MSc thesis

Information security

Using hash values to identify fragments of evidence

Taking the concept of known file hash databases a step further.

Ketil Kintel
NISlab
Høgskolen i Gjøvik

ketil.kintel@hig.no

Oslo, August 27th 2004

Abstract

As more criminals utilize computer equipment to do mischief and storage on computers grow, an investigator is faced with an explosive growth in data which needs to be sifted through in order to find evidence. This makes investigation more and more time consuming and leads to less effective enforcement of law and order.

This thesis explores the possibilities of using hash values in identification of known evidence fragments. A method is sketched, tested and suggested as input in the process of deciding whether a full-blown forensic examination is warranted.

Sammendrag (Abstract in Norwegian)

Siden flere utnytter datautstyr til å begå kriminelle handlinger og lagringskapasiteten på slikt utstyr øker, fører dette til en eksplosiv økning i den datamengden som må gjennomgåes for å finne tekniske bevis. Det gjør at etterforskninger tar lengre og lengre tid og vi får en mindre effektiv håndhevelse av lov og orden.

Denne oppgaven ser på muligheten for å bruke hash-verdier til å identifisere bruddstykker av kjent bevismateriale. En metode er skissert, testet og foreslått som en faktor når man skal avgjøre om en fullstendig undersøkelse er berettiget.

.

Table of Contents

Abstract 2
Table of Contents 3
List of Tables 4
List of Figures 5
1 Introduction 6
1.1 Dangers of Superficial Investigations 6
1.2 Introduction to Digital Forensics 7
1.3 Slack Space and Other Residual Data 10
1.4 The Two Fundamental Problems of Digital Forensics 11
1.5 Uncertainty and Degradation of Digital Evidence 12
1.6 Problem Statement 13
1.7 Demarcation to Static Environments 14
1.8 Review of the State of the Art 14
1.9 Claimed Contributions 16
1.10 Agenda 17
2 Materials and Methods 18
2.1 Choice of Methods 18
2.2 Use of Hashing Algorithms 18
2.3 Generation of the Data Set 19
2.4 Generation of the Hash Database 22
2.5 Experiments on the Data Set 23
3 Results 24
4 Discussion 26
4.1 Selecting the Recommended Block Size 26
4.2 Accounting for Duplicate Hashes 26
4.3 Evaluation Against Proposed Guidelines 28
4.4 Levels of Uncertainty 30
4.5 Resistance to Tampering 30
4.6 Feasibility for a Real World Implementation 31
5 Conclusion and Further Work 32
5.1 Further Work 32
6 Acknowledgements 34
7 References 35
8 Appendices 40
8.1 Detailed result tables 40
8.2 Detailed database contents 48
8.3 Suggested introductory Literature 49
8.4 Practical Problems 50
8.5 Script and Program Listings 51
8.6 The Multiple Format Extraction Tool 7-Zip 65
8.7 Bill of Materials 65
Index 83

List of Tables

Table 1 – The 30 most common file extensions of the files in the test database 21
Table 2 – Identification potential of in the test data for full 512 byte (one sector) blocks 24
Table 3 – Other distribution of the duplicate hashes from the full 512 byte (one sector) blocks 25
Table 4 – Summary of requirement satisfaction for the proposed method. 30
Table 5 – Identification when using 512 byte hash blocks (full sector) 40
Table 6 - Identification when using 384 byte hash blocks 41
Table 7 - Identification when using 256 byte hash blocks (half sector) 41
Table 8 - Identification when using 192 byte hash blocks 41
Table 9 - Identification when using 128 byte hash blocks (quarter of sector) 42
Table 10 - Identification when using 96 byte hash blocks 42
Table 11 - Identification when using 64 byte hash blocks (double MD5 block size) 43
Table 12 - Identification when using 48 byte hash blocks 43
Table 13 - Identification when using 32 byte hash blocks (MD5 block size) 44
Table 14 - Identification when using 24 byte hash blocks 44
Table 15 - Identification when using 16 byte (128 bits) hash blocks (MD5 result size) 45
Table 16 - Identification when using 12 byte hash blocks 45
Table 17 - Identification when using 8 byte hash blocks 46
Table 18 - Identification when using 6 byte hash blocks 46
Table 19 - Identification when using 12 byte (32 bits) hash blocks (theoretical minimum for this experiment if pure random data in database) 47
Table 20 - Identification when using 3 byte hash blocks 47
Table 21 - Identification when using 2 byte (16 bits) hash blocks (Unusable) 47
Table 22 - Identification when using 1 byte (8 bits)hash blocks (Unusable) 48
Table 23 – Detailed distribution of source for the 1 025 856 files used in creating the database 48
Table 24 – All file extensions occurring more than 400 times in the database 49

