dictionary attack on the LM hash was able to reveal, which revealed 9 passwords. The password that was not found by the dictionary attack on the NT hash but found by the dictionary attack on the LM hash was *blink182*. The reason for this is again the differences in how the LM hash and the NT hash is constructed. When constructing the LM hash you would get two hash values, where the first hash value is generated from *blink18* and the second value is generated from *2*. The mangling rule named *Two numbers Hybrid Brute*, which is mentioned earlier, will append the numbers ranging from 0 to 99 to every word in the wordlist to see if the LM hash of this variant will match the LM hash value of the password that is used. As an example, when the dictionary attack reach the word *blink* in the wordlist, the LM hash of this word will be constructed and compared to the hash value of the real password. This will not give a match, since the password to be found for the first LM hash value is *blink18*. Then *blink0* will be tried and so it continues until it match is found or until *blink99* is reached, since 99 is the last value supported by this specific mangling rule. In this case we will get a match for *blink18*. As the number *2* exist in the wordlist, the second LM hash value will also be cracked.

When constructing the NT hash, the password is not split into segments, and this is the reason why *blink182* will not be found during the dictionary attack on the NT hash. Since the password is not divided into two segments, the whole *blink182* have to match, and not just *blink18* and then *2*, as was the case for the LM hash. Since the *Two numbers Hybrid Brute* mangling rule only supports numbers up to and including

99, *blink99* will be the last variant of *blink* that is tried before the dictionary attack moves on to test the next word. As a consequence of that, *blink182* will not be found during the dictionary attack on the NT hashes. This highlights that the NT hash is more secure than the LM hash. Since the whole password is used when the NT hash is generated, the whole password must match a word tried by the dictionary attack, and not just parts of it. Other than the *blink182* password, the dictionary attack on the NT hashes found the same passwords as when running the dictionary attack on the LM hash. For the same reasons as for the LM hashes, the *myspace1* password was neither found during the dictionary attack on the NT hashes. The only difference is that while the dictionary attack on the LM hash of the *myspace1* password revealed the last character which is *1*, the dictionary attack on the NT hash of the same password did not reveal anything of the password. This supports what already have been discussed. All the revealed passwords did also in this case belong to the category named *Most Popular*. The remaining 22 passwords (73%) became subject to the cryptanalysis attack.

The results from the cryptanalysis attack on the NT hash, which can both be seen in Figure 8.20 and in Table 8.14 in Section 8.3.2.2, show that 10 passwords (33%) were revealed during the cryptanalysis attack. After the completion of both the dictionary attack and the cryptanalysis attack 18 passwords (60%) were revealed in total. This left us with 12 unrevealed passwords (40%). If we take a closer look at Table 8.14, it can be seen that most of the 10 passwords that were revealed through the cryptanalysis attack on the NT hash belongs to the category named *Mnemonic Passwords*. Actually 6 of the 10 mnemonic passwords were revealed, and in addition also one password from the *Most Popular* category and two passwords from *Random Passwords*. The 4 remaining password from the *Mnemonic Passwords* category that were not revealed is *m.f0vgk* (7 characters), *v*qbt4un* (8 characters), *5GCTw4tG@N* (10 characters) and *tcC!tW84s* (9 characters). The last two are the passwords that were produced by us, and both of them have corresponding phrases that are easy for us to remember even though they consist of many characters. As can be seen from Listing 8.1 in Section 8.2, the maximum password length that are supported by the *Vista special* tables is 8 characters, and the corresponding character set are quite limited as only numbers from 0 to 9 and lowercase letters are supported. As *5GCTw4tG@N* is 10 characters long it was expected that this password was not revealed during the cryptanalysis attack. The *tcC!tW84s* password consists of 9 characters. This is also too long for the *Vista special* tables to crack. But at Ophcrack's web page there are other tables as well, and if we take a look at the rainbow tables that is named *Vista eight*, which is 134.6GB in size, they support passwords that are 8 characters in length. The corresponding character set is: *0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!**. This shows that if the *tcC!tW84s* password had been one character shorter it would be revealed by the *Vista eight* rainbow tables. This shows that it may be wisely to choose a password consisting of more than 8 characters, even though it consists of both lower- and uppercase letters and a special character. The other two passwords (*m.f0vgk* and *v*qbt4un*) that were not revealed by the *Vista special* rainbow tables are both within the supported password length of the *Vista special* tables, but they have different character sets because they have different length. Both of these passwords consist of lowercase letters, one number and one special character. The only thing that prevents both of them from being revealed by the rainbow tables is the special characters. We have now considered the only four passwords from the *Mnemonic Passwords* category that were not revealed by the cryptanalytic attack. This was to highlight that the characters that these passwords consisted of were not included in the character set. All the revealed passwords consist of characters that are included in the character set.

If we take a look at the *Random Passwords* category there was only two passwords (*qaSt4@* and *vANe$a*) that were revealed by the cryptanalysis attack. The reason why these two passwords were revealed is because they both consist of only 6 characters. And if we take a look at the character set for passwords that are 6 characters or less in Listing 8.1, we can see that it covers a lot of characters, including special characters. That is why these two passwords were revealed. If they had been one character longer, they would not have been revealed because the character set for passwords of 7 characters in length do not include special characters. Both of the passwords contain a special character. They are also the only passwords in the *Random Passwords* category that are just 6 characters in length, and the other passwords are not revealed by the cryptanalysis attack.

In the category called *Most Popular* there was also two passwords (*myspace1* and *blink182*) that were revealed by the cryptanalysis attack. The reason why these passwords were not revealed during the dictionary attack has already been explained. As can be seen, both *myspace1* and *blink182* consist of 8 characters, which is the longest password length that is supported by the *Vista special* tables. And from Listing 8.1 it can be seen that the corresponding character set are quite limited. The reason why both these passwords were revealed is because they only consist of lowercase letters in addition to numbers. If for example only one of the characters had been an uppercase letter the passwords would not have been revealed. It has now been shown that it might be small differences that determine if a password is going to be revealed or not.

We have now considered the password length of the revealed passwords from the cryptanalysis attack on the NT hashes, and all the revealed passwords consist of 8 characters or less. This result are according to what could be expected from studying the different character sets for the different password length for the *Vista special* rainbow tables. Based on these results, and by studying the character sets that are available at Ophcrack's web page [44], it seems that for the time being your password should be hard to crack with the rainbow tables at Ophcrack's web pages if you select a password that is more than 8 characters in length, contain both uppercase- and lowercase letters, and also special characters. It seems that numbers are included by *all* the character sets that are used for the different rainbow tables available at Ophcrack's web pages. It should be noted that it is also possible to use a password consisting of for example 8 characters as long as not all the characters that the password consists of is supported by the character set. The password should also belong to the *Mnemonic Password* category as the password may be easier to remember, but still appear as a random string of characters. This is described in more details in Section 2.1.1 in Chapter 2. It should be noted that the password requirements mentioned above might change quickly as the character sets grow larger and also longer passwords are being supported by the character set. At this time it seems to be a trade-off between the number of password characters that are supported, and the number and variety of characters in a character set. This may change as the processors are becoming more powerful and as there are more or less no storage limitations any more.

Earlier in this discussion we claimed that most of the passwords that were revealed by the cryptanalysis on the NT hashes belonged to the *Mnemonic Passwords* category. Even though this is true it needs a closer explanation. Recall that the hash values that were cracked during the dictionary attack were removed from the list before the cryptanalysis attack was performed on remaining hash values. If the passwords that belong to the *Most Popular* category had been included in the cryptanalysis attack, another 5 passwords (*abc123*, *password*, *qwerty1*, *fuckyou*, *123abc*) would probably have been revealed as they have all characters that are included by their corresponding character set. This would have made the total of revealed passwords from the *Most Popular* category as much as 7 passwords, and would then have been the category with the most revealed passwords instead of the *Mnemonic Passwords* category. Another aspect to point out is that the *Mnemonic Password* category, which is said to be the category containing most of the passwords that were revealed during the cryptanalysis attack, is the category with the largest number of passwords that are 8 or less in size. To be exact, 8 of the 10 passwords in the *Mnemonic Passwords* category consist of 8 characters or less. A length of 8 characters is the maximum password length that is supported by the *Vista special* rainbow tables. This shows that uneven distribution of passwords with different lengths among the three password categories have probably influenced this result.

We have now seen that the dictionary attack performed on the NT hashes revealed 8 passwords while the cryptanalysis attack revealed 10 passwords. What can be said about whether it was advantageous to run a dictionary attack on the NT hashes before the cryptanalysis attack or not? Recall that all of the 8 passwords that were revealed during the dictionary attack belonged to the *Most Popular* category. And the purpose with the dictionary attack was to reveal as many as possible of the weakest passwords. As already pointed out, the 2 passwords from the *Most Popular* category that were not revealed during the dictionary attack were revealed during the cryptanalysis. From the discussion above we found out that if the passwords that were revealed during the dictionary attack had been included in the cryptanalysis attack, probably as many as 7 of the 10 passwords that is included in the *Most Popular* category would have been revealed by the cryptanalysis attack. But the biggest advantageous with the dictionary attack, which has not been pointed out yet, is that the dictionary attack revealed three passwords that would not have been revealed if only the cryptanalysis attack had been performed. These passwords are *password1*, *baseball1* and *football1*, and are all 9 characters long. And because the *Vista special* rainbow tables do not support passwords of this length they would not have been found. And if we can spend 53 seconds, which was the time it took to complete the dictionary attack, to reveal three extra passwords, we would say that it was advantageous to run the dictionary attack. In other words, our hybrid solution which consists of both a dictionary attack and a cryptanalysis attack, made sure that all the passwords that are included in the *Most Popular* category were revealed. This would not have been the case if the dictionary attack had been omitted.

Before we move on to the MD5 hashes, it should be highlighted that the same results were obtained when the dictionary attack and the cryptanalysis attack were performed on the SAM file that originated from the Windows 7 RC operating system. This means that also the same discussions applies for the NT hashes stored in the Windows 7 RC SAM file.

9.2.3 Time Measurements and Password Cracking of MD5

The last type of hash value that was used is the MD5 hash. In Phase 1 the dictionary attack on the *MD5 hash* of the selected password (8rA*_pHa$8) was measured to take maximum 51 minutes and 48 seconds, which constitute 10.79% of the available time. This leaves us with 89.21% unused time, but remember that only one password was used. A cryptanalysis attack was not performed for the MD5 hash, and since the dictionary attack was the only attack that was carried out, the idea was to let it run for the whole period of 8 hours. As the maximum time it might take for one password was measured to 51 minutes and 48 seconds, it was expected that we had to abort the dictionary attack manually after 8 hours. Even though we are using 30 passwords in Phase 2, the time estimate for the maximum time it might take for all the 30 passwords will not be 30 * (51 minutes 48 seconds). This is because the dictionary attack considers many passwords in parallel and not the passwords one by one. This is confirmed by Figure 8.23 in Section 8.3.2.3 as the revealed passwords are found in random order. The password related to *user8* is for example the first password that is revealed, while the password related to *user1* is one of the last passwords that are revealed. When we aborted the dictionary attack after 8 hours it had revealed 8 passwords (27%), which can be seen in Table 8.17. As for the dictionary attack on both the LM and the NT hashes, all the revealed passwords during the dictionary attack on the MD5 hashes belonged to the *Most Popular* category. It may be added that this result is the exact same result that we got from the dictionary attack on the NT hashes. In this case, the only two passwords that were not found from the *Most Popular* category were also *myspace1* and *blink182*. These were not found for the same reasons during the dictionary attack on the MD5 hashes as for the case with the NT hashes. The dictionary attack on the LM hashes revealed one more password (*blink182*) compared to the dictionary attack on the NT and MD5 hashes. In other words, this means that the cryptanalysis attack on the MD5 hashes would have had the same basis as the cryptanalysis attack on the NT hashes because 73% of the 30 passwords were still not revealed after the dictionary attack. But as no cryptanalysis attack was carried out on the remaining MD5 hashes, the remaining 22 passwords (73%) stayed unrevealed. It is important to highlight that the attacks performed is attacking the actual password that is hashed, and not the weaknesses of the MD5 algorithm.

It would only be possible to say if the dictionary attack was advantageous to perform in addition to the cryptanalysis attack if the cryptanalysis attack actually had been carried out. But since most of the weakest passwords were revealed, and the result was exactly the same as for the dictionary attack on NT hashes, it is a reason to believe that it would be advantageous to perform the dictionary attack. If a cryptanalysis attack was supposed to be carried out after the dictionary attack, the dictionary attack would of course not be run for 8 hours. Actually this is not necessary as all the revealed passwords were revealed when the dictionary attack had run for 1-2 hours. This can be verified in Figure 8.23. And because our upper limitation for the dictionary attack was set to 2 hours, this would not have been a problem. The remaining hours could have been used for the cryptanalysis attack. As already pointed out, the cryptanalysis attack on the MD5 hashes was omitted due to lack of MD5 rainbow tables and due to time constraints.

9.2.4 Discussion of General Results

When it comes to the results in general, it may be added that the reason why we used the same wordlist for all the dictionary attacks was to make the attacks as equal as possible for the different types of hashes. It would be hard to make a comparison if we used a more comprehensive wordlist for one of the attacks. Then some may argue that the conditions for the cryptanalysis attack were not the same since they used different kinds of rainbow tables. The reason for this is that the different types of hashes are constructed in different ways, and therefore need rainbow tables which are produced for that specific hash type.

It may be discussed that since it was unused time after the dictionary attacks and the cryptanalysis attacks had completed, we may have used a larger wordlist or more comprehensive mangling rules for the dictionary attack, and also a larger character set for the cryptanalysis attacks. This is possible, but we claim that since such a big portion of the passwords were revealed based on the choices made for this particular experiment, the trade-off between cracking time and revealed password was adequate. A small increase of the wordlist, the character set or more comprehensive mangling rules may result in an unexpected large increase in the time it takes to complete the attack. We had taken this into consideration by introducing a limitation of the time available for both the dictionary attack (2 hours) and the cryptanalysis attack (6 hours). The cryptanalysis attack was dedicated most of the time as the dictionary attack was included to reveal the weakest passwords before running the cryptanalysis attack. The chance of revealing the more secure passwords was considered greater when running the cryptanalysis attack as the more secure passwords would not be included in the wordlist used for the dictionary attack. These limitations were set in advance, before we actually knew how long time each of the attacks would take. This was to prevent the attacks from taking too much time, as we wanted to find out how many of the 30 selected passwords we were able to reveal during a working day (8 hours). The results show that both the dictionary attacks and the cryptanalysis attacks were completed or aborted within these limitations.

If we were supposed to compare the use of a dictionary attack with a cryptanalysis attack we would highlight that while the dictionary attack requires a relatively small wordlist, even though there exist different types of wordlist were some of them are bigger than others, the rainbow tables used in a cryptanalysis are a lot bigger. This requires a quite large storage capacity. Another aspect with the rainbow tables is that they must either be downloaded or generated in advance of the attack, and this may take quite a long time depending on your download speed and the system you are using. It is important to be aware that people that have lost their password or attackers with criminal purposes probably would invest a big effort in the action of recovering the password. It should also be mentioned that, as have been seen from the results of the empirical password study, a dictionary attack will probably take shorter time to complete than a cryptanalysis attack. This will of course depend on how comprehensive mangling rules you have included and also on the size of the wordlist and the rainbow tables. But as already mentioned, a hybrid solution like the one used in this study with first running a dictionary attack and then performing a cryptanalysis attack, seems like a good solution to recover passwords.

Another observation is that when it comes to cracking passwords with the use of rainbow tables, it does not matter if the password is strong or weak as long as the password is within the requirements sat for the particular rainbow tables. In other words, if the character set for the rainbow table covers all the characters used in the password, the password will be revealed no matter if it is strong or weak. An example of this is *myspace1*, which is one of the weak passwords used in this study as it is included in the *Most Popular* category. As can be seen from Table 8.11 in Section 8.3.2.1, which shows the results obtained from both the attacks carried out on the LM hashes, the *myspace1* password was revealed during the cryptanalysis attack (as the table entry is not in italic). Then, take a look at Figure 8.15 from the same section which shows some of the passwords revealed during the cryptanalysis attack. The revealed passwords that are shown are listed in the order they were found. As we can see, the *myspace1* (user3) password is not listed in the view shown in the figure. Note that the figure does not show all the revealed passwords. This may be verified by the scroll bar to the right in the figure. The *myspace1* password is actually found as one of the last passwords. From Figure 8.15 you can see that there are a lot of passwords that seem a lot more secure than the *myspace1* password, which has been revealed before the *myspace1* password. An example is the *qaSt4@* (user21) password, which is the first password that is revealed. This password even includes a special character. The idea with this example is to show that to select a secure password the password needs to contain characters that are not included in the character set used for the cryptanalysis attack. And usually you don't know what character set or rainbow tables an attacker use. This makes it harder to choose a secure password, but this will be discussed in later sections.

The results obtained from the empirical password study shows that even though it is important to choose a strong password, this may not be sufficient alone. The underlying system does also influence how easy the password can be recovered. A good example of this is Windows XP, which utilizes the LM hash together with the NT hash. Because of the weaknesses with the construction of the LM hash, which is already pointed out several times, the rainbow tables can support a character set consisting of all the characters necessary to reveal all the passwords. The results obtained from our study, showing that all the 30 selected passwords may be revealed using rainbow tables, confirms this statement. It may also be confirmed by investigating the character set as well. To avoid the case illustrated in this example you should either disable the use of LM hash or you should use a password longer than 14 characters if you are using Windows XP. When the password is 15 characters or longer an operating system that uses both LM hash and NT hash will not store the LM hash, only the NT hash. One way to disable the use of LM hash is given in Appendix C.

This study shows that a big portion of the selected passwords were revealed, even with computers that are not particularly powerful. As computers are rapidly getting more powerful, with more storage capacity and processor power, a lot of passwords will probably be revealed in the future. This increase in computer power will lead to that more computations can be done in a shorter amount of time. Even though the use of salt in the generation of the password hash will probably offer sufficient protection from password recovery for a while, this will not offer sufficient protection in the future because of the aspects pointed out above. As a result of this, the use of password will probably not be a good authentication mechanism in future systems and other mechanisms may have to be considered.