List of Figures

Figure 1 - The distribution of origin of the files in the test data base. 20
Figure 2 - The identification potential for each of the block sizes in the experiment 25

1 Introduction

Computer technology has seen an exponential increase in data storage, and it is not uncommon with terabyte storage. With storage space, the number of files has grown in much the same manner, making a typical desktop computer contain up to 100.000 files. [NIST 03]

As the number of computer related crimes grows [CERT 03] [Kruse et al 01], more criminals utilize computer equipment to do mischief [Politidirektoratet 03] and crimes become more complex [Stephenson 03]; a strain is put on law enforcement.

The following stories provide some examples of what this may lead to.

1.1 Dangers of Superficial Investigations

In 2002, a man was arrested in the UK. The police searched his computer, found images describing sexual child abuse and then raised criminal charges against him.

In order to be acquitted, he had to hire investigators himself. These investigators did a more thorough examination of his computer than the police had done, and found a modem hijacking trojan that most likely was the source of the images in question [Schwartz 03].

The real damage for the UK man was not only that he was under false charges for almost a year, but that he was expelled from the society of his home town, was divorced and lost the visitation rights to his kids.

Currently, a Nordic police operation called Enea is running. 209 persons in Norway are suspected for possession of images of sexual child abuse, and have had their computer equipment seized. Because the centre of computer crime[1] does not have the capacity to investigate all the seized equipment, it is

[1] Politiets datakrimsenter, Oslo

mainly to be done by inexperienced officers from local police precincts [Olsen et al 04]. 70 of the arrested persons have not pled guilty to the charges. We can only guess if and how many of those are victims of similar events as the UK man.

In both these cases, the police was trying to ease the growing strain on investigative resources in a way we believe is wrong (at least in a Norwegian setting) - by sacrificing the depth and quality of the investigation.

Fairness is important, but it should work both ways - the guilty ones should be punished and the innocent ones should go free. This thesis is an effort to help preserving today's level of legal protection by increasing the knowledge about how more thorough and reliable evidence can be achieved in today's computer systems.

1.2 Introduction to Digital Forensics

The following chapters serve as a primer to those aspects of Digital Forensics important to understanding the underlying problems and potential impact of this work. Readers experienced in Forensic work may skip forward to chapter 1.6.

Digital Forensic is defined by [Gary 01] as:

> *The use of scientifically derived and proven methods toward the preservation, collection, validation, identification, analysis, interpretation, documentation and presentation of digital evidence derived from digital sources for the purpose of facilitating or furthering the reconstruction of events found to be criminal, or helping to anticipate unauthorized actions shown to be disruptive to planned operations.*

The use of the phrase "scientifically derived" is not accidental. The field has traditionally been a domain of practioners rather than researchers. This is supported by the fact that the very first Digital Forensic Research Workshop was held as late as 2001, and the very first issue of the International Journal of Digital Evidence was published as late as 2002.

Digital Forensics are performed in an investigative context to serve a specific objective which relates to the environment where the investigation takes place [Mocas 04]. The investigation process is supposed to reconstruct the incident and give answers to the What, Who, When, Where, How and Why questions.

One of the main legal requirements is that the process used to examine evidence must be reproducible. In addition, the information must possess the following characteristics [Palmer 02]:

> *Relevant and/or Material: Will this information assist decision-makers in their tasks?*
>
> *Credible and/or Competent: Is the information believable, trustworthy, and true and, if so, by what measure?*

To illustrate, the process of digital forensics may be compared to archaeology or arson investigation [Casey 03]. A regular case can be explained by archaeology. An archaeologist studies artefacts to find out what it is, when and where it was used in order to say something about the original context. If, however, the criminal has tried to conceal the crime and destroyed vital evidence, the process resembles more an arson investigation, where one must rely heavily on crime scene characteristics to understand what happened.