9.3 The YALP Tool

When we performed the experiments presented in Chapter 5 and Chapter 6, we experienced that some of the procedures and tools were hard to carry out. As a reaction to this we decided that a tool realizing some of the procedures was desirable. We also urged for a tool that could handle many different operating systems, was able to both reset and recover the login password, and that was easy to use. There are many different password handling solutions residing on the Internet, and there are multiple live CDs available offering a lot of tools, not just password handling tools. We could not find any live CD that had a primary focus of collecting password resetting and password recovery tools. We found a lot of security distributions containing a lot of security tools, but none of them were plain and simple, and they did not have the main focus on password handling. Because of this, we decided that a tool which could fulfill our demands was desirable. Therefore we ended up with creating a tool called *YALP* (Yet Another Local Password (tool)).

YALP is embedded with some of the most used password resetting and password recovery tools. This tool, or more precisely security distribution, also includes our own tools/scripts realizing some of the procedures described in Chapter 5. The tool serve as a good alternative to other security live CDs/distributions because it supports many different file systems, is easily configurable, does not require any specific hardware and is small and efficient. Although YALP is small it is build upon a basic Linux system, originating from the very popular Linux distribution *Debian*. Because YALP is based on Debian, it will support the same packages and software meant for Debian. The underlying system also has the same basic tools as many other Linux distributions such as Fedora [92] and Ubuntu [9], making YALP very easy to use. The menus and scripts that we have included in YALP ease the use of some of the often unfamiliar tools included in most Linux distributions. The script *mountos.sh* which automates the mounting process, and is presented in Appendix B.4 is a good example. A normal user might be grateful when it comes to having a script that automates this process. The mounting process of a host system is usually necessary to perform, because files have to be changed on the host system. We have chosen to initiate the mounting script automatically whenever the user needs to mount the host system, if the host system not already is mounted. One of the advantages with the scripts we have created is that the user needs no insight in how to accomplish the procedures that is implemented.

Procedure 2 (Edit Shadow File in Unix Systems) is an example of a procedure to reset the password in Ubuntu Server 8.10, which is pretty difficult to carry out. The procedure has some prerequisites demanding that the user has knowledge of how to create the correct hash value (MD5) for the new password to use. One of the scripts we have included in YALP, *editshadow.pl*, automates this process for the user, and only interfere with the user when necessary, for instance prompting the user for the desired password. The *editshadow.pl* script is presented in Appendix B.3. This script and all the other scripts included in YALP need improvements, and for instance support for other systems and hash algorithms. This is left for further work and is therefore mentioned in Section 9.8.3.

None of the tools included in YALP cost money, but some are nevertheless considered as the best tools available to recover and reset passwords.

9.4 Implications of Poor Login Security

A poor login security can lead to many unwanted situations. This thesis demonstrates that most operating systems have by default a poor login security, either through backdoors or through weak password handling. Enforcing the user to choose a secure password might be crucial to avoid the passwords from being revealed by unwanted persons. Yet systems need to have some sort of recovery mechanism so that authorized persons can regain access to their data. One may say that the purpose of a login mechanism is lost when it can easily be circumvented. Why should a person have to remember a difficult password when the underlying login security is poor? A common expression is that when the computer is lost, so is the files inside the computer. This is not a good excuse to have a poor login security. It should be the case that if the computer is lost or stolen, nobody should easily able to steal the data residing inside the computer. At least, the login security should stop the unwanted person when the login screen appears. An unwanted person should not be able to use the system as an implication of a poor login security.

It should not be the case that a person can quickly bypass the login mechanism on every computer he or she is presented with. This makes the whole purpose of having a login mechanism unnecessary. Some login mechanisms can be circumvented in less than a minute, as presented in Section 7.1. This makes leaving your computer unattended for only a short period of time a bad choice.

So what are then the implications of a poor login security? Although the list of implications is endless, we have chosen to present some of the implications to highlight the importance of a good login security. In this section we have chosen to separate between the implications of a weak login mechanism and the implications of a bad choice of password. The reason why we separate between these two implications is because the login mechanism is something a user usually trusts and is unable to change, while the choice of password is something a user can affect.

In the next sections we present some implications that arise when a weak login mechanism is used or a weak login password is chosen. A combination of these will of course be even worse.

9.4.1 Implications of a Weak Login Mechanism

Having a weak login mechanism on a system can make the process of resetting or recovering the password an easy task. This will then lead to unwanted persons being able to easily reveal the password. If a system gets compromised, the attacker can harm not only the owner or user of the system, but also an entire enterprise. A step into an enterprise might be through an employee's computer. This step can lead to that new networks become available and new barriers can be exploited. Important personal information or information vital for an enterprise can be stolen, often leading to large costs for the enterprise. Loss of personal information might even damage a person's reputation or life. Some people might say that their computer do not contain important information, but an attacker can easily insert unwanted information on your computer that may harm you in the future. This unwanted information can for instance be child pornography or just a virus waiting for the user to perform an important task on his computer. If your computer is a subject to a malicious code attack, your computer can be used in all sorts of criminal and harmful ways.

A weak login mechanism can lead the user of the system into believing that since the chosen password is secure, so is the login mechanism. For instance as we experienced with Windows XP, a usually secure password containing 10 characters is not secure since the underlying login mechanism is not secure. Then again one may wonder why windows have chosen to enhance their security with a security mechanism that is proven to be insecure. Microsoft's choice of replacing the weak LM hash with the NT hash that is based on MD4, comes as a big surprise to the authors of this thesis, as the MD4 algorithm is proven insecure. A more scary thought is that the new Windows operating system Windows RC 7 still uses the same weak NT hash algorithm. Windows also seems to ignore the fact that salt is a good prevention of cryptanalysis attacks with rainbow tables. In Windows 7 RC we do not see any indications of that the hash value protected with a salt.

9.4.2 Implications of a Weak Password

By choosing a weak password, it can easily be revealed through password cracking or password guessing. If the password ends up in wrong hands, the same unwanted events as described in previous chapter can happen. What is even worse with revealing a password to an unwanted person is that a clever attacker knows that it is very common to use the same password, or nearly the same password, other places. The attacker then has access to other services, and harm can be done on these places as well. If the attacker manage to reveal your password, he might come unnoticed from the attack, leaving the compromised machine and user unaware of what have happened. The attacker can then strike his attack in a later moment without the user's knowledge.

As mentioned before, choosing a strong or secure password is very hard, because it is dependent on the underlying password mechanism. The user generally needs to know how the system the password is going to be used on is working, in order to decide the strength of the password. This substantiates the importance of a good and secure underlying password handling mechanism.

It is important to choose a strong/secure password. As we identified in the experiments presented in Chapter 8, systems that has a good enough password handling mechanism is also dependent on a strong/secure password to not be compromised. It is not sufficient enough to rely on the underlying password mechanisms, because most attackers try common passwords, dictionary attack with mangling rules (described in Section 2.1.2.2), before eventually a brute-force attack is performed. The brute-force attack might take very long time if the set of characters is large, but the password will probably be revealed if the password originates from a dictionary. If this is the case and the attacker is in possession of the password file, a good password hash algorithm will not stop the attacker from revealing the password.

9.5 Possible Solutions to Prevent Unauthorized Access

In this thesis, we have identified many procedures and tools that can give authorized persons access to the system even though the login password is lost. The problem with these procedures and tools is that unwanted persons might also get access to the system. A good system is capable of providing access to authorized persons, while unauthorized persons are denied access. An even better system is able to help and provide alternatives to authorized persons, but still deny access to unwanted persons, if the password is lost. In this section we present some possible solutions to prevent access for unauthorized persons.

9.5.1 Choosing a Secure Password

A good and secure password is often referred to as a word or phrase which is easy to remember yet hard to guess. With tons of password guessing tools residing on the Internet, creating a secure password becomes more and more difficult. During the experiments with passwords in our empirical password study, described in Chapter 8, we experienced that the importance of choosing a secure password will highly depend on the system the password is created on. It is very sad to say that in some systems, for instance in Windows XP (described in Section 8.3.2.1), choosing a secure password will not largely impact the overall process of guessing the password, because the underlying password handling can be exploited. This means that the user has to know how secure the underlying system is, to choose password that is sufficiently secure. We experienced that for Windows XP the password is not secure unless it is 15 characters or longer, resulting in the system using a more secure password handling mechanism (NT hashing), not storing the password as a LM hash. This is of course not the case if LM hash is disabled, something that is not done by default. By not doing this by default, the ignorant user will not be aware of the security risk when using Windows XP. On the other hand, if a user for instance uses Fedora 10 the passwords are stored in a more secure way, and secure passwords do not have to be longer than for instance 10 characters.

We also experienced that if the password handling used by the underlying system is weak, such as with Windows XP and the LM hash, a password that seems stronger that another may not be more secure after all, because all passwords with a length less than 15 characters will be found anyway. If the password is subject to a cryptanalytic attack using rainbow tables, and the characters that the password consists of resides in the *charset*, the password will be revealed regardless of its strength.

If you disregard the fact that a password may be secure on one system and not on the other, there are some basic rules on how to produce secure passwords. The rules/properties mentioned in Section 2.1.1 presented by Klein [Kle91] and AccessData [13] will make the password harder to guess when a dictionary attack is performed. To protect against cryptographic attacks the password handling mechanism has to be sufficiently secure and the password has to be of a certain length.

In Section 2.1.1 at page 9, we describe various ways to create secure passwords. The password has to appear be as random as possible, and the length of the password has to be above a certain length. The problem with randomness is that it makes the password harder to remember, and of course the creation of a random password with the correct properties is a study in itself. Some security experts advice the user to write down the password, but we mean that this is not a good advice. The user can then easily lose their password, the attacker might be able to steal the password, or the user may become a subject to social engineering.

To remember the password it is suggested to choose a random password which has a tone that is easy to remember. It can be hard to come up with such passwords, so another method is using so called mnemonic phrases. They are built in a way that makes the corresponding passwords easy to remember but also looks random. This makes it easier to remember password that is very long.

We suggest to change the password as often as possible, where the new password does not bear any resemblance to the old. This is can stop the attacker if the password is lost or gets stolen. Using the same password for many services is not a good idea, because some services might have weaknesses in the security.

What we realized from the studies was that a secure password will not necessarily stop successful attempts to reset the password; it will only hinder the attacker from revealing the password. To stop an attacker from resetting the password, additional security in the password handling must be enabled. A solution to this might be to encrypt the password files in such a way that the attacker will fail to change the password hashes. An integrity check to see if the password files have been modified after boot might also be desirable.

In our opinion, generating a password that is based on a mnemonic phrase, has a length longer than 14 characters and without any of the weak properties mentioned earlier, is the only option to get a secure password. This is of course without changing the built in password handling. This password has to be changed regularly and has to be created without any connection with the owner of the password or famous phrases residing on the Internet. A password of 14 characters in length can of course be reduced to 10 characters if the LM hash is not used. As far as we know, no rainbow tables have so far been created to crack hashes containing passwords of this length, which is not hashed with the LM algorithm.

This, may not yield in the future as new weaknesses are revealed all the time and computers are becoming more powerful. As we speak, distributed generation of rainbow tables take place on web pages such as [106]. This page also claims to be able to perform distributed cryptanalysis attack using rainbow tables.

As discussed in the beginning of Section 9.5 a system should be able to provide a password resetting mechanism in such a way that only authorized persons can reset their lost password. Microsoft's solution to this, which is described in Procedure 6, is very good. We think that this solution should be an enforced requirement when the user is installing the operating system. The password reset disk must of course be kept in safe place, so that unauthorized persons are unable to steal this together with the computer.

9.5.2 A More Secure Password Handling

In this thesis we have given approaches for accessing a password protected system even though the password is lost. Many of the procedures and tools used in these approaches utilize the weaknesses in a password handling mechanism in an operating system. Some of the weaknesses were already identified in the background chapter, and some were identified when different resetting and recovery approaches were carried out. Some weaknesses tend to be fixed as new versions of an operating system is produced, but still remain unchanged in older operating systems, even though these systems are still being used all over the world. Many people and enterprises avoid updating their computers, because of the large readjustments to the new system. The new systems have often not been tested to be stable enough, or lack the required backward compatibility. One may also wonder why some systems seem to ignore well known weaknesses year after year.

In Windows XP, which is still widely used by enterprises and end-users, we identified multiple weaknesses with the password handling. Windows XP implemented backward compatibility with earlier Windows systems by including the LM hash. In Section 2.3.1.2 we presented these weaknesses. Many of the tools that we described earlier, such as *John the Ripper* and *Cain and Abel*, utilize these weaknesses by adding support for cracking this hash. Even though the password additionally is stored as an NT hash, the wise attacker knows that attacking the weakest link is the most effective, thus the attacker focus on the LM hash. A solution to Windows XP's backward compatibility may be to disable the LM hash by default, and rather compute it if needed. An improvement to the weak LM hash came with the NT hash. In Windows Vista, the LM hash is not enabled by default, leading to the procedures and tools that attack the LM hash not longer being feasible to use. In both Windows Vista and Windows 7 RC, LM hash can be enabled by setting the register value *NoLmHash* from 1 to 0. In our opinion, the possibility to manually enable the LM hash should not be available. The results we got in Section 8.3 are proof enough that the LM hash should not be used. If it has to be used, this is a process we mean the operating system should handle without the intervention from the user. The operating system could for instance temporarily generate the LM hash when needed, and then delete it afterwards. Some may say that making it impossible for the user to activate the LM hash is too strict, and letting users have the possibility to make changes is one of the strengths with systems such as Linux. We agree that it is good for users to have options, but because the LM hash is tremendously bad and users may be given false expectations of security, we rather support that users are enforced not to use LM hash in their systems. We would also advice them to upgrade their existing operating systems, so that they only support the more secure NT hash.

Another discussion may arise with the statements that NT hash is secure enough. It is very strange that even in the newest systems of Windows operating systems the same NT hash, which is generated by the use of the proven insecure MD4 hash algorithm, is being used. In our opinion Microsoft should update the NT hash's underlying hash algorithm to a more secure hash algorithm. This hash algorithm should always be kept up to date, and replaced if a better algorithm is introduced. A reason why Microsoft neglects preforming these steps, might be because the MD4 hash algorithm is a very fast hash algorithm.

The lack of using salt when generating the LM and NT hash is in our opinion the worst weakness in the password handling mechanism used by all of the different Windows systems. We suggest that Microsoft update their hash generation to use salt. This would hinder many of the tools presented in this thesis from revealing the passwords. A good example is presented in Chapter 8. Both the LM and NT hash were attacked with rainbow tables because of the lack of salt and many passwords were revealed. The MD5 hash was attacked only with a dictionary attack because salt had been used, thus only about one third of the passwords were revealed. The reason why the passwords were revealed in this attack was because the passwords suffered from many weaknesses. If the LM and NT hashes had been generated with the use of salt a cryptanalytic attack would have been difficult to perform, and much less passwords would probably have been revealed. It is of course a necessity that the salt is random and of a certain length, making the set of rainbow tables to generate much larger. As mentioned earlier in this thesis, in possession of a supercomputer or a large distributed network of computers, generating the rainbow tables after the salt is known might be possible.

Some of the Unix-based operating systems we experimented with, such as Ubuntu Server 8.10 and FreeBSD, use the MD5 hash algorithm together with the crypt() function to produce some of their hash values for the passwords. MD5 is also proven to be weak [WY05], but these weaknesses were never exploited by us in the experiments presented in this thesis. It is however advisable to use a hash algorithm with no proven weaknesses to hide the passwords.

When it comes to the password resetting approaches presented in this thesis, we also presented weaknesses with the password handling mechanism that could be utilized to perform password resetting. An obfuscation of the file(s) containing the hash values of the passwords is desirable. The obfuscation should not be easily reversible in such a way that the user could read the password file unless he is authorized. One of the more positive things about the password handling mechanism provided in Windows systems, is that they at least *try* to obfuscate the hashes in the SAM file. This obfuscation is easily reversible with multiple tools available on the Internet. Tool 7 (Ophcrack) is one of many tools mentioned in this thesis that has this ability. In the Unix-based systems presented in this thesis, no obfuscation of the password files is present. The password files are only readable by root, or users with Administrator privileges. The files are also easily accessed using a live CD or through single-user mode that often requires no authentication. Introducing a good enough obfuscation of the password files, could prevent the procedures and tools that directly edit the password files from succeeding. Password resetting would then be hard to perform and the hash values of the password hard to crack. The unauthorized user would then first have to crack the obfuscation of the password file. This would prevent many of the simple and fast resetting procedures and tools. Confidentiality and integrity would be desirable to provide for the password files, for instance through encryption and integrity checks. As this would require a password, a Trusted Platform Module (TPM) could be a good solution. If this hardware device had the property of being unreadable and not writable by others than the operating system, the password handling mechanism would be much more secure. In the newest computers released by Apple such a device is included, although Mac OS X 10.5.6 does not encrypt the hash file. Using a TPM would be much wiser than hiding the password file, like Microsoft has done in their systems. More about this can be found in Section 2.3.1.2 at page 39.

Many systems could improve the process where a user chooses a password, either during the installation, when a password is changed or when a new user is added. First of all the password should not be accepted until all requirements for a secure password is fulfilled. The operating system should not enforce a simple “password hint”. The question has to be hard for the attacker to answer, thus not a question such as *What is your mother’s maiden name?*, or *What’s the name of your first pet?* By having such an easy question, as Schneier claims in [107], this will result in the normal security protocol (passwords) falling back to a much less secure protocol (secret question).

As mentioned above it is desired that access to the system can be recovered by an authorized person, yet prevented for an unauthorized person. The authorized user should in our opinion only have one solution to reset or recover the password instead of many different solutions that all can be exploited. Throughout this thesis we have identified many procedures and tools for various operating systems that can be used by both authorized and unauthorized persons.

We desire one solution that has to be as secure as possible at all times. If a fix to a weakness is published, no unauthorized persons should have time to exploit the weakness because an enforced update has patched the system. The solution has to be simple to use, and should provide sufficient protection. This solution should of course prevent the use of all procedures and tools presented in this thesis. A possible solution could be to enforce the user to create one, and only one, password reset disk during installation. This is a solution that is already implemented by Microsoft (Procedure 6), but the solution is only optional. The password reset disk should then be kept in a safe place, not together with the system. If the user loses his/her password, only this disk can recover the password. The user has to be made aware of the importance of this disk, and that if the disk is lost a new disk should be created immediately. Because only one disk is prevailing, the old disk is then useless. It is important to make the user aware of that if both the password reset disk and the password are lost, the system becomes useless. It is of course important that the password reset disk only works for the system it was created on. This means that if a user has two identical systems with the same username, a password reset disk will only work on the system where it was created.

9.5.3 Protection Mechanisms

To prevent unauthorized access to a system, it is common to use a login mechanism where you type in a username and a login password. In the two latter sections we have indicated that choosing a secure password and using a system with a secure password protection mechanism can prevent unauthorized access to the system. In this thesis we have presented procedures and tools to circumvent the login procedure and obtain full access to the system. When the login procedure has been circumvented, we have even presented ways to reset the password with system built-in tools. In all of the Unix-based systems we have experimented with we have accomplished full access through single-user mode. From single-user mode we have been able to reset the password manually by editing the password files, or through the build-in Unix tool *passwd* (Tool 6 at page 108). In our opinion it seems very insecure to let anyone that has physical access to the computer have the possibility to boot into single-user mode. No authentication is needed to get full administrator access. We think that booting into single-user mode should be disabled, thus making it harder to get access to the system without the use of any prerequisites.

In Windows systems we have also demonstrated ways to accomplish full access to the system and then be able to edit the password files using various tools. In all of the Windows systems we have managed to replace an executable file, which we were possible to execute before logging into the system, giving us full System access. Procedure 4 presents an approach to replace the *sethc.exe* file. A minimum requirement should be that the files that possible to execute before a user has logged in, have been integrity checked. By performing an integrity check on these files, they cannot easily be customized to provide full access for unauthorized persons.

Another solution to the problems presented above was to enable encryption of the entire file system. This would probably prevent unauthorized access to the password files as well, thus preventing resetting or recovery attacks on these password files from being carried out successfully. In this thesis we have left encryption of files, partitions or even entire disks to further work. To prevent unauthorized persons from accessing the system, we suggest that the operating system enables encryption by default, as many people ignore messages to enable features, thinking that the login mechanism will stop an unwanted person.

One of the tools, *Kon-Boot*, presented at page 50, exploit a weakness in the kernel, and a "password-bypass-backdoor" is opened. To prevent such attacks, it is important that the kernel becomes patched and updated in such a way that the backdoor is closed. We could not get *Kon-Boot* to work on Windows 7 RC. There can be many different reasons to this; one of them may be that Microsoft has patched the weakness *Kon-Boot* is trying to exploit.

A protection mechanism that might be suggested is a login procedure loaded as part of the BIOS or for instance the boot loader GRUB. Schemes for this are already available on the Internet. As far as we know, most of the schemes can easily be circumvented, and have therefore not been proposed as good solutions to prevent wrong persons from getting access to the system. Authentication schemes concerning BIOS and GRUB is considered as out of scope for this thesis, thus left to further work.

A possible solution to authenticate users, which is not focused on in this thesis, is the use of a domain login scheme implemented with for instance FreeTDS [108]. Another popular technology developed by Microsoft, Active Directory, also offer this solution. The passwords are then securely stored on a centralized server. No local access will reveal any password files or password hash values. Biometrics can also be implemented to be used instead of password to authenticate users.

9.6 Windows versus Unix

When it comes to login security, some of the operating systems presented in this thesis may appear better than others. Some systems might have a security feature that another system lacks, and vice versa. This could be because they have focused on different security aspects. In this section we will put the two main families of systems, Windows and Unix, up against each other.

When we looked at login security we focused on two aspects concerning password handling, password resetting and password recovery. The goal was to get access to the system, not only the system files, but also to be logged in to the system as an Administrator. As we have discussed before, most of the password resetting procedures and tools, and the password recovery tools can be utilized by both authorized and unauthorized persons. In this section we will try to compare the login security of Windows operating systems and Unix-based operating systems. We will take into consideration how many of the procedures and tools that were successfully carried out on each of these systems.

Based on our experience, resetting was easier to carry out on all of the Unix-based operating systems. There are two reasons for this. First of all, to enter single-user mode none of the Unix-based system had any prerequisites, such as a live CD or rainbow tables. As described in Procedure 7, 9 and 9, just by editing some boot-parameters or pressing some buttons at boot, full file system access was granted. The password files were then editable and Tool 6 (*passwd*) could be used to reset the password. The other reason why most of the passwords from a Unix-based system were easy to reset, was that the password files were not obfuscated. In principle the password files are always only readable by root, but when we entered single-user mode root access was granted. Root access to the files was also easily granted through a live CD. As we demonstrated with the script *editshadow.pl*, presented in Appendix B.3, which was an implementation of Procedure 2, the password was reset by editing the password files. With Windows on the other hand, the password file is obfuscated. Even though the obfuscated file can be reversed by decrypting it with the *SYSKEY* as described in 2.3.1.2, this file is not easily editable unless you use a tool. Tool 1 (PCLoginNow) and Tool 2 (Offline NT Password & Registry Editor) were able to reset passwords in Windows operating systems by uncovering the obfuscated *SAM* file after the *SYSKEY* was revealed.

When it comes to password recovery, all of the Unix-based systems used salt when generating the hash values. This is a very important step, because the cryptanalytic attack using rainbow tables is harder to accomplish when the hash is created this way. Separate rainbow tables for each salt then have to be produced. In all of the Windows systems salt was not used when generating hash values. We experienced that many passwords were revealed as a consequence of the lack of this security feature. A very weak hash algorithm was also included in Windows XP. Even though the theory of a trade-off in time and memory using tables generated in advance has been known since 1980 [Hel80], Microsoft released their new operating system Windows 7 RC without the use of salt.

A combination of the positive things with both Unix-based systems and Windows system would be the best solution. It is hard to say which one is better than the other as both had their weaknesses. We have found more procedures and tools for Windows systems, than with Unix-based systems, but this could be the case because Windows systems are much more popular as desktop and laptop operating systems than any of the other Unix-based systems presented in this thesis.

An important fact is that none of the systems were bulletproof and we even found a tool, Tool 4 (Kon-Boot), which worked on both Windows and Unix-based operating systems. it is hard to point out which of the operating systems presented in this thesis that are the best, but by following the advices we gave in Section 9.5, a lot of security weaknesses can be fixed.

9.7 Lessons Learned

The intention with this section is to point out some lessons learned and some key aspects that might have been carried out in a different manner.

9.7.1 Laboratory Equipment

It became clear that the computers used in our laboratory environment were not as powerful as we might have wanted. During the empirical password study it would have been advantageous to have more powerful computers available, and this could probably have enhanced the results obtained from the study. This may have led to that more comprehensive attacks could have been carried out, for example with more mangling rules for the dictionary attacks, and also that the current attacks could have been completed in a less amount of time.

9.7.2 Selection of passwords

When it comes to the selection of passwords for our empirical password study, this may have been done differently. We selected 30 passwords that were distributed among three distinct categories. The three categories were *Most Popular*, *Mnemonic Passwords* and *Random Passwords*. The *Mnemonic Passwords* were obtained by using a generator on the Internet, and the generator produced both mnemonic passwords and their corresponding mnemonic password phrase. Instead of using a generator, we could have asked different people to come up with a mnemonic password that we could have used in the study. The passwords should then have fulfilled certain requirements in order to make the passwords vary in strength. Another option could be to increase the total number of passwords included in the empirical password study.

9.7.3 Time measurements

As already pointed out, the amount of unused time in the empirical password study turned out to constitute a big portion of the available time. As a result of this, we could have decreased the amount of time that we wanted to use on the attacks. Another possible solution could be to expand the wordlist for the dictionary attack, or to use a bigger character set and/or rainbow tables when running the cryptanalysis attack on the NT hashes. Note that it would not be any point in extending the character set for the LM hash as all possible characters a password may contain was already included.

9.8 Further Work

In this section we suggest some topics that can be further investigated. These topics are either out of scope of this thesis or omitted due to time constraints.

9.8.1 Disk Encryption

As a proactive solution to unauthorized access, many corporations use disk encryption. This can protect vital information of an organization from being stolen, if for instance your computer is lost.

Many of the systems used in the experiments have a possibility to enable encryption using different solutions. There exist both encryption solutions that are already embedded in the operating system and third-party solutions. The third-party solutions can operate beneath the operating system, and some can be used on top of the operating system, initiated from within the system.

For Windows operating systems, EFS and BitLocker are two possible encryption solutions. According to Microsoft, BitLocker protects both personal and system files on the drive that Windows is installed on. This will prevent files stored on your computer from being stolen if unauthorized users try to access the computer. EFS on the other hand, is used to help protect individual files on any drive on a per-user basis [109]. BitLocker works before Windows boots up and seamlessly operates beneath the operating system, while the EFS mechanism works after Windows has started [110].

A feature included in Ubuntu 8.10 Desktop and Server edition is the encrypted filestore feature. This filestore is mounted in the *Private* folder in the */home* directory and is automatically unlocked and locked as the user logs in and out respectively [111].

Some of the other systems we have experimented with may also have embedded encryption solutions available. There exist many other third-party encryption solutions such as TrueCrypt [101] and SafeGuard Easy [112].

A possible study could be to investigate whether it is feasible to reset or recover the Administrator password when encryption is enabled, and if successfully carried out, check if files are readable. It could be the case that only some of the files become readable.

9.8.2 Use of Supercomputers

When we carried out the Empirical Password Study described in Chapter 8, we identified that the use of supercomputers could enhance the speed of the password cracking process. A possible study could therefore be to use one or more supercomputers to perform the attacks presented in the study. If for instance the workload could be distributed among several supercomputers this could enhance the results. In a study performed by Bengtson [Ben07], a scheme using password crackers on several computers in parallel is suggested. Testing the passwords in our study using the clustering technique that Bengtson presents, could be desirable.

Originally we wanted to generate our own rainbow tables by using the supercomputer available at The Norwegian University of Science and Technology (NTNU), but due to time constraints this idea have led to a possible master thesis next year instead. By using *rtgen*, which is mentioned in Section 2.4.2.1, or *Winrtgen* (Tool 12), large rainbow tables for many different hash algorithms can be generated. Based on these ideas, a poster named *Generation of Rainbow Tables* was produced. The poster was presented at The 8th Annual Meeting on High Performance Computing and Infrastructure, NOTUR2009, in Trondheim, Norway [2]. The poster is included in Appendix F.

In this thesis we omitted the use of MD5 rainbow tables due to the lack of rainbow tables and due to time constraints. The generation of or the use of MD5 rainbow tables could therefore be a topic to investigate further. Note that for the MD5 rainbow tables the use of salt has to be considered.

Creating rainbow tables and performing cryptanalysis simultaneously might also be a possible solution. When for instance a hash value and salt is known, the rainbow tables corresponding to the actual hash value and salt could be generated on the fly.

If there is not enough time available to generate the rainbow tables on the supercomputer, only a benchmark giving an estimate for how long time the desired process would take, could be desirable. Due to time restrictions given for the use of a supercomputer, the generation may not be possible to carry out.

9.8.3 YALP

The tool YALP needs a lot of improvements. New scripts could be generated for other procedures, and also our existing scripts can be extended to include more features, such as support for more operating systems. It would be beneficial to for instance expand the *editshadow.pl* script to support hash values created by other hash algorithms than the ones already included. In addition, other already existing tools could be added.

A graphical interface for YALP will improve the user experience. People that are not familiar with text-based interfaces will benefit of a graphical interface that will help them understand the features of YALP, and ease the usage of the tool.

The instructions on how to built YALP is included in Appendix A. For an advanced user these instructions can easily be modified to support more features, or to customize the tool in a preferred way. The scripts we created for YALP are included in Appendix B.

9.8.4 Other Hardware and Operating Systems

Although we have made a selection among different operating systems, the presented procedures and tools might also work for other operating systems. It could be desirable to test whether these procedures and tools work on systems running with other hardware specification as well. It cannot be guaranteed that the procedures and tools work on systems with other hardware specifications and operating systems than the ones included in this thesis.

9.8.5 Other Exploitable Weaknesses

Weaknesses other than the ones that are already pointed out in this thesis may also exist. Also these weaknesses could probably be exploited. One example is the Windows memory management. Windows operating systems' memory management leaves data all over the memory in the normal course of operations. When you for example enter your password at the login screen, it will be stored somewhere in memory in some form. According to Bruce Schneier, Windows then swaps the page out to disk, and it becomes the tail end of some file. The data is then moved to some far out portion of the hard disk, and remain there. It is said that Linux and Mac OS do not offer any better solutions regarding this problem [12].

If encryption of the hard disk is enabled, there already exist methods that tries to obtain the password used for the encryption. One such method is to cool down the memory to try to get hold of the password that is already in memory for some time after the computer is shut down.

Another interesting study could be to investigate where and how the password hint, which is required when the user creates a password, is stored on the computer.

9.8.6 Other Authentication Mechanisms

In this thesis experiments on passwords and login mechanisms have been evaluated. It has been revealed that passwords may not be sufficient enough for authentication purposes in the future, as computers become more and more powerful. As further work it could be desirable to look closer into other authentication mechanisms, such as using fingerprint scanners or mobile phones. It might be interesting to implement new mechanisms and then evaluate what kind of recovery solutions they can provide. An evaluation of other already existing authentication mechanisms and their recovery solutions might also be interesting to look further into.

9.8.7 Resetting and Recovery of Passwords Over Networks

This thesis has focused on how to reset or recover a password when a person has physical access to a computer. Further studies can be done on how to reset or recover a password over a network. Some network resetting and recovery solutions exist, such as services that require password hints. In these kinds of services the password is either directly reset or recovered through mail when the correct answer to the password hint is given. In this thesis the password hint has been described as a feature that degrades the security level of the system. A secure network resetting or recovery solution should therefore be investigated.

Chapter 10

Conclusion

This thesis concentrates on how to reset or recover an administrator password on various operating systems. It should be highlighted that even though this topic may be considered as a task with malicious intentions, this thesis has nothing but good intention. The intention is to help authorized users who have lost their password get access to the current operating system. All of the procedures and tools described in this thesis require physical access to the computer containing the operating system where the password is lost. Some may say that when the security of a system is defined, the first to be considered is the physical security of your computer. In other words, this means that if an unauthorized person acquires physical access to your system, the files on the system are characterized as lost. These statements should not be an excuse to have a poor login security.

The first part of this thesis had the purpose of giving a basic understanding of the login mechanism and password handling of the selected operating systems. This highlights existing weaknesses of the login mechanism, and may motivate the operating system vendors to improve the password handling security in their systems. The second part gives a comprehensive step-by-step description of various existing procedures and tools. These can be used by a forgetful person to reset or recover the lost Administrator password. The last part presents an empirical password study that was carried out to test the strength of 30 different passwords.

In this thesis 6 procedures and 10 tools used to reset or recover the Administrator password were presented. All of the procedures and tools had at least one successful attempt to reset or recover an administrator password. The obtained results show that there exist several ways to both reset and recover an administrator password. It should be added that the procedures and tools had different properties. Some of the procedures and tools supported families of operating systems (for example Windows), some supported multiple operating systems (but not all in the family), and others had only support for one operating system. One of our experiences was that the procedures and tools varied in their degree of difficulty, and that some were more time-consuming than others. It quickly became clear that tools were needed to automate some of the procedures that were hard to carry out, thus making them easier to accomplish. Based on these conclusions, the authors of this thesis decided to create a tool called *Yet Another Local Password (tool) (YALP).* The tool is a collection of implemented procedures, and already public available tools. Our experience is that with the use of our tool the procedures could be completed in a lesser amount of time, and also with less effort. In other words, YALP has quite good characteristics when it comes to quickness and simplicity.

It is hard to tell which procedure and tool to use because they have different features, and do not necessarily work on the same operating systems. Despite this, there are procedures and tools that were favored by the authors as a result of their versatility, quickness and simplicity. One of these is the Kon-Boot tool which works on both Windows and Unix systems, and requires almost no intervention from the user. Procedure 4, which replaces a program that can be initiated before login, has a nice feature that works on all the Windows operating systems used in this thesis, including the new Windows 7 RC operating system. An implemented version of this procedure is contained within YALP.

The empirical password study was carried out to emphasize two important aspects. First of all the study shows that it is important to chose a secure password, and secondly it shows that the underlying password handling mechanism of the operating system influence the password strength. In other words, if the underlying password handling mechanism is poor, the strength of the password is irrelevant. This is confirmed by the result obtained from the case when the LM hashes were evaluated. The weaknesses in the generation of the LM hash led to all of the 30 passwords being disclosed in 2 hours 15 minutes 57 seconds, which is far within the time limit of 8 hours. Microsoft improved the password handling used on for example Windows XP by disabling the LM hash, and introducing the more secure NT hash in newer operating systems. Despite this, the NT hash should not be considered secure enough as it still lacks the use of salt. This can be substantiated by the results presented in the empirical password study regarding the NT hashes, where 60% (18 passwords) of the passwords were revealed. With the NT hash, choosing a sufficiently secure password could prevent the password from being revealed. A secure password is a password that is easy to remember yet hard to guess. Based on the tools and rainbow tables used for the NT hashes, a *mnemonic* password with more than 8 characters in length should be selected. The password should also contain both uppercase- and lowercase letters, and special characters. It should be noted that the password requirements mentioned above might change quickly as processors are becoming more powerful, and as storage limitations is no longer an issue.

For the MD5 hashes, only a dictionary attack was performed. The cryptanalysis attack was omitted due to lack of rainbow tables and due to time constraints. The dictionary attack was aborted after 8 hours, which was the time limit for this study. During this attack 8 passwords had been revealed, and all of these passwords had a root that can be found in a dictionary.

The empirical password study demonstrates that the use of an initial dictionary attack might be advantageous. This is because the dictionary attack is quick to perform, and also may reveal passwords that the cryptanalysis attack fails to reveal, as was the case with the NT hashes. This of course depends on the size of the wordlist and the mangling rules that are used.

Even though Microsoft has introduced a more secure password handling by utilizing the NT hash, still many persons and corporations stick with Windows XP. This is partly because Windows Vista has been criticized for offering weak performance. From the results obtained from the empirical password study regarding the LM hash it is shown that as long as the password is less than 15 characters, every password is revealed no matter how secure the password might be. This is alarming because it may lead to losses for both individuals and corporations, both economically and personally. [pon08] pointed out that a lot of people lose their laptops at U.S airports, but it is also sufficient that the computer is left unattended for a while. It should be highlighted that if Windows XP is used, the use of LM hash should be disabled or a password consisting of more than 14 characters should be selected. When the password is 15 characters or longer, an operating system that utilizes both LM hash and NT hash, such as Windows XP, will only store the NT hash and not the LM hash. The results obtained from the empirical password study therefore show that the use of password should not be considered as a good authentication mechanism for future operating systems, and other solutions should be considered.

Every operating system should have a password resetting or recovery mechanism. If a password is lost only authorized persons should be able to recover the administrator password. The authorized user should only have *one* solution to reset or recover the password, instead of many different solutions. A good and already implemented solution is Microsoft's Windows Recovery Disk. It is desirable that the user is *enforced* to create the disk during the installation of the operating system. The user needs to be aware that this is the only solution to reset the password, and the disk should therefore not be lost. If all the other procedures and tools presented in this thesis have been prevented, the person with the Windows Recovery Disk is the only person that can reset the administrator password.

All of the operating systems included in the experiments presented in this thesis were subject to password resetting or recovery. Some of these systems had better solutions than others when it came to the obfuscation of the password file. The Windows operating systems have tried to obfuscate the SAM file, while the shadow file used by Linux systems are world readable. The systems also use different hash algorithms and some are more secure than others. In all of the Unix systems a salt is used when creating a hash value of the password, this is unexpectedly not the used in Windows systems. Even though the hash algorithm has been improved in newer Windows systems, it is the opinion of the authors of this thesis that the lack of salt serves as the greatest weakness in the hash construction.

In an operating system a good confidentiality and access control policy can prevent unauthorized access to the file system. On the other hand, none of the systems in this thesis enforce encryption. The problem with giving the ignorant user an option is that he/she will often choose the default option. A user should be enforced to enable disk encryption. This will prevent many of the procedures and tools described in this thesis from being carried out. It is also desirable that the operating system requires a user to select a good password, and prevents the user from continuing the installation process if certain requirements are not fulfilled. This also has to yield when a new user is added to the system. In addition, the system should not enforce a password hint, as this will probably downgrade the security.

Bibliography

[Ano99] Anonymous. *Maximum Linux Security: A Hacker's Guide to Protecting Your Linux Server and Workstation with CD-ROM.* SAMS, Indianapolis, IN, USA, 1999.

[Ben07] Johnny Bengtsson. Parallel Password Cracker: A Feasibility Study of Using Linux Clustering Technique in Computer Forensics. *Workshop on Digital Forensics and Incident Analysis, International*, 0:75–82, 2007. Available from: http://doi.ieeecomputersociety.org/10.1109/WDFIA.2007.10.

[BT04] Nathan Boeger and Mana Tominaga. *FreeBSD system programming.* GNU Free Documentation License, 2004. Available from: http://www.khmere.com/freebsd_book/.

[GLLR06] Lawrence A. Gordon, Martin P. Loeb, William Lucyshyn, and Robert Richardson. Csi/Fbi Computer Crime and Security Survey. 2006. Available from: http://i.cmpnet.com/gocsi/db_area/pdfs/fbi/FBI2006.pdf.

[Hel80] Martin E. Hellman. A Cryptanalytic Time - Memory Trade-off. *IEEE TRANSACTIONS ON INFORMATION THEORY*, IT-26:401–406, 1980. Available from: http://www-ee.stanford.edu/~hellman/publications/36.pdf.

[Jaa00] Martin G. Jaatun. LMHASH SPOOFING VHA SAMBA KLIENT. *FFI note*, 2000.

[Kle91] Daniel V. Klein. Foiling the Cracker: A Survey of, and Improvements to Password Security, (revised paper with new data). In *14th DoE Computer Security Group*, May 1991. Available from: http://www.klein.com/dvk/publications/passwd.pdf.

[KRC06] Cynthia Kuo, Sasha Romanosky, and Lorrie Faith Cranor. Human Selection of Mnemonic Phrase-based Passwords. In *SOUPS '06: Proceedings of the second symposium on Usable privacy and security*, pages 67–78, New York, NY, USA, 2006. ACM. Available from: http://doi.acm.org/10.1145/1143120.1143129.

[Leh03] Greg Lehley. *The Complete FreeBSD: Documentation from the Source 4th Edition.* O'REILLY COMMUNITY PRESS, 2003.

[MKM04] George V. Neville-Neil Marshall Kirk McKusick. *The Design and Implementation of the FreeBSD Operating System.* Addison Wesley, 2004.

[Oec03] Philippe Oechslin. Making a Faster Cryptanalytic Time-Memory Trade-Off. In *The 23rd Annual International Cryptology Conference, CRYPTO '03*, volume 2729 of *Lecture Notes in Computer Science*, pages 617–630, 2003. Available from: http://infoscience.epfl.ch/getfile.py?recid=99512&mode=best.

[pon08] Airport Insecurity: The Case of Missing & Lost Laptops. *Ponemon Institute*, June 30, 2008. Available from: http://www.dell.com/downloads/global/services/dell_lost_laptop_study.pdf.

[RJR08] Ernest Rothman, Brian Jepson, and Rich Rosen. *Mac OS X For Unix Geeks.* O'Reilly Media, Inc., 2008.

[RS04] Mark E. Russinovich and David A. Solomon. *Microsoft Windows Internals, Fourth Edition: Microsoft Windows Server 2003, Windows XP, and Windows 2000.* 2004.

[RT74] D. M. Ritchie and K. Thompson. The UNIX Time-Sharing System. *Communications of the ACM*, 17:365–375, 1974.

[Sch96] Bruce Schneier. *Applied Cryptography: Protocols, Algorithms and Source Code in C (Second Edition).* John Wiley & Sons, 1996.

[SL05] Edward Skoudis and Tom Liston. *Counter Hack Reloaded : A Step-by-Step Guide to Computer Attacks and Effective Defenses (2nd Edition) (Prentice Hall Series in Computer Networking and Distributed Systems).* Prentice Hall PTR, December 2005. Available from: http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20"&amp;path=ASIN/0131481045.

[Smi03] Roderick W. Smith. *FreeBSD: The Complete Reference.* McGraw-Hill/Osbourne, Berkley, CA, 2003.

[Tod07] Dobromir Todorov. *Mechanics of User Identification and Authentication - Fundamentals of Identity Management.* Auerbach Publications, 2007.

[WY05] Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions. In *EUROCRYPT*, pages 19–35, 2005. Available from: http://dx.doi.org/10.1007/11426639_2.

[YBAG04] J. Yan, A. Blackwell, R. Anderson, and A. Grant. Password Memorability and Security: Empirical Results. *Security & Privacy Magazine, IEEE*, 2(5):25–31, 2004. Available from: http://dx.doi.org/10.1109/MSP.2004.81, doi:10.1109/MSP.2004.81.

Web References

[1] NordSec 2009. The 14th Nordic Conference on Secure IT Systems; 2009. Available from: http://nordsec2009.unik.no/.

[2] NOTUR2009. The 8th Annual Meeting on High Performance Computing and Infrastructure in Norway; 2009. Available from: http://www.notur.no/notur2009/.

[3] Microsoft. How to log on to your Windows XP-based computer if you forget your password or if your password expires; 2008. Help and Support. Available from: http://support.microsoft.com/kb/321305.

[4] Net Applications. Operating System Market Share; 2009. marketshare.hitslink.com. Available from: http://marketshare.hitslink.com/report.aspx?qprid=10&qpmr=24&qpdt=1&qpct=3&qpcal=1&qptimeframe=M&qpsp=120&qpnp=1.

[5] remote exploit. BackTrack 4 beta release; 2009. Available from: http://www.remote-exploit.org/backtrack.html.

[6] techm4sters. Protech ONE; 2007. Available from: http://www.techm4sters.org/.

[7] IITAC International Institute; Cognitive Core UG (haftungsbeschränkt). Damn Vulnerable Linux; 2009. Available from: http://www.damnvulnerablelinux.org/.

[8] harakiri. Trinity Rescue Kit, CPR for your computer; 2008. trinityhome.org. Available from: http://trinityhome.org/Home/index.php?wpid=1&front_id=12.

[9] Ubuntu. Download an Ubuntu CD. Canonical Ltd; 2009. Available from: http://www.ubuntu.com/getubuntu/download.

[10] IDI NTNU. DAIM - Digital Arkivering og Innlevering av Masteroppgaver; 2009. Available from: http://daim.idi.ntnu.no/.

[11] Stallman R. Linux and the GNU Project; 2009. www.gnu.org. Available from: http://www.gnu.org/gnu/linux-and-gnu.html.

[12] Schneier B. Choosing Secure Passwords; January 11, 2007. schneier.com. Available from: http://www.schneier.com/blog/archives/2007/01/choosing_secure.html.

[13] AccessData. Password Recovery with PRTK/DNA; 2006. whitepaper. Available from: http://www.accessdata.com/media/en_US/print/papers/wp.PRTK-DNA_Password_Recovery.en_us.pdf.

[14] Brown S. Top 10 Most Common Passwords; 26 May, 2006. modernl.com. Available from: http://modernl.com/article/top-10-most-common-passwords.

[15] Naylor D. Top 10 passwords 2008; 10 April 2008. davidnaylor.co.uk. Available from: http://www.davidnaylor.co.uk/top-10-passwords-2008.html.

[16] Schneier B. MySpace Passwords Aren't So Dumb; 2006. schneier.com. Available from: http://www.schneier.com/essay-144.html.

[17] Schneier B. Real-World Passwords; 2006. schneier.com. Available from: http://www.schneier.com/blog/archives/2006/12/realworld_passw.html.

[18] Microsoft. What makes a strong password; 2006. microsoft.com. Available from: http://www.microsoft.com/protect/yourself/password/create.mspx.

[19] Schneier B. Write Down Your Password; June 17, 2005. schneier.com. Available from: http://www.schneier.com/blog/archives/2005/06/write_down_your.html.

[20] HooBie Inc. Brutus; 2009. hoobie.net. Available from: http://www.hoobie.net/brutus.

[21] Freeworld. THC Releases; 2007. freeworld.thc.org. Available from: http://freeworld.thc.org/releases.php.

[22] phenoelit. Default Password List; 2008. phenoelit-us.org. Available from: http://www.phenoelit-us.org/dpl/dpl.html.

[23] The A.R.G.O.N. Index of /achilles/wordlists; 07. theargon.com. Available from: http://www.theargon.com/achilles/wordlists/.

[24] Edge J. Rainbow tables for password cracking. 2006;Available from: http://lwn.net/Articles/208418/.

[25] Microsoft. Boot Configuration Data in Windows Vista; 2008. A paper named BCD.docx. Available from: http://www.microsoft.com/whdc/system/platform/firmware/bcd.mspx.

[26] Allen M. How Linux Works CTDP Guide Version 0.6.0; June 1, 2000. The Computer Technology Documentation Project. Available from: http://comptechdoc.org/os/linux/howlinuxworks/.

[27] The Linux Information Project. Runlevel Definition; 2007. Available from: http://www.linfo.org/runlevel_def.html.

[28] tsstahl(nickname). An introduction to run-levels. System Administration Tips and Resources. 2005;Available from: http://www.debian-administration.org/articles/212.

[29] Ubuntu. login - begin session on the system; 2008. Ubuntu manuals. Available from: http://manpages.ubuntu.com/manpages/intrepid/en/man1/login.1.html.

[30] Linux.com. An introduction to services, runlevels, and rc.d scripts. Feature. 2006;Available from: http://www.linux.com/articles/114107.

[31] Éric Lévénez. Unix History; 2009. levenez.com. Available from: http://www.levenez.com/unix/.

[32] Boeger N, Tominaga M. FreeBSD system programming. GNU Free Documentation License; 2004. Available from: http://www.khmere.com/freebsd_book/.

[33] OSx86 Project. EFI; 2009. wiki.osx86project.org. Available from: http://wiki.osx86project.org/wiki/index.php/EFI.

[34] Apple. rc – command script for boot; 2006. Mac OS X Man Pages. Available from: http://developer.apple.com/documentation/Darwin/Reference/ManPages/man8/rc.8.html.

[35] OSx86 Project. FAQ; 2009. wiki.osx86project.org. Available from: http://wiki.osx86project.org/wiki/index.php/FAQ#Do_I_need_Apple_hardware_to_run_Mac_OS_X.3F.

[36] fizzgig. pwdump6; 2009. foofus.net. Available from: http://foofus.net/fizzgig/pwdump/.

[37] Indiana University. In Windows 2000, XP, and Vista, what is the SID (security identifier)?; 2009. Knowledge Base. Available from: http://kb.iu.edu/data/aotl.html.

[38] Rubens P. Use Ophcrack to Defuse Windows' Security Timebomb; November 5, 2008. Enterprise Networking Planet. Available from: http://www.enterprisenetworkingplanet.com/netsecur/article.php/3783156/Use-Ophcrack-to-Defuse-Windows-Security-Timebomb.htm.

[39] Price J. ASCII Chart; 2009. JimPrice.Com. Available from: http://www.jimprice.com/jim-asc.shtml.

[40] Fairhurst G. Characters and Strings, & ASCII; 2001. University of Aberdeen. Available from: http://www.erg.abdn.ac.uk/users/gorry/eg2069/ascii.html.

[41] Tsang T. Brute Force Search of a DES Keyspace; 2008. Cornell University. Available from: http://instruct1.cit.cornell.edu/courses/ece576/FinalProjects/f2008/tt236/tt236/high_level_design.html.

[42] Glass E. The NTLM Authentication Protocol and Security Support Provider; 2006. sourceforge.net. Available from: http://davenport.sourceforge.net/ntlm.html#theLmResponse.

[43] Tropical Software. DES Encryption; 2007. tropsoft.com. Available from: http://www.tropsoft.com/strongenc/des.htm.

[44] Ophcrack. What is ophcrack?; 2009. sourceforge.net. Available from: http://ophcrack.sourceforge.net/.

[45] Microsoft. ASCII character chart (ASCII printing characters). 2007;Available from: http://office.microsoft.com/en-us/project/HA011331361033.aspx.

[46] The Unicode Consortium. The Unicode Standard: A Technocal Introduction; 2008. unicode.org. Available from: http://www.unicode.org/standard/principles.html.

[47] Microsoft. Local Security Authority; 2009. msdn.microsoft.com. Available from: http://msdn.microsoft.com/en-us/library/ms721592(VS.85).aspx.

[48] Ubuntu. passwd - the password file; 2008. Ubuntu manuals. Available from: http://manpages.ubuntu.com/manpages/intrepid/en/man5/passwd.5.html.

[49] Ubuntu. shadow - encrypted password file; 2008. Ubuntu manuals. Available from: http://manpages.ubuntu.com/manpages/intrepid/en/man5/shadow.5.html.

[50] kernel.org. crypt, crypt_r - password and data encryption; 2008. Linux Programmer's Manual. Available from: http://www.kernel.org/doc/man-pages/online/pages/man3/crypt.3.html.

[51] GNU. 32.3 Encrypting Passwords; 2008. GNU Libtool Manual. Available from: http://www.gnu.org/software/libtool/manual/libc/crypt.html.

[52] Drepper U. Unix crypt using SHA-256 and SHA-512; 2008. people.redhat.com. Available from: http://people.redhat.com/drepper/SHA-crypt.txt.

[53] Ubuntu. login.defs - shadow password suite configuration; 2008. Ubuntu manuals. Available from: http://manpages.ubuntu.com/manpages/intrepid/en/man5/login.defs.5.html.

[54] Dribin D. How Mac OS X Implements Password Authentication, Part 2. Dave Dribin's Blog. 2006;Available from: http://www.dribin.org/dave/blog/archives/2006/04/28/os_x_passwords_2/.

[55] MacShadow KB. Mac OS X password hashes; 2009. macshadows.com. Available from: http://www.macshadows.com/kb/index.php?title=Mac_OS_X_password_hashes.

[56] Nordahl-Hagen P. Offline NT Password & Registry Editor; 2008. bootdisk. Available from: http://home.eunet.no/~pnordahl/ntpasswd/bootdisk.html.

[57] Bania P. Kon-Boot; 2009. piotrbania.com. Available from: http://piotrbania.com/all/kon-boot/.

[58] Lagerweij B. Bart's Preinstalled Environment (BartPE) bootable live windows CD/DVD; 2006. Version 3.1.10a. Available from: http://www.nu2.nu/pebuilder/.

[59] NightMan. VistaPE; 2008. vistape.net. Available from: http://www.vistape.net/.

[60] Mark Russinovich BC. NTFSDOS; 2001. A mirror of the original page. Available from: http://arcadecontrols.com/Mirrors/mirror/www.sysinternals.com/Utilities/NtfsDos.html.

[61] Avira. Avira NTFS4DOS Personal; 2009. free-av.com. Available from: http://www.free-av.com/en/tools/11/avira_ntfs4dos_personal.html.

[62] sala. Password Renew; 2007. sala source. Available from: http://www.kood.org/windows-password-renew/.

[63] Bakowski D. WindowsGate 1.1, Enabling/disabling Windows logon password validation; 2008. Forum at www.911cd.net. Available from: http://www.911cd.net/forums//index.php?showtopic=21204.

[64] PcLoginNow. PC Login Now; 2009. pcloginnow.com. Available from: http://www.pcloginnow.com/.

[65] My Digital Life. Hack to Log On to Windows XP and 2000 without Changing Existing Password with DreamPackPL; 2004. mydigitallife.info. Available from: http://tinyurl.com/dreampackpl2004.

[66] Tait J. DreamPackPL with Password Protection; 2004. Plugin for BartPE. Available from: http://userwww.sfsu.edu/~john/bartspe/DreamPackDocs/DreamPackPL_pw.htm.

[67] RainbowCrack. Project RainbowCrack; 2009. project-rainbowcrack.com. Available from: http://project-rainbowcrack.com/.

[68] Openwall Project. John the Ripper password cracker; 2009. openwall.com. Available from: http://www.openwall.com/john/.

[69] Openwall Project. John the Ripper's cracking modes; 2009. openwall.com. Available from: http://www.openwall.com/john/doc/MODES.shtml.

[70] Openwall Project. Wordlist rules syntax; 2009. openwall.com. Available from: http://www.openwall.com/john/doc/RULES.shtml.

[71] dninja. CeWL - Custom Word List generator; 2009. digininja.org. Available from: http://www.digininja.org/cewl.php.

[72] Montoro M. Cain & Abel; 2009. oxid.it. Available from: http://www.oxid.it/cain.html.

[73] Montoro M. Password Crackers; 2009. oxid.it. Available from: http://www.oxid.it/ca_um/.

[74] LCPSoft. LCP; 2007. lcpsoft.com. Available from: http://www.lcpsoft.com/english/index.htm.

[75] fizzgig. fgdump; 2008. foofus.net. Available from: http://swamp.foofus.net/fizzgig/fgdump/fgdump-2.1.0-exeonly.zip.

[76] ncuomo. Windows 2k/NT/XP's syskey encryption; 2006. studenti.unina.it. Available from: http://www.studenti.unina.it/~ncuomo/syskey/.

[77] Montoro M. Projects; 2009. oxid.it. Available from: http://www.oxid.it/projects.html.

[78] Inc Sun Microsystems. VirtualBox; 2009. virtualbox.org. Available from: http://www.virtualbox.org/.

[79] Gupta V. How to Reset/Recover Forgotten Windows NT/2000/XP/2003 Administrator Password?; 2009. Tweaking with Vishal v2. Available from: http://tinyurl.com/2axlpr.

[80] Microsoft. How to Use the Net User Command. 2007;Available from: http://support.microsoft.com/kb/251394.

[81] phirleon (nickname). password hash; 2008. vpsmedia Forum. Available from: https://www.vpsmedia.com/forum/showthread.php?t=3.

[82] rearthur2003 (nickname). Replace Sethc.exe With Cmd.exe; 2007. Blackhat Forums. Available from: http://www.blackhat-forums.com/index.php?showtopic=3613.

[83] Apple Inc. Mac OS X: Changing or resetting an account password; 2009. Apple Support. Available from: http://support.apple.com/kb/HT1274.

[84] Microsoft. How to create and use a password reset disk in Windows Vista; 2009. support.microsoft.com. Available from: http://support.microsoft.com/kb/930381.

[85] Microsoft. Create a Password Reset Disk; 2001. microsoft.com. Available from: http://www.microsoft.com/windowsxp/using/setup/learnmore/tips/keramidas1.mspx.

[86] semprini, freshmint, Hephaestus, William42, RancidPickle, (nicknames). How to recover a lost Linux root or Windows 2000 Administrator password; 2000. Everything2 forum. Available from: http://everything2.com/title/How%2520to%2520recover%2520a%2520lost%2520Linux%2520root%2520or%2520Windows%25202000%2520Administrator%2520password.

[87] TechTarget. Bourne shell; 2005. SearchEnterpriseLinux.com Definitions. Available from: http://searchenterpriselinux.techtarget.com/sDefinition/0,,sid39_gci214635,00.html.

[88] Pash A. Reset your lost OS X password; 2007. MAC TIP. Available from: http://lifehacker.com/software/mac-tip/reset-your-lost-os-x-password-278898.php.

[89] Admin G. Lost root password?; 04. BSDZONE. Available from: http://www.bsdzone.net/weblog/archives/2004/11/19/lost-root-password/.

[90] ImgBurn. Download: ImgBurn v2.4.2.0 (1,926 KB). LIGHTNING UK!; 2009. Available from: http://www.imgburn.com/index.php?act=download.

[91] Wal K. Accessing a Fedora Logical Volume from Ubuntu; 2008. inux-sxs.org. Available from: http://www.linux-sxs.org/storage/fedora2ubuntu.html.

[92] Red Hat. Get Fedora 10 Desktop Edition Now; 2009. fedoraproject.org. Available from: http://fedoraproject.org/en/get-fedora.

[93] PcLoginNow. PCLoginNow2_0_stable; 2008. sourceforge.net. Available from: http://sourceforge.net/project/showfiles.php?group_id=231735.

[94] S.L. WinZip Computing. Download WinZip Evaluation Version; 2009. executable. Available from: http://www.winzip.com/downwz.htm.

[95] The Shmoo Group. Rainbow Tables; 2006. rainbowtables.shmoo.com. Available from: http://rainbowtables.shmoo.com/.

[96] The Shmoo Group. Rainbow Tables; 2006. rainbowtables.shmoo.com. Available from: http://205.127.87.136:6969/torrents/alpha_num_sym32_space.torrent?6BF7984266AB058D020E49B36B84F950E0BFA933.

[97] Debian. Debian; 2009. debian.org. Available from: http://www.debian.org/.

[98] Torvalds L. The Linux Kernel Archives; 2009. kernel.org. Available from: http://www.kernel.org/.

[99] Debian Live Project. Debian Live; 2009. debian.org. Available from: http://debian-live.alioth.debian.org/.

[100] Debian Live Project. DebianLive/live-helper; 2009. debian.org. Available from: http://wiki.debian.org/DebianLive/live-helper.

[101] TrueCrypt Foundation. TrueCrypt; 2009. truecrypt.org. Available from: http://www.truecrypt.org/.

[102] RainbowCrack. List of Rainbow Tables; 2009. project-rainbowcrack.com. Available from: http://project-rainbowcrack.com/table.htm.

[103] Aberystwyth University. The University's password generator page; 2000. aber.ac.uk. Available from: http://users.aber.ac.uk/auj/portfolio/mnemonic.shtml.

[104] PC Tools. Secure Password Generator; 2009. pctools.com. Available from: http://www.pctools.com/guides/password/.

[105] Klein DV. Foiling the Cracker: A Survey of, and Improvements to Password Security, (revised paper with new data). In: 14th DoE Computer Security Group; May 1991. Available from: http://www.klein.com/dvk/publications/passwd.pdf.

[106] freerainbowtables. Free Rainbow Tables; Distributed Rainbow Table Project; 2009. freerainbowtables.com. Available from: http://www.freerainbowtables.com/.

[107] Schneier B. Secret Questions; 2009. schneier.com. Available from: http://www.schneier.com/blog/archives/2009/05/secret_question.html.

[108] FreeTDS. Domain Logins; 2006. FreeTDS User Guide: A Guide to Installing, Configuring, and Running FreeTDS. Available from: http://www.freetds.org/userguide/domains.htm.

[109] Microsoft Corporation. What's the difference between BitLocker Drive Encryption and Encrypting File System?; 2009. Windows Help and How-to. Available from: http://windowshelp.microsoft.com/Windows/en-US/help/62f28747-c146-483c-b378-ff3f3977f0111033.mspx.

[110] Ou G. Prevent data theft with Windows Vista's Encrypted File System (EFS) and BitLocker. 2007;Available from: http://articles.techrepublic.com.com/5100-10878_11-6162949.html.

[111] Thomas K. A User's Look at Ubuntu 8.10 Intrepid Ibex; 2008. lifehacker article. Available from: http://lifehacker.com/5072351/a-users-look-at-ubuntu-810-intrepid-ibex.

[112] Utimaco Safeware. SafeGuard Easy; 2009. Available from: http://go.utimaco.com/safeguard-easy/.

[113] Erwan Le Gall MD. [HOWTO] Build and using a custom kernel; 2008. debian-live-devel mailing list. Available from: http://lists.alioth.debian.org/pipermail/debian-live-devel/2008-April/003456.html.

[114] ActiveState. ActivePerl; 2009. activestate.com. Available from: http://www.activestate.com/activeperl/.

Appendix A

How to Build YALP

This appendix will give a rough description on how to build the tool YALP. YALP is a live CD based on Debian Lenny. The CD was created with a program called live-helper available for Debian. This tutorial may also work on future releases of Debian.

There are some prerequisites before the tool can be built. Live-helper [99] has to be installed on the system subject to building YALP and the scripts and folders must apply to the commands presented in the script below. We used VirtualBox [78] and installed Debian [97] as a virtual machine. Inside this virtual machine we installed subversion and live-helper. Through subversion we fetched a YALP folder including the *build_YALP.sh* script presented below in Appendix A.1 and the necessary files. The *build_YALP.sh* script then created an image (binary.iso) file, namely the YALP live CD image.

In the YALP live CD we include a custom build Linux kernel with support for UFS file system used by FreeBSD. To build this custom kernel we used the *build_kernel.sh* script also presented below in Section A.2. Erwan Le Gall and Michel Depeige have a great tutorial in how to integrate a custom kernel in a Debian live environment at [113]. The *build_kernel.sh* script must be modified if other versions than 2.6.29.3 are used. We used the kernel configuration *.config* that was used when a standard Debian live distribution was made. The *.config* file is then located in */boot* in the newly created distribution. When running the *build_kernel.sh* script various questions concerning drivers may appear, the default value is usually a good option. When the script finished the output was some *.deb* files which can then be moved to custom_kernel folder used in *build_YALP.sh*.

A.1 build_YALP.sh

The YALP built script (build_YALP.sh) is presented below:

```
#!/bin/bash

custom_kernel="../custom_kernel/2.6.26.2"
```

```
images="../images"
other_packages="../other_packages"
other_scripts="../other_scripts"
our_scripts="../our_scripts"
local_hooks="../local_hooks"

aptitude install --assume-yes live-helper

mkdir livecd
cd livecd

lh_config --iso-volume "YALP" --iso-publisher "Jørgen Blakstad and Rune\
     Walsø Nergård" --hostname YALP --username user --bootappend-live "\
    keyb=no" --mirror-bootstrap "http://ftp.no.debian.org/debian/" --\
    mirror-chroot "http://ftp.no.debian.org/debian/" --mirror-binary "\
    http://ftp.no.debian.org/debian/" --packages "chntpw locate john \
    ntfs-3g openssh-client libdigest-md4-perl libcrypt-passwdmd5-perl \
    libfuse2 fuse-utils dmsetup libsm6 lvm2 less" --syslinux-menu \
    enabled --syslinux-timeout 10

mkdir config/chroot_local-includes/home
mkdir config/chroot_local-includes/home/user
cp $our_scripts/* $other_scripts/* config/chroot_local-includes/home/\
    user/
cp $other_packages/* $custom_kernel/* config/chroot_local-packages/
cp $images/yalp.rle config/binary_syslinux/yalp.rle
sed 's/LH_SYSLINUX_SPLASH=""/LH_SYSLINUX_SPLASH="config"/\
    binary_syslinux"/yalp.rle"/' config/binary >test
sed 's/LH_SYSLINUX_MENU_LIVE_ENTRY="Start Debian Live"/\
    LH_SYSLINUX_MENU_LIVE_ENTRY="Start YALP"/' test >test2
mv test2 config/binary
rm test
sed 's/LH_LINUX_PACKAGES="/LH_LINUX_PACKAGES="none" #/' config/chroot >\
    test
sed 's/LH_LINUX_FLAVOURS="/LH_LINUX_FLAVOURS="dlb" #/' test >test2
mv test2 config/chroot
rm test
cp $local_hooks/* config/chroot_local-hooks
lh_build
```

Listing A.1: Script to build YALP.

A.2 build_kernel.sh

The custom Linux kernel built script (build_kernel.sh) is presented below:

```
#!/bin/bash

# This script will/should build a custom kernel possible to use with \
    the build of YALP

distro="2.6.26.2"
lsource="http://kernel.org/pub/linux/kernel/v2.6/linux-$distro.tar.bz2"
lpatch="http://www.kernel.org/pub/linux/kernel/v2.6/patch-$distro.bz2"
workingdir="`pwd`"
configdir="$workingdir/config" # Kernel config dir

cd /usr/src/
rm -fr *.deb

wget $lsource
tar -xvjf linux-$distro.tar.bz2
cd linux-$distro
wget $lpatch
bunzip2 patch-$distro.bz2
patch -R -p1 < patch-$distro
cp $configdir ./.config # Used config located in /boot/ in a already \
    created livecd
sed 's/# CONFIG_UFS_FS_WRITE is not set/CONFIG_UFS_FS_WRITE=y/' .config\
     >test # (vim .config -> CONFIG_UFS_FS_WRITE=y)
mv test .config
aptitude install --assume-yes module-assistant gcc libncurses-dev \
    fakeroot kernel-package zlib1g-dev
fakeroot make-kpkg --initrd --revision=dlb.1.0 --append-to-version=-dlb\
     kernel_image kernel_headers modules
aptitude install --assume-yes lzma-source squashfs-source aufs-source
cd /usr/src
rm -rf /usr/src/linux
ln -s linux-$distro linux
m-a update
m-a build -k /usr/src/linux-$distro lzma
m-a build -k /usr/src/linux-$distro squashfs
mkdir $workingdir/$distro
cp /usr/src/*.deb $workingdir/$distro
```

Listing A.2: Script to build custom kernel of YALP.

Appendix B

YALP Scripts

In this appendix the various scripts created for YALP are presented. A description on how to build YALP can be found in Appendix A.

B.1 startup.sh

```
#!/bin/bash

# This script is the script that is initiated when YALP has finished \
    the boot process.
# It is a menu for all the other scripts. To work properly, the \
    following files has to be included:
# agreement, menu, menu1, menu2, menu3 and the rest of the scripts.

# some variables

mountdir="/mnt/os"
folder="/home/user"
#folder="/media/Documents/master/thesis/scripts/YALP/our_scripts"


# initial licence and agreement

sudo cat $folder/agreement
echo "Do you understand and agree to the terms mentioned above?"
read agree
if [ -z $agree ]
       then
       sudo shutdown -h now
```

```
elif [ $agree != "yes" ]
       then
       sudo shutdown -h now
fi

# a method checking if a system is mounted

ismounted () {
       if [ -d $mountdir ]
              then
              tmp=‘ls -A $mountdir | grep -c .‘
              if [ $tmp -eq 0 ]
                     then
                     echo "It looks like no host filesystem has been \
                        mounted."
                     echo "Will first run a mount script (mountos.sh)"
                     sudo sh $folder/mountos.sh
                     if [ $? -eq 113 ]
                            then
                            echo "Will return to main menu!"
                            menu
                     fi
              fi
       else
              echo "It looks like no host filesystem has been mounted."
              echo "Will first run a mount script (mountos.sh)"
              sudo sh $folder/mountos.sh
              if [ $? -eq 113 ]
                     then
                     echo "Will return to main menu!"
                     menu
              fi
       fi
}


# supplementary tools menu

menu3 () {
       sudo cat $folder/menu3
       read option
       case $option in
              1)
```

```
                        sudo sh $folder/mountos.sh
                        menu
                        ;;
                2)
                        sudo fdisk -l
                        menu
                        ;;
                3)
                        sudo truecrypt -h|less
                        ;;
                4)
                        menu
                        ;;
                *)
                        echo "Please enter a valid number!"
                        menu3
                        ;;
        esac
}


# password recovery tools menu

menu2 () {
        sudo cat $folder/menu2
        read option
        case $option in
                1)
                        sudo ophcrack-cli|less
                        ;;
                2)
                        sudo john|less
                        ;;
                3)
                        sudo perl $folder/lm2ntcrack.pl
                        ;;
                4)
                        menu
                        ;;
                *)
                        echo "Please enter a valid number!"
                        menu2
                        ;;
        esac
}


# password resetting tools menu

menu1 () {
        sudo cat $folder/menu1
        read option
        case $option in
                1)
                        ismounted
                        sudo perl $folder/editshadow.pl $mountdir/etc/\
                           shadow
                        menu
                        ;;
                2)
                        ismounted
                        sudo perl $folder/editmasterpasswd.pl $mountdir/\
                           etc/master.passwd
                        menu
                        ;;
                3)
                        ismounted
                        sudo sh $folder/replace_startup_files.sh
                        menu
                        ;;
                4)
                        sudo chntpw
                        ;;
                5)
                        menu
                        ;;
                *)
                        echo "Please enter a valid number!"
                        menu1
                        ;;
        esac
}

# main menu with options

menu () {
        sudo cat $folder/menu
        read option
        case $option in
                1)
                        menu1
                        ;;
                2)
                        menu2
                        ;;
                3)
                        menu3
                        ;;
                4)
                        exit
                        ;;
                5)
                        sudo shutdown -r now
                        ;;
                6)
                        sudo shutdown -h now
                        ;;
                *)
                        echo "Please enter a valid number!"
                        menu
                        ;;
        esac
}

menu
```

Listing B.1: YALP startup script.

B.2 replace_startup_files.sh

```
#!/bin/bash

# This script ease the process of performing the "Replace a pre-login \
    executable file" procedure
# It will replace the sethc.exe with cmd.exe and make a backup of the \
    old sethc.exe file.
# This script has also got support for reverting back to normal after \
    the resetting has been done.

path=$1
mountfolder="/mnt/os"
xp="$mountfolder/WINDOWS/system32"
vistaN7="$mountfolder/Windows/System32"

if [ -z $path ]
        then
        if [ -d $xp ]
                then
                path=$xp
        elif [ -d $vistaN7 ]
                then
                path=$vistaN7
        else
                echo "Please enter the path to the windows system32 \
                   folder e.g. /mnt/os/Windows/System32/"
                read tmp
                path=$tmp
        fi
fi

finished="false"
while [ $finished == "false" ]
do
        echo --------------------------------------------------------- \
            --------------------------------
        echo This script can replace the sticky keys startup program or \
            revert to backup if already done.
        echo --------------------------------------------------------- \
            --------------------------------
        echo
```

```
echo "Do you want to 'replace' the sethc.exe or 'revert' to \
   backup?"
read answer
case $answer in
       replace)
              if [ -e $path/sethc.exe.bak ]
                     then
                     echo "It seems that a backup file exists. \
                        Maybe you have already replaced the \
                        file."
                     echo "Are you sure you want to replace? \
                        Type 'yes' to confirm."
                     read answer3
                     if [ $answer3 == "yes" ]
                            then
                            cp $path/sethc.exe $path/sethc.exe.\
                               bak
                            cp $path/cmd.exe $path/sethc.exe
                            echo ------------------------------\
                                ------------------------------ \
                               -----------------
                            echo Replace done. You can now \
                               reboot the machine and by \
                               pressing shift 5 times
                            echo enter a Windows command \
                               console, cmd.exe, to perform \
                               further password handling.
                            echo ------------------------------\
                                ------------------------------ \
                               -----------------
                            exit
                     fi
              else
                     echo "Are you sure you want to replace the \
                        startupfile? Type 'yes' to confirm."
                     read answer2
                     if [ $answer2 == "yes" ]
                            then
                            cp $path/sethc.exe $path/sethc.exe.\
                               bak
                            cp $path/cmd.exe $path/sethc.exe
                            echo ------------------------------\
                                ------------------------------ \
                                   -----------------
                               echo Replace done. You can now \
                                   reboot the machine and by \
                                   pressing shift 5 times
                               echo enter a Windows command \
                                   console, cmd.exe, to perform \
                                   further password handling.
                               echo ------------------------------\
                                    ------------------------------ \
                                   -----------------
                               exit
                       fi
               fi
               ;;
       revert)
               echo Will try to revert from the backupfile \
                   hopefully located at:
               echo $path/sethc.exe.bak
               mv $path/sethc.exe.bak $path/sethc.exe
               echo -------------------------------------------- \
                   ----------------------------------
               echo Revert done. You can now reboot the machine \
                   and sticky keys should be restored
               echo to normal. The changes done from Windows \
                   command console will not be reverted.
               echo -------------------------------------------- \
                   ----------------------------------
               exit
               ;;
       *)
               echo Will exit, have a nice day!
               exit
               ;;
   esac
done
```

Listing B.2: Replace startup file script.

B.3 editshadow.pl

```
#!/usr/bin/perl
use Crypt::PasswdMD5;
use File::Copy;

# This script edits the /etc/shadow file. It replaces an old md5 hash \
    with a new one created for a desired password


my $old = "";
foreach (@ARGV) { $old = "$_" }
if($old eq ""){
       print "Please enter the folder where the shadow file is located \
           ie. '/etc/shadow' : ";
       chomp($old = <STDIN>);
}
#my $old = '/etc/shadow';
my $new = 'shadow.tmp';
my $bak = "$old.bak";
my $typo = 0;
my $edit = 0;

open(OLD,'<',$old)   or die "can't open $old: $!";
open(NEW,'>',$new)   or die "can't open $new: $!";

my($username,$temp,$temp2,$temp3,$alg,$salt2,$hash2,$rest,$line);
while($line = <OLD>){
       if($line =~ m/"$1"$/){
#  print "$line";
              $username = substr $line, 0, index($line, ':');
              print "Is '$username' the correct user? Enter 'yes' or '\
                  no': ";
              chomp(my $correctuser = <STDIN>);
              if($correctuser eq "yes"){
                     print "Insert desired password: ";
                     chomp(my $password = <STDIN>);
                     print "Insert desired salt: ";
                     chomp(my $salt = <STDIN>);
                     $crypted = unix_md5_crypt($password, $salt);
                     my $hash = substr $crypted, 4+length($salt);
                     $edit = 1;
                     $temp = substr $line, length($username);
```

```
                        $alg = substr $temp, 2, 1;
                        $temp2 = substr $temp, 4;
                        $salt2 = substr $temp2, 0, index($temp2, '$');
                        $temp3 = substr $temp2, length($salt2);
                        $hash2 = substr $temp3, 1, index($temp3,':')-1;
                        $rest = substr $temp3, length($hash2);
                        $line =~ s/$salt2/$salt/;
                        $line =~ s/$hash2/$hash/;
#   print "$line"n";
                        print NEW $line;
                }elsif($correctuser eq "no"){
                        print NEW $line;
                }else{
                        print "You typed wrong, please type 'yes' or 'no\
                            '!"n";
                        $typo = 1;
                }
        }else{
                print NEW $line;
        }
}
if($typo == 0 && $edit == 1){

        close(OLD)      or die "can't close $old: $!";
        close(NEW)      or die "can't close $new: $!";

        move($old, $bak) or die "can't rename $old to $bak: $!";
        move($new, $old) or die "can't rename $new to $old: $!";
}else{
        close(OLD)      or die "can't close $old: $!";
        close(NEW)      or die "can't close $new: $!";

        unlink($new)     or die "can't delete $old: $!";
}
```

Listing B.3: Edit shadow file script.

B.4 mountos.sh

```
#!/bin/bash
os=$1
mountdir="/mnt/os"

# This script will help the user with mounting a host system, it needs \
    a lot of improvements.


if [ -z $os ]
        then
        echo "Please enter a os you want to mount! (Win/Ubuntu/Fedora/\
            FreeBSD/unmount)."
        read mount
        os=$mount
fi
if [ -d $mountdir ]
        then
        echo "will unmont $mountdir if it is already mounted."
        umount $mountdir
else
        mkdir $mountdir
fi
case $os in
        unmount|umount)
                rmdir $mountdir
                echo unmounting done
                exit
                ;;
        *)
                echo "Mount directory $mountdir created."
                echo "The partition device can be found later, by running\
                     'sudo fdisk -l'."
                echo "Here is the output of fdisk -l:"
                fdisk -l
                echo "Enter partition device with full path, ie. /dev/\
                    sda2."
                echo "(press enter if you want us to try common options \
                    (/dev/sda1 or /dev/hda1))."
                read device
                dev=$device
                if [ -z $dev ]
                then
                if [ -a /dev/sda1 ]
                        then
                        echo "Partition device is /dev/sda1."
                        dev="/dev/sda1"
                elif [ -a /dev/hda1 ]
                        then
                        echo "Partition device is /dev/hda1."
                        dev="/dev/hda1"
                else
                        echo "Harddrive device is not found."
                        echo "Please enter partition device:"
                        read device
                        dev=$device
                fi
        elif [ -a $dev ]
                then
                echo "Will try to mount with $dev as partition \
                    device"
        else
                echo "Harddrive device was not supported!"
                echo "Please run this script again with correct \
                    device values"
                exit 113
        fi
esac
case $os in
        FreeBSD|freeBSD|freebsd|bsd|BSD)
        echo will try mount $os
                mount -t ufs -o rw,ufstype=ufs2 $dev $mountdir
                if [ "$?"-ne 0 ]; then echo "Could not mount! returning \
                    to main menu."; exit 113; fi
                ;;
        Ubuntu*|ubuntu*|Fedora*|fedora*)
        echo will try mount $os
                mount -t ext3 $dev $mountdir
                if [ "$?"-ne 0 ]; then echo "Could not mount! returning \
                    to main menu."; exit 113; fi
                ;;
        Win*|win*)
        echo will try mount $os
                mount -t ntfs-3g $dev $mountdir
                if [ "$?"-ne 0 ]; then echo "Could not mount! returning \
                   to main menu."; exit 113; fi
                ;;
        Mac*|mac*)
        echo will try mount $os
                echo "$os not yet supported!"
                exit 113
                ;;
        *)
                echo "$os not supported, filesystem not mounted."
                echo "Is $os an operating system?"
                echo "Please try again, returning to main menu."
                rmdir $mountdir
                exit 113
                ;;
esac
if [ -d $mountdir ]
        then
        tmp=`ls -A $mountdir | grep -c .`
        if [ $tmp -eq 0 ]; then
                echo "Something went wrong, $dev was not mounted to \
                   $mountdir."
                echo "Have you typed in the correct host os?"
                rmdir $mountdir
                exit 113
        else
                echo "It looks like the file system was successfully \
                   mounted to $mountdir."
        fi
fi
```

Listing B.4: Mount operating system script.

Appendix C

How to Disable LM Hashing

In this appendix a possible way to disable the storing of the LM hash on your computer is presented. Recall that this is done by default in Windows Vista. The approach, found in [38], is presented below:

1. Click Start, click Run, type *regedit*, and then click OK.

2. Locate and then click the following key in the registry:
   HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa

3. On the Edit menu, point to New, and then click DWORD Value.

4. Type *NoLMHash*, and then press ENTER. If the DWORD value is already there, edit the existing DWORD instead.

5. On the Edit menu, click Modify.

6. Type 1, and then click OK.

7. Restart your computer, and then change your password.

Appendix D

How to Add 30 Users

In this appendix we present the scripts that we have made to easily add the 30 users in the Empirical Password Study presented in 8. Please note that there are 2 different scripts presented below, depending on whether you want to add the users on a Windows or Unix system. The scripts are written in Perl which is included in most Linux distributions, and can for instance be found for Windows at ActiveState's web page [114]. The scripts require a list of passwords, listed each password per line presented in a file, as an argument. The programs have to run under Administrator privileges. The script for Unix also requires that the *useradd* program is present in the operating system.

D.1 addusers_win.pl

The script below adds 30 users on a *Windows* system.

```
#!/usr/bin/perl

my $passwdfile = $ARGV[0];
if($passwdfile eq ""){
       print "Please enter the name of the passwordfile (located the \
          same place as this script): ";
       chomp($passwdfile = <STDIN>);
}

open(PASSWDFILE,'<',$passwdfile)   or die "can't open $passwdfile: $!";

$count = 1;
while($password = <PASSWDFILE>){
       $username = "user$count";
       chomp($password); # remove newlines:)
       print "Will add $username with password $password "n";
```

```
        $cmd = "NET USER $username $password /add";
        $cmd2 = "NET LOCALGROUP Administrators $username /add";
        $cmd3 = "NET GROUP ""Domain Admins"" $username /add ";
        system $cmd;
        system $cmd2;
        $count = $count + 1;
}
```

Listing D.1: Script to add 30 users in a Windows system.

D.2 addusers_lin.pl

The script below adds 30 users on a *Unix* system.

```
#!/usr/bin/perl

my $passwdfile = $ARGV[0];
if($passwdfile eq ""){
        print "Please enter the name of the passwordfile (located the \
           same place as this script): ";
        chomp($passwdfile = <STDIN>);
}

open(PASSWDFILE,'<',$passwdfile)   or die "can't open $passwdfile: $!";

$count = 1;
while($password = <PASSWDFILE>){
        $username = "user$count";
        chomp($password); # remove newlines:)
        print "Will add $username with password $password "n";
        $cmd = "sudo useradd $username -p `echo $password | mkpasswd -s \
           -H md5`";
        system $cmd;
        $count = $count + 1;
}
```

Listing D.2: Script to add 30 users in a Unix system.

NOTE: If you want to copy/paste the scripts above to a file, be aware of that the / at the end of some the lines just means line break, and shall not be a part of the script.

Appendix E

Attachments

The attachments to this thesis consist of both an enclosed CD and *electronic* attachments. The latter is submitted to the DAIM system [10].

E.1 Attached CD

Enclosed in this thesis is a CD containing our self made tool YALP, a bootable live CD. To boot the CD you have to insert it into your CD ROM drive and then select to boot from CD ROM during startup. YALP is a tool that can be used for password resetting and password recovery.

E.2 Electronic Attachments

The electronic attachment (thesis.zip), which is submitted to the DAIM system [10], includes the following directories:

- **YALP** contains all the necessary files and folders to build YALP.
- **Other** contains other scripts and files that were used in the experiments described in our thesis. This includes password files and a character set.

It should be added that *README.txt* files, which explain the content of these directories in more details, are also included in the *thesis.zip* file.

Generation of Rainbow Tables

The material on this poster is extracted from the Master Thesis “Tools and Techniques for Resetting or Recovery of Administrator Passwords on Popular Operating Systems”.

Prepared by **Jørgen Blakstad** and **Rune W. Nergård**,
under the supervision of Prof. Danilo Gligoroski
Department of Telematics, NTNU
Email: jorgenbl@stud.ntnu.no, runewals@stud.ntnu.no

Background

Passwords can be used to protect the user account on your computer preventing unauthorized persons to enter the system. But what if you, as an authorized person, forget this password? And would it not be nice to check the strength of the password before you use it? A strong password can prevent unauthorized persons to access the system, and a good password consists of both uppercase and lowercase letters, numbers and special characters. It should also be as long as possible, appear like a random generated password and still be easy to remember. Rainbow tables can be used to recover your password.

The passwords are stored on the computer as hash values. Popular hash algorithms used are LM hash, NT hash, MD5 hash and SHA-512 hash. The LM and NT hash values are stored in the SAM file in Windows systems, and the MD5 and SHA-512 hash values, among others, are stored in password files in UNIX systems. Many UNIX systems use salt and password as input when generating the hash value. A salt contains random bits, and a new salt is generated each time it is used. This makes the hash value harder to crack with rainbow tables. Windows systems do not use salt.

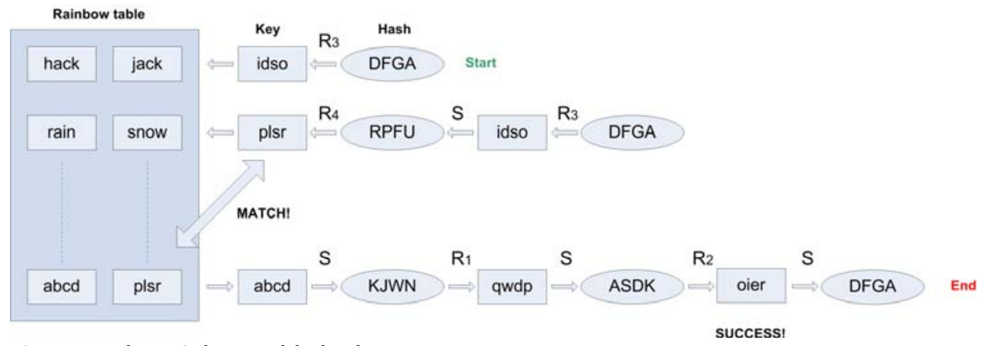


Figure 1: The rainbow table lookup process.

Rainbow tables contain a connection between a hash value and its corresponding password. By having this connection already available for all possible hash values, a quick search through the tables for a desired hash value can reveal the password.

Hellman [1] proposed a time-memory trade-off when applying a cryptanalytic attack. Hellman used something called hash chains to decrease the memory requirements. Instead of storing all the hash values in memory, the hash values are organized in chains. Only the first and the last element of the chain are stored in the memory, saving memory at the cost of cryptanalysis time. Additional memory trade off is acquired by reducing the hash values into keys.

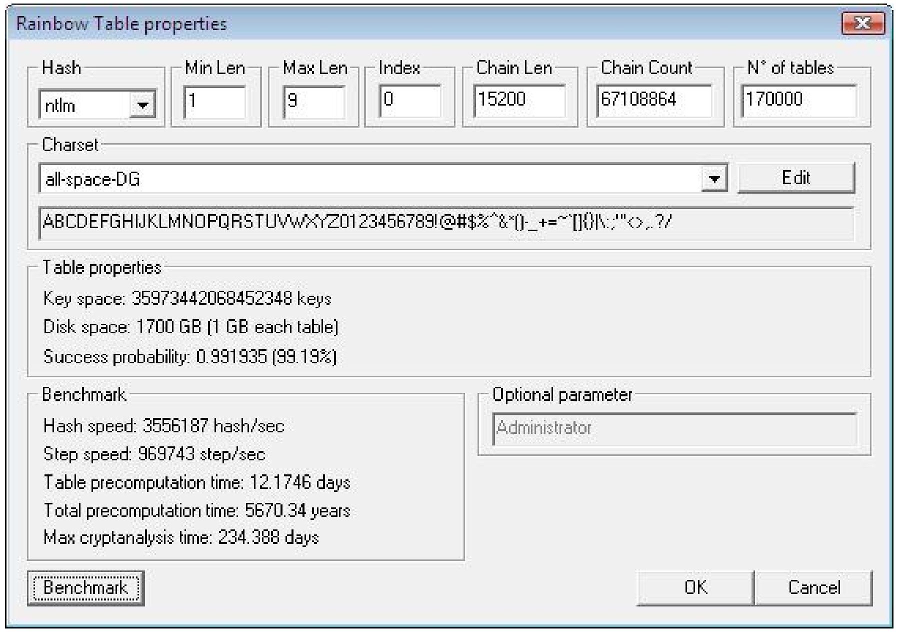
In Figure 1, the hash value *DFGA* is subject to cracking. In the top right corner of the figure, DFGA is reduced with a reduction function and the key is looked up in the rainbow tables. The key had no match with any of the endpoint keys, and on the next line the key is hashed again and then further reduced with a new reduction function. The lookup process now finds a match and from the corresponding startpoint key *abcd* through a number of reduction functions the desired key *oier* is derived.

Instead of using a single reduction function R, Oechslin [2] suggest using a sequence of related reduction functions R_1 up to R_{t-1}. He called the new chains *rainbow chains*, and the tables containing these chains *rainbow tables*.

There are many reasons for having rainbow tables available. One reason is that people forget their password to their user account and desperately want to access their files again. Then you can crack the hash value using rainbow tables to recover the lost password. Another reason can be to check the strength of a password before use. If the chosen password is revealed by the use of rainbow tables, you might be asked to choose a stronger password. You can either obtain precomputed rainbow tables, or you can make you own tables, using for instance the W*inrtgen* [3] or the *rtgen* [4] tool.

Approach

This approach will give an indication of how long time it will take to produce rainbow tables that can be used to crack NT hash values. The tool named *Winrtgen* will be used in this approach. This approach may also be


Rainbow Table properties

Hash: ntlm
Min Len: 1
Max Len: 9
Index: 0
Chain Len: 15200
Chain Count: 67108864
N° of tables: 170000

Charset: all-space-DG
ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$%^&*()-_+=~`[]{}|\:;"'<>,.?/

Table properties
Key space: 35973442068452348 keys
Disk space: 1700 GB (1 GB each table)
Success probability: 0.991935 (99.19%)

Benchmark
Hash speed: 3556187 hash/sec
Step speed: 969743 step/sec
Table precomputation time: 12.1746 days
Total precomputation time: 5670.34 years
Max cryptanalysis time: 234.388 days

Optional parameter: Administrator

Figure 2: The *winrtgen* tool.

applied to generate rainbow tables for other types of hash values. The computer used when these estimations were done was a Dell OptiPlex GX270 Pentium 4 2.60 GHz 1 GB.

When producing rainbow tables you have to consider the following settings: Type of hash, minimum and maximum password length, chain length, chain count, number of tables and charset. As can be seen from Figure 2, *ntlm* was chosen to be the hash type, the maximum password length has been set to 9 characters, the rainbow length was set to 15200, the chain count to 67108864 and the number of tables that are going to be generated was 170000. The figure below use the charset named *all-space-DG* but other charsets can be chosen.

As already mentioned, Figure 2 shows W*inrtgen* with settings that might be used to create the rainbow tables used to crack the NT hash values. From the *Benchmark* menu at the lower left of the window different calculations are shown. Note that the *Total precomputation time* is estimated to over 5670 years on the system used in this approach and the *Max cryptanalysis time*, which is the maximum time it might take to crack the hash value, is estimated to over 234 days. Performing this approach on the existing system is therefore undesirable. A requirement of 99% or more in *Success probability* led to this estimate. The requirement was achieved with the current settings, as can be seen in Figure 2. As a consequence of the process taking too much time on the current system, an interesting study would be to see how long time the same process would be estimated to take on a supercomputer. If the estimated time seems acceptable, it would also be interesting to actually run this process. But if a supercomputer is used it may be beneficial to change the chain length, chain count and number of tables. If you increase for example the chain length, the result will be that the rainbow table will decrease in size but on the other hand the new chain requires a more powerful processor. The idea is to still gain adequate *Success probability*. It should be noted that it is also possible to use a tool named *rtgen* for this process. *rtgen* is available for UNIX system.

Goal
The thought behind this presentation is to highlight the fact that the time spent to generate such rainbow tables most likely will be dramatically reduced with the use of a supercomputer. If you compare the computation times of generating rainbow tables on our system and a supercomputer, the amount of rainbow tables will be dramatically larger when using a supercomputer. This will lead to a significantly greater chance of revealing the desired password.

Conclusion

As a system administrator for a large corporation it would be an advantage to have a large set of rainbow tables, to recover and/or test the employee’s passwords. Based on this approach it can be concluded that building the desired rainbow tables will take too much time on our system. This is why it is desirable to test this process on a supercomputer, and then see how long time it will take.

References

[1] Martin E. Hellman. *A cryptanalytic time-memory trade-off*. IEEE TRANSACTIONS ON INFORMATION THEORY, IT-26:401-406, 1980. Available from: http://wwwee.stanford.edu/~hellman/publications/36.pdf

[2] Philippe Oechslin. *Making a Faster Cryptanalytic Time-Memory Trade-Off*, 2003. Available from: http://infoscience.epfl.ch/getfile.py?recid=99512&mode=best

[3] Montoro M. W*inrtgen*, 2009.
Available from: http://www.oxid.it/projects.html

[4] Project RainbowCrack, RainbowCrack v.1.3, 2009.
Available from: http://project-rainbowcrack.com/

All in a day's work: Password cracking for the rest of us

Jørgen Blakstad[1], Rune Walsø Nergård[1], Martin Gilje Jaatun[2], and Danilo Gligoroski[1]

[1] Department of Telematics, NTNU, NO-7491 Trondheim , Norway
[2] SINTEF ICT, NO-7465 Trondheim, Norway
{jorgenbl,runewals}@stud.ntnu.no

Abstract. The majority of computer systems are still protected primarily with a user name and password, and many users employ the same password on multiple systems. Additionally, some of the most popular operating systems such as Windows XP, Windows Vista and the upcoming Windows 7, still use ad-hoc constructed hash functions such as LM and NT, while many Linux variants use the long time broken hash function MD5. This paper describes an experiment where we have tested the strength of a selection of passwords when converted to LM, NT and MD5 hashes, respectively, using commonly available tools. Our conclusion is that a large number of passwords can be cracked within a normal working day, and that all LM hash passwords can be recovered before morning coffee. The use of such weak hash functions in the process of user authentication in these operating systems poses a significant threat to an organization's security.
Keywords: Password, Security, Cracking, LM Hash, NT, MD5

1 Introduction

Most computer systems are still protected primarily by a username and password combination. Using passwords and password management routines for giving access rights is a technique that is as old as the history of operating systems. Generally speaking the development in the way how passwords were kept and used in operating systems went through these phases:

1. Keep passwords in pure text form [1].
2. Encrypt passwords with some naive encrypting algorithms or modifications of DES and keep just the encrypted parts [2, 3].
3. Encrypt passwords with DES and keep just the encrypted parts [2, 3].
4. Pre-pend the passwords with some added random value called ”salt” and then encrypt them with DES. Keep just the encrypted parts [2].
5. Hash the passwords with some cryptographic hash function and keep just the hash digests [4–6].
6. Pre-pend the passwords with some added random value called ”salt” and then hash them with some cryptographic hash function. Keep just the hash digests [7, 8].

It is a common understanding in the information security community that the techniques that combine salting and hashing of the passwords (item 6 in the previous list) are much more secure than the techniques described in items 1 – 5. The security of the techniques under items 5 and 6 also depend on the security of the cryptographic hash function employed. If the used hash function is weak (i.e. it is easy to find preimages, second preimages or collisions for that hash function), then protection of the passwords by that hash function is also weak.

In the area of cryptographic hash functions the most popular family of hash functions is the MD4 family, which includes functions like MD5, HAVAL, RIPEMD, RIPEMD-160, SHA-0, SHA-1, SHA-2 and many others [9–11]. Many of those cryptographic functions have been practically broken (such as MD4, MD5, HAVAL, RIPEMD, SHA-0), and SHA-1 was theoretically broken in 2005 [12].

However, it is a shocking and disappointing fact that currently the most popular operating systems still use weak cryptographic hash functions (ad-hoc constructed hash functions that do not belong even to the MD4 family) and many of them do not even use salting techniques. Typical examples are Microsoft Windows XP that uses an ad-hoc hash solution called "LM hash", the more recent Microsoft Vista and the latest Windows 7 operating systems that use an ad-hoc hash function called "NT hash" (seen as an improved version of LM hash), and some versions of the popular Linux operating system Ubuntu that use the long time broken MD5 hash function.

Motivated by the fact that those most popular operating systems are still using very weak and practically broken hash functions for dealing with passwords and user authentication techniques, we chose 30 passwords of different strengths, and wanted to see how many could be cracked during a period of *1 day* (8 hours). In this experiment dictionary attacks and attacks with the use of rainbow tables, which are both password cracking techniques, were applied.

In our experiments we have included exactly the three aforementioned types of hash functions: LM hash, NT hash and MD5 hash.

The remainder of the paper is structured as follows: In Section 2 we give a brief description of our laboratory environment. Section 3 describes how our experiment was carried out, with results presented in Section 4. We discuss the results in Section 5, and offer our conclusions and suggestions for further work in Section 6.

2 Laboratory Environment

We made fresh installations of Windows XP, Windows Vista, Windows 7 RC and Ubuntu 8.10 on identical systems. It should be added that we installed an extra hard disk to be able to store the rainbow tables used for the LM hashes, as the rainbow tables were 64GB in size. The dictionary and the rainbow table attacks on the LM- and NT hashes were run in parallel. This was done on two Windows Vista systems. The dictionary attack that was run on the MD5 hashes was not run in parallel with any of the other attacks. The MD5 dictionary attack was run on Ubuntu 8.10.

The rainbow tables used to crack the LM hash were much smaller than those for other hashes because of their weak cryptographic properties (as there are less combinations to try). Oechslin [13] also used the LM hash as an example when he introduced the concept of rainbow tables. We chose to include the NT hash because this is the new hash type included by Microsoft on newer Windows versions. The NT hash is supposed to solve many of the weaknesses of the LM hash, but still no salt is included. This is why the NT hash also is susceptible to the rainbow table attack. The MD5 hash was chosen because it is a common type of hash used in Unix systems. MD5 hashes are supported by John the Ripper [14], which is the tool that we used for the MD5 hashes in this experiment, and they may be cracked with the use of a dictionary attack. At RainbowCrack's web pages [15] rainbow tables for MD5 are available but we did not include these in the experiment due to time constraints.

3 Experimental Procedure

Table 1 gives an overview of which techniques (dictionary attack and/or rainbow table attack) that were applied to what type of hash. For all the dictionary attacks performed we used the wordlist that is already included in the Cain & Abel tool [16]. In the case where John the Ripper was used for the dictionary attack the same wordlist was imported. For the LM hashes we used rainbow tables specifically designed to crack LM hashes and these tables, which in all consists of 64 subtables, were in total 64GB in size. The LM rainbow tables were downloaded for *free* from [17].

For the NT hashes we used rainbow tables that were specifically designed to crack NT hashes but the tables we used for this purpose were not free. It is possible to obtain free NT rainbow tables from Ophcrack's web pages [18], such as the one named *Vista free*, but these do not include as many characters as the versions you have to pay for. The NT rainbow tables that we bought are called *Vista special*, are 8GB in size and were obtained from Ophcrack's web page as well. The *Vista special* rainbow tables in all consists of 4 rainbow tables.

Type of hash	**Techniques**
LM hash	Dictionary attack and rainbow table attack
NT hash	Dictionary attack and rainbow table attack
MD5	Dictionary attack

Table 1. Applied techniques corresponding to their type of hash.

3.1 Selection of Passwords

In this experiment we wanted to test passwords of different strength. That is why we created three different password groups, and then added 10 password to each group. *Most Popular*, *Mnemonic Passwords* and *Random Passwords* are the names of these groups. The ten passwords added to the *Most Popular* group are taken from Bruce Schneier's web page [19], 8 of the *Mnemonic Passwords* are taken from a mnemonic password generator [20] while 2 are created by us. The *Random Passwords* are collected from a random password generator [21] found on the Internet. The group labeled *Most Popular* contain some of the most common passwords used, and as we can see from Table 2, these passwords do not seem very secure. As a contrast, the *Random Passwords* are considered very secure because they are randomly generated with both lower-case- and upper-case letters, numbers and special characters. One disadvantage with the randomly generated passwords are that they are hard to remember, and that is why we have included some mnemonic passwords. A mnemonic password is a password where a user for example chooses a phrase that is easy to remember (mnemonic) and uses a character to represent each word in the phrase. Mnemonic passwords are therefore passwords that seem like random generated passwords but are easier to remember because they are generated from phrases that are easy to remember. As already mentioned, we chose to create two of the mnemonic passwords ourselves (5GCTw4tG@N and tcC!tW84s), and this was to also include mnemonic passwords of nine and ten characters in length. It was desirable to test passwords with different characteristics. As an example of a mnemonic password we can take a look at one of our self-created password: *5GCTw4tG@N - Fifth Grade Comm Tech waiting for their Graduation at NTNU.* An overview of the 30 different passwords and to which group they belong is given in Table 2. Table 3 lists all the selected *Mnemonic Passwords* and their corresponding phrase.

Most Popular	**Mnemonic Passwords**	**Random Passwords**
password1	hm71li	qaSt4@
abc123	k0fzug	vANe$a
myspace1	fp6jar	!aFu9ut
password	msi89a0	C7e++AV
blink182	%l41pc	w2U$atHe
qwerty1	m.f0vgk	$_Ch6cU5
fuckyou	v*qbt4un	Ke42A2Pe*
123abc	qtdra#	rA3$_ey*c
baseball1	5GCTw4tG@N	b*#D9*7yEG
football1	tcC!tW84s	8rA*_pHa$8

Table 2. An overview of the 30 selected passwords, and their corresponding category.

Mnemonic Passwords	**Phrase**
hm71li	Hefin manages Sven's first lugubrious identity
k0fzug	Kath's nothing frazzles Zoe's unexpected gadget
fp6jar	Frank promotes Zach's jocular absentminded rabbit
msi89a0	Morgan seizes if eighth Nina argues nothings
%l41pc	Percy liberates Fourier's first priceless cabbage
m.f0vgk	Morgan stops for only Vivian glamorises kilns
v*qbt4un	Victor's handy quarter boosts Tim's fourth unknown number
qtdra#	Quentin traps Dave's reputable aloof hash
5GCTw4tG@N	Fifth Grade Comm Tech waiting for their Graduation at NTNU
tcC!tW84s	The current crisis in the world fights for survival

Table 3. 10 mnemonic passwords with their corresponding phrase.

Type of hash	**Tools**
LM hash	Cain & Abel
NT hash	Cain & Abel, Ophcrack
MD5	John the Ripper

Table 4. An overview of the tools used

3.2 Applied Methodologies for Password Cracking

For the LM and NT hashes, we first applied a dictionary attack to exclude some of the weakest passwords before we started to crack the rest of the passwords with the use of rainbow tables. For the MD5 hashes we only applied a dictionary attack. We created 30 users on a Windows XP machine, 30 users on a Windows Vista machine and 30 users on a Ubuntu 8.10 machine to be able to attack all of the three different types of hashes (LM hash, NT hash and MD5 hash). The user accounts, and thus also the corresponding password files, were created on virtual machines using a program called VirtualBox [22]. We used VirtualBox because it is easy to revert the actions done on a virtual machine. To create this many users we made scripts to automate this process. The script for adding users in a UNIX system adds 30 users with their passwords hashed with MD5, The result was a shadow file created in Ubuntu 8.10 containing 30 MD5 hash values.

When we planned this experiment, and before we knew how long time the dictionary and the cryptanalysis attack would take, we agreed to stop the dictionary attacks on the LM hashes and the NT hashes (the SAM files) after maximum 2 hours and then use the remaining 6 hours for the rainbow tables (8 hours in total). The reason why we planned with such time constraints was that we did not expect all the passwords to be revealed and that we therefore had to cancel the jobs before completion, as the idea was to only use *1 day*. We also wanted the cryptanalysis attack to get most of the available time as we thought that the chance was bigger that the rainbow table attack would crack the more difficult passwords. When it comes to the MD5 hashes the dictionary attack was supposed to run for 8 hours continuously, and then be aborted if it had not completed by then.

4 Results

In this section the achieved results regarding the empirical password study will be presented. Keep in mind that this experiment was supposed to reveal how many of the 30 selected passwords that were possible to reveal during 1 day (8 hours).

4.1 Timing

Phase 1 of this experiment was a timing phase used to measure how long time both the dictionary attack and the rainbow table attack with the use of rainbow tables would take when trying to crack the LM and NT hash. This phase also measured the time of the dictionary attack on the MD5 hash. The results from this timing phase is presented below. We still separate between LM hash, NT hash and MD5 hash.

LM hash For the LM hash we used Cain & Abel [16] for both the dictionary and the rainbow table attack, and for the former we used a wordlist (Wordlist.txt) which is included by default in the Cain & Abel tool. When the LM hash is constructed, the password is first divided into two segments with maximum 7 characters in each segment. The two segments are then hashed separately. The password used in this phase consists of 10 characters and therefore 2 LM hash values are constructed, one for the 7 first characters and one for the remaining 3 characters. As expected, none of these two hash values was cracked after running the dictionary attack, and thus the password was not revealed.

For the dictionary attack we had to open the *Windows Task Manager* and use the *CPU Time* to measure the time. The total time for the dictionary attack *1 minute and 8 seconds* (68 seconds). This time measurements show how long time it may take if the password is not found during the dictionary attack.

Now let us take a look at the timing of the *rainbow table attack*. We used only one table to be able to create an estimate instead of running the whole attack, as this would take longer time than we wanted to spend on the timing phase. The rainbow table attack, using just one of the tables, took 291,50 + 49,91 = 341,41 seconds = 5 minutes 41 seconds, when the password was not revealed by this single rainbow table. When calculating the total time for the single rainbow table we included *Total disk access time* and *Total rainbow table time*. Now we had to make an estimate for the maximum time the rainbow table attack might take when including all the 64 rainbow tables: 5 minutes 41 seconds * 64 tables = 364 minutes 10 seconds (21850 seconds) = *6 hours 4 minutes*. This estimate illustrates the worst case scenario, if no password is found during the rainbow table attack.

Table 5 gives an overview of the timing results from the dictionary attack and from the estimate of the rainbow attack. The latter is based on the accomplishment of one of the 64 rainbow tables. Both these attacks were performed on one password (8rA*_pHa$8).

The idea with the timing phase was to find out how long time the dictionary attack and the rainbow table attack might take, and if it would be advantageous to include a dictionary attack to reveal the easiest passwords before running the rainbow table attack with the rainbow tables. Table 5 gives an indication of how much time the different attacks might take. The dictionary attack only constitute 0.24% of the available time and the rainbow table attack is estimated to constitue 75.87% of the time. This leaves us with as much as 23.90% unused time.

NT hash For the NT hash we used Cain & Abel [16] for the dictionary attack and Ophcrack for the rainbow table attack. Let us first consider the dictionary attack. We used the same wordlist (Wordlist.txt) as for the LM hash.

The dictionary attack was measured to take 54 seconds, but from this we had to deduct 5 seconds as this was the time it took to start Cain & Abel. This makes the total time for the dictionary attack *49 seconds*. This gives an indication of how long time it might take if the password is not found during the dictionary attack.

The rainbow table attack on the NT hash, using just one of the tables, took *8 minutes 20 seconds*. The field displaying the time is labeled *Time elapsed*. In this case we do not need to deduct anything as Ophcrack has a build-in timing function which starts when the cracking process begins, and not when the program is loaded. As this was just an estimate when using one of the tables, we have to make an estimate of how long time it might take when all the 4 rainbow tables are included. The 4 NT rainbow tables mentioned here are the tables that together form the non-free *Vista special* tables. The total estimate for the use of rainbow tables on the NT hash is: 8 minutes 20 seconds (500 seconds) * 4 tables = *33 minutes 20 seconds* (2000 seconds). This estimate illustrates the case if no password is found during the rainbow table attack on the NT hash.

Table 5 gives an overview of the timing results from the dictionary attack and from the estimate of the rainbow table attack performed on the NT hash value. Both these attacks were performed on one password.

MD5 hash For the MD5 hash we only performed a dictionary attack, and we used John the Ripper to find out how long time the dictionary attack would take on the selected password (8rA*_pHa$8). To start the dictionary attack we had to run a couple of commands:

1. umask 077
2. sudo unshadow /etc/passwd /etc/shadow > mypasswd
3. sudo john −−wordlist=Wordlist.lst −−rules mypasswd

First, the *umask* command was used to set adequate permission rights. Next we used the *unshadow* command to obtain the content of the passwd file and of the shadow file, and then store the content in a new file named *mypasswd* in this case. The third command, *john*, was used to actually start the dictionary attack using John the Ripper. As can be seen from the third command we used the same wordlist as for both the LM hash and the NT hash. By using $--rules$ in the same command we also specified that we wanted to use certain mangling rules for the dictionary attack. Cain & Abel, which was used for both the LM and NT hash, also used certain mangling rules. The command *sudo* was used to get administrator privileges. Note that these commands also have to be applied in Phase 2 to run the dictionary attack on all the 30 users.

We used the timing function included in the John the Ripper tool, and the figure shows that the dictionary attack, including the word mangling, took *51 minutes and 48 seconds* (3108 seconds) to complete on the selected password.

Table 5 gives an overview of the timing results from the dictionary attack performed on the MD5 hash value. Note that the dictionary attack was performed on just one password.

Hash	**Technique**	**Time**
LM	Dictionary attack	1 minute 8 seconds (68 seconds)
LM	Rainbow attack	6 hours 4 minutes (21850 seconds)
NT	Dictionary attack	49 seconds
NT	Rainbow Table attack	33 minutes 20 seconds (2000 seconds)
MD5	Dictionary attack	51 minutes 48 seconds (3108 seconds)

Table 5. Time measurement for the LM, NT and MD5 hash

4.2 Password Cracking

The LM hashes were obtained from a Windows XP machine, the NT hashes from a Windows Vista machine and the MD5 hashes from a Ubuntu 9.05 machine. We also performed an experiment on the side for the NT hashes, with the use of a machine running the newly released Windows 7 RC operating system. This was to be able to highlight possible improvements that Microsoft had done regarding the handling of the login password. A full outline of the selected passwords can be found in Table 2.

LM hash Note that a password consisting of 7 characters or less will only have one LM hash value.

Nine passwords were revealed from the dictionary attack, and are listed in italics in Table 8. 30% av the passwords were revealed from the dictionary attack, while 70% of them will be a subject for the rainbow table attack.

We estimated the time for the dictionary attack on the one selected password to be 1 minute and 8 seconds. The dictionary attack carried out with all the 30 selected passwords was measured to take 1 minute and 9 seconds. But also in this case we had to deduct the time it took for Cain & Abel to start as we used the *CPU Time* displayed in *Windows Task Manager*. Remember that the CPU time may not be completely accurate because of the influence from other processes. After deducting the 6 seconds that Cain & Abel used to start, the dictionary attack was estimated to take *1 minute 3 seconds*. This shows that it does not take longer time to run a dictionary attack on a password file containing several LM hashes than a password file containing just one LM hash. In fact, the estimate indicates that the case with several LM hashes takes shorter time than when cracking just one LM hash, but be aware of that minor inaccuracy may influence the result.

Now let us consider the *rainbow table attack* with the use of the LM rainbow tables. It should be noted that the users with a password that was revealed during the dictionary attack was moved from the list of passwords we wanted to reveal during the rainbow table attack. Examining the "Revaled LM?" column in Table 8, we can see that all the remaining *21 passwords* were revealed.

This means that we have revealed *all* the 30 selected passwords by using both a dictionary attack and a rainbow table attack on the LM hashes.

Even though we managed to reveal the 21 remaining passwords we still have to calculate how much time the rainbow table attack took to complete. To calculate the time we included *Total disk access time* and *Total cryptanalysis time* as reported by the Cain & Abel tool. The rainbow table attack, using all the rainbow tables, took 1724.58 + 6369.71 = 8094.29 seconds = *2 hours 14 minutes*.

Table 6 gives an overview of the timing results from the dictionary attack and from the rainbow table attack. Recall that the rainbow table attack using the LM rainbow tables on the selected password was estimated to take 6 hours 4 minutes. The dictionary attack performed in this phase was carried out on all the 30 selected passwords, while the remaining 21 hash values were subject to the cryptanalysis attack.

Technique	**Time**
Dictionary attack	1 minute 3 seconds (63 seconds)
Rainbow Table attack	2 hours 14 minutes (8094 seconds)

Table 6. Phase 2: Time measurements for the LM hashes

The dictionary attack constitutes 0.2% of the available time of 8 hours, and the rainbow table attack constitues 28% of the time. This leaves us with 71.68% unused time.

NT hash We used Cain & Abel for the dictionary attack on the NT hashes, while Ophcrack was used for the rainbow table attack.

A complete list of all the revealed passwords from the dictionary attack is given in the "revealed NT?" column in Table 8 (indicated by *Yes* in italics).

We are going to point out some differences from the case with the LM hashes. Firstly, note that *8 passwords* were revealed from the dictionary attack.

At this point we experienced a minor difference between the SAM file obtained from the Windows Vista machine and the SAM file from the machine running Windows 7 RC. When we imported the SAM file that originated from Windows 7 RC we noticed that the loaded NT hash values were different than those that were loaded when using the Windows Vista SAM file, even though the same user account passwords were utilized. And when we tried to run the dictionary attack no passwords were revealed. At this point we wondered if Microsoft had improved their password handling for Windows 7 RC by for instance including a salt when generating the NT hash value. This turned out to not be the case. We tried to import the Windows 7 RC SAM file into Ophcrack to see if the hash values still were different from the hash values in the Windows Vista SAM file, and in Ophcrack they were the same. This sat aside our theory about improved password security. When we had loaded the Windows 7 RC SAM file in Ophcrack we chose *Save* from the top menu and then *Save to file*. The saved file was then imported into Cain & Abel, which was the tool used for the dictionary attacks. Now, the hash values were the same for both the Windows 7 RC SAM file and Windows Vista SAM file.

We estimated the time for the dictionary attack on the NT hash for the selected password to be 49 seconds. The dictionary attack carried out in this phase with all the 30 selected passwords was measured to take 58 seconds. After deducting the 5 seconds it took for Cain & Abel to start, as we used the *CPU Time* displayed in *Windows Task Manager* to measure the time, the dictionary attack was measured to take *53 seconds*. This shows that it does not necessarily take significantly longer time to run a dictionary attack on a password file containing several NT hash values than it does with a password file containing just one NT hash value.

Let us now consider the *rainbow table attack* with the use of the NT rainbow tables. The users with a password that was revealed during the dictionary attack were moved from the list of passwords we wanted to reveal during the rainbow table attack. *10 passwords* of the 22 remaining passwords were revealed from the rainbow table attack. In other words, the rainbow table attack with the NT rainbow tables was able to reveal 45.5% of the 22 passwords that were subject to the rainbow table attack.

The results presented above show that 8 passwords were revealed during the dictionary attack and 10 passwords were reveled during the rainbow table attack, meaning that 18 of the 30 selected passwords were revealed in total. Table 8 list all the 30 passwords to give an overview of which of the selected passwords that were revealed. This can be seen from the column labeled *NT Revealed?*. The 18 passwords that were found are listed with *Yes* in the *NT Revealed?* column. Note that the 8 passwords revealed during the dictionary attack have *italic* fonts. This means that the passwords revealed during the rainbow table attack is marked with “Yes” in the *NT Revealed?* column but do **not** have their font in *italic*. 12 of the 30 passwords were not revealed.

As already mentioned, we have been able to reveal 18 of the 30 selected passwords by using both a dictionary attack and a rainbow table attack on the NT hash values. 27% of the 30 selected passwords were revealed during the dictionary attack while 33% of the passwords were revealed during the rainbow table attack. This means that 40% of the NT hash values were not cracked.

The rainbow table attack took *4 hours 26 minutes.* The rainbow table attack using the NT rainbow tables was estimated to take 33 minutes and 20 seconds. Table 7 gives an overview of the timing results from the dictionary attack and from the rainbow table attack. The dictionary attack performed in this phase was carried out on all the 30 selected passwords, while the remaining 22 hash values were subject to the cryptanalysis attack.

Technique	**Time**
Dictionary attack	53 seconds
Rainbow Table attack	4 hours 26 minutes (15960 seconds)

Table 7. Time measurements for the NT hashes

The dictionary attack constitutes 0.18% of the available time and the rainbow table attack constitues as much as 55.42% of the time. This leaves us with 44.40% unused time.

It should be added that we got almost the same results when using the Windows 7 RC SAM file as we did when we used the Windows Vista SAM file. The results from the dictionary attack was identical, and the only difference was that the rainbow table attack performed on the Windows 7 RC SAM file took 5 hours 15 minutes and 28 seconds to complete while, as already presented, the same attack took 4hours and 26 minutes on the Windows Vista SAM file. The exact same passwords were revealed in both cases. This means that the NT hash values stored in the Windows Vista SAM file and the Windows 7 RC file are identical, and that Microsoft has not increased the security to the newly released Windows 7 RC operating system regarding the login passwords that are hashed and stored in the SAM file. Microsoft has for example still not included a salt when generating the NT hash values.

MD5 hash For the MD5 hash a dictionary attack was the only attack that was applied, using the John the Ripper tool for the whole 8 hours period.

The 8 passwords that were revealed during the dictionary attack constitute 27% of all the 30 selected passwords. This means that the remaining 22 passwords, which constitute the remaining 73%, were not found since the dictionary attack was the only attack performed on the MD5 hash values.

The results presented above show that 8 passwords (27%) were revealed during the dictionary attack, meaning that 22 of the 30 selected passwords were not revealed at all. Table 8 lists all the 30 passwords to give an overview of which of the selected passwords that were revealed, and which did not. This can be seen from the column labeled *MD5 Revealed?*.

Number	**Length**	**Revealed LM?**	**Revealed NT?**	**Revealed MD5?**
1	9	*Yes*	*Yes*	Yes
2	6	*Yes*	*Yes*	Yes
3	8	Yes	Yes	No
4	8	*Yes*	*Yes*	Yes
5	8	*Yes*	Yes	No
6	7	*Yes*	*Yes*	Yes
7	7	*Yes*	*Yes*	Yes
8	6	*Yes*	*Yes*	Yes
9	9	*Yes*	*Yes*	Yes
10	9	Yes	*Yes*	Yes
11	6	Yes	Yes	No
12	6	Yes	Yes	No
13	6	Yes	Yes	No
14	7	Yes	Yes	No
15	6	Yes	Yes	No
16	7	Yes	No	No
17	8	Yes	No	No
18	6	Yes	Yes	No
19	10	Yes	No	No
20	9	Yes	No	No
21	6	Yes	Yes	No
22	6	Yes	Yes	No
23	7	Yes	No	No
24	7	Yes	No	No
25	8	Yes	No	No
26	8	Yes	No	No
27	9	Yes	No	No
28	9	Yes	No	No
29	10	Yes	No	No
30	10	Yes	No	No

Table 8. The results obtained from the various attacks.

We estimated the time for the dictionary attack on the MD5 hash for the selected password to be 51 minutes and 48 seconds. The dictionary attack carried out in this phase with all the 30 selected passwords was aborted after 8 hours, as this was the time limit set for the experiment. This means that the attack did not complete. In other words, the dictionary attack took 100% of the available time.

5 Discussion

Figures from the company Net Applications [23] indicate that Windows XP still has more than 60% of the total market share for desktop operating systems. Although there are large uncertainties associated with these figures, it is clear that a very large proportion of laptops in the world today are running Windows XP, and these laptops by default will have LM hashes enabled. Our small experiment revealed that not only can all LM hashes on a computer be cracked in a day, but at two and a quarter hours it can even be performed before most office workers have finished their morning coffee.

The results from our attacks on the LM hash show that in this case it did not matter to which category the password belongs because all the passwords were revealed after the completion of both attacks. According to our intention, almost all of the weakest passwords were revealed during the dictionary attack. As this was the purpose of running the dictionary attack, this part of the experiment has to be considered successful. It is however important to add that in this case it was not really necessary to run the dictionary attack at all, since the character set used for the cryptanalysis attack included every character that is possible to employ in the password. It is possible to include so many characters in the character set and still be able to reveal the password in relatively short time because of the weaknesses related to the construction of the LM hash, e.g., the password is converted to uppercase and split into two segments before it is hashed. This reduces the number of combinations that has to be tried to reveal the password. In other words, this means that the rainbow table attack would have revealed all the 30 passwords.

The results from the rainbow table attack on the NT hash show that 10 passwords (33%) were revealed. After the completion of both the dictionary attack and the rainbow table attack, 18 passwords (60%) were revealed in total. This left us with 12 unrevealed passwords (40%). Most of the 10 passwords that were revealed through the rainbow table attack on the NT hash belongs to the category named *Mnemonic Passwords*. Actually 6 of the 10 mnemonic passwords were revealed, and in addition also one password from the *Most Popular* category and two passwords from *Random Passwords*. The 4 remaining password from the *Mnemonic Passwords* category that were not revealed is *m.f0vgk* (7 characters), *v*qbt4un* (8 characters), *5GCTw4tG@N* (10 characters) and *tcC!tW84s* (9 characters). The last two are the passwords that were produced by us, and both of them have corresponding phrases that are easy for us to remember even though they consist of many characters. The maximum password length that are supported by the *Vista special* tables is 8 characters, and the corresponding character set are quite limited as only numbers from 0 to 9 and lowercase letters are supported. As *5GCTw4tG@N* is 10 characters long it was expected that this password was not revealed during the rainbow table attack. The *tcC!tW84s* password consists of 9 characters. This is also too long for the *Vista special* tables to crack. The other two passwords (*m.f0vgk* and *v*qbt4un*) that were not revealed by the *Vista special* rainbow tables are both within the supported password length of the *Vista special* tables, but they have different character sets because they have different length. Both of these passwords consist of lowercase letters, one number and one special character. The only thing that prevents both of them from being revealed by the rainbow tables is the special characters. We have now considered the only four passwords from the *Mnemonic Passwords* category that were not revealed by the rainbow table attack. This was to highlight that the characters that these passwords consisted of were not included in the character set. All the revealed passwords consist of characters that are included in the character set.

If we take a look at the *Random Passwords* category there were only two passwords (*qaSt4@* and *vANe$a*) that were revealed by the rainbow table attack. The reason why these two passwords were revealed is because they both consist of only 6 characters. If they had been one character longer, they would not have been revealed because the character set for passwords of 7 characters in length do not include special characters. Both of the passwords contain a special character. They are also the only passwords in the *Random Passwords* category that are just 6 characters in length, and the other passwords are not revealed by the rainbow table attack.

The two passwords *myspace1* and *blink182* not revealed by the dictionary attack, were revealed by the rainbow table attack. Both *myspace1* and *blink182* consist of 8 characters, which is the longest password length that is supported by the *Vista special* tables. The reason why both these passwords were revealed is because they only consist of lowercase letters in addition to numbers. If for example only one of the characters had been an uppercase letter the passwords would not have been revealed. It has now been shown that small differences might determine whether a password is going to be revealed or not.

The same results were obtained when the dictionary attack and the cryptanalysis attack were performed on the SAM file that originated from the Windows 7 RC operating system. This means that also the same discussions applies for the NT hashes stored in the Windows 7 RC SAM file.

If we were supposed to compare the use of a dictionary attack with a rainbow table attack we would highlight that while the dictionary attack requires a relatively small wordlist, the rainbow tables are generally a lot bigger, requiring a quite large storage capacity. Rainbow tables must either be downloaded or generated in advance of the attack, and this may take quite a long time depending on your download speed and the system you are using. It is important to be aware that attackers with ill intentions probably would invest a big effort to recover the password.

Another observation is that when it comes to cracking passwords with the use of rainbow tables, it does not matter if the password is strong or weak as long as the password is within the requirements sat for the particular rainbow tables.

The results obtained from the empirical password study show that even though it is important to choose a strong password, this may not be sufficient by itself. The underlying system does also influence how easily the password can be recovered. A good example of this is Windows XP, which utilizes the LM hash together with the NT hash. Because of the weaknesses with the construction of the LM hash, the rainbow tables can support a character set consisting of all the characters necessary to reveal all the passwords. The results obtained from our study, showing that all the 30 selected passwords may be revealed using rainbow tables, confirms this statement. It may also be confirmed by investigating the character set as well. To avoid the case illustrated in this example you should either disable the use of LM hash or you should use a password longer than 14 characters if you are using Windows XP. When the password is 15 characters or longer an operating system that uses both LM hash and NT hash will not store the LM hash, only the NT hash.

This study shows that a big portion of the selected passwords were revealed, even with computers that are not particularly powerful. As the storage capacity and processor power of mainsteam computers keeps increasing, more computations can be performed in a shorter amount of time. Even though the use of a salt in the generation of the password hash will probably offer sufficient protection from password recovery for a while, it will only be a temporary respite. It has been generally known even before Klein's studies [24] that it is important to choose a good password – our findings indicate that in the near future, it will not matter how good your password is; it'll get cracked anyway.

6 Conclusions and Further Work

The password as an authentication mechanism is headed for obsolence, as the password lengths required to thwart ranibow table attacks are rapidly approaching unmanageable (or unrememberable) proportions. If you are still using the LM hash on your laptop, you might as well put your passord in a cleartext file and call it "password.txt" - according to our results, anyone who wants your password will have it by the end of the working day. Even though we were not able to crack all the NT hashes, it seems that it is only a matter of time before rainbow tables for all practical password lengths will be generally available also for the NT hash.

An obvious opportunity for further work would be to employ rainbow tables for MD5 to see how a vanilla Linux distribution bears up under a cracking attack. More fundamental contributions could be made in finding alternatives to passwords that are more secure, but with the same level of social acceptance and ease of use.

Acknowledgements

The research for this paper was conducted as part of a MSc thesis at the Norwegian University of Science and Technology (NTNU).

References

1. M. Wilkes, *Time - Sharing Computer Systems.* New York, NY, USA: American Elsevier, 1968.

2. R. Morris and K. Thompson, “Password security: a case history,” *Commun. ACM*, vol. 22, no. 11, pp. 594–597, 1979.
3. M. Luby and C. Rackoff, “A study of password security,” *J. Cryptol.*, vol. 1, no. 3, pp. 151–158, 1989.
4. E. Glass, “The ntlm authentication protocol and security support provider,” sourceforge.net, 2006. [Online]. Available: http://davenport.sourceforge.net/ntlm.html#theLmResponse
5. M. G. Jaatun, “Lmhash spoofing vha samba klient,” *FFI note*, 2000.
6. J. M. Johansson, “The great debates: Pass phrases vs. passwords. part 1 of 3,” 2004. [Online]. Available: http://technet.microsoft.com/en-us/library/cc512613.aspx
7. F. K. Developers, *FreeBSD Handbook*, 2009. [Online]. Available: http://www.freebsd.org/docs.html (June 2009)
8. U. Drepper, “Unix crypt using sha-256 and sha-512,” people.redhat.com, 2008. [Online]. Available: http://people.redhat.com/drepper/SHA-crypt.txt
9. R. Rivest, “The MD4 message-digest algorithm,” IETF Request for Comments (RFC) 1320.
10. ——, “The MD5 message-digest algorithm,” IETF Request for Comments (RFC) 1321, Internet Activities Board, Internet Engineering Task Force.
11. *Secure Hash Standard*, National Institute of Science and Technology Std. Federal Information Processing Standard (FIPS) 180-1.
12. X. Wang, Y. L. Yin, and H. Yu, “Finding collisions in the full SHA-1,” *Lecture Notes in Computer Science*, vol. 3621, pp. 17–36, 2005, Proceedings of Crypto 2005.
13. P. Oechslin, “Making a Faster Cryptanalytic Time-Memory Trade-Off,” in *The 23rd Annual International Cryptology Conference, CRYPTO ’03*, ser. Lecture Notes in Computer Science, vol. 2729, 2003, pp. 617–630. [Online]. Available: http://infoscience.epfl.ch/getfile.py?recid=99512&mode=best
14. O. Project, “John the ripper password cracker,” openwall.com, 2009. [Online]. Available: http://www.openwall.com/john/
15. RainbowCrack, “Project rainbowcrack,” project-rainbowcrack.com, 2009. [Online]. Available: http://project-rainbowcrack.com/
16. M. Montoro, “Cain & abel,” oxid.it, 2009. [Online]. Available: http://www.oxid.it/cain.html
17. “Rainbow tables,” shmoo.com, The Shmoo Group, 2006. [Online]. Available: http://rainbowtables.shmoo.com/
18. Ophcrack, “What is ophcrack?” sourceforge.net, 2009. [Online]. Available: http://ophcrack.sourceforge.net/
19. B. Schneier, “Myspace passwords aren’t so dumb,” schneier.com, 2006. [Online]. Available: http://www.schneier.com/essay-144.html
20. “The university’s password generator page,” aber.ac.uk, Aberystwyth University, 2000. [Online]. Available: http://users.aber.ac.uk/auj/portfolio/mnemonic.shtml
21. “Secure password generator,” pctools.com, PC Tools, 2009. [Online]. Available: http://www.pctools.com/guides/password/
22. “Virtualbox,” virtualbox.org, Sun Microsystems, Inc, 2009. [Online]. Available: http://www.virtualbox.org/
23. “Operating System Market Share.” [Online]. Available: http://marketshare.hitslink.com/
24. D. V. Klein, “Foiling the cracker: A survey of, and improvements to password security, (revised paper with new data),” in *14th DoE Computer Security Group*, May 1991. [Online]. Available: http://www.klein.com/dvk/publications/passwd.pdf