The more technical term for this is correlation, as defined by [Stephenson 02]:

Digital forensic correlation is the comparison of evidentiary information from a variety of sources with the objective of discovering information that stands alone in concert with other such information, or corroborates or is corroborated by other evidentiary information.

Regardless of the strategy of the investigation, the following analysis methods are probably used most of the time [Casey 04]:

- Temporal – To find the time and sequence of events
- Relational – To find the relationships between suspects, victims and the crime scene.
- Functional – To determine how a computer system functioned. This is done in order to
    - Determine the proper working of the system during the relevant time period.
    - Determine if the individual or a computer was capable of performing actions necessary to commit the crime.
    - Gain a better understanding of a piece of digital evidence or the crime as a whole
    - Gain insight into an offender's intent and motives. E.g. to determine if the act was purposeful or accidental.
    - Prove that digital evidence was tampered with, if this was the case.

The investigative process proceeds from preservation of the state of the technology, to collection of data or acquisition, to examination of data, to the

analysis and extraction of evidence, and to the preservation and presentation of the evidence. [Reith et al 02]. Both the acquisition step [NIST 01] and the preservation step [Hosmer 02] is covered elsewhere, and is of no relevance to this thesis.

To find the truth, an investigator must identify:

- Inculpatory evidence: Evidence that verifies existing data or theories.
- Exculpatory evidence: Evidence that contradicts existing data or theories.

To be able to find both evidence types, all data must be identified and analysed. This is a huge task because of the increasing size of storage systems.

1.3 Slack Space and Other Residual Data

Residual data are data present in a system without being attached to a file. The most common examples are swap files, unallocated clusters and file slack space.

Unallocated clusters are usually found within a file partition, but it can also be found between partitions, since partitions usually start on a whole cylinder [Casey 04].

To be able to explain the concept of slack space, we will have to take a step back: Physical hard disk technology has evolved into a complex subject. Some older encodings use 557 physical bytes to encode a sector of 512 logical bytes [Casey 04]. Newer technologies use advanced mathematics to encode logical bits into the magnetic field of the platter. What is important for us is that most disks in use today present the data to the rest of the computer hardware and operating system in groups of 512 bytes, also called sectors [Carrier 02].

For most file systems[2] the file content is stored in groups of sectors called clusters or blocks. Space for file data is allocated in whole clusters. The unused part of the last cluster is called "slack space".

If the slack space contains data, it can have many sources: It could be a part of the data file in the rest of the cluster [Bryson et al 02]; it could be a part of a file stored in the cluster at one point in time, but later deleted; or, in older versions of Windows, it could even be contents of RAM from when the file was created [Casey 04].

Each cluster consists of one or a power of two consecutive 512-byte sectors [Microsoft 04a]. Some file systems can break the cluster into fragments, but these are never smaller than a sector [Carrier 02].

When it comes to Windows systems, all volumes of sizes greater than 513MB, regardless of file system, have a default cluster size of at least two sectors [Microsoft 04b]. There are, however, some ways to initialise a file system that makes the cluster size one sector, regardless of disk size [Microsoft 01].

As larger disk drives are used, so does the cluster size. This leads to larger areas of slack space. Thus even if the investigator must sift trough more data, the value of each fragment will be greater.

1.4 The Two Fundamental Problems of Digital Forensics

The two fundamental problems of Digital Forensics were defined [Carrier 03] as the following:

[2] There are many, but the top 5 is considered to be FAT, VFAT, NTFS, EXT2 and UFS [Bryson et al 02]

The Complexity Problem in digital forensics is that acquired data are typically at the lowest and most raw format, which is often too difficult for humans to understand.

The Quantity Problem in Digital Forensics is that the amount of data to analyze can be very large. It is inefficient to analyze every single piece of it.

We will not discuss those here, but remembering them may ease the understanding of some of the issues mentioned later.

1.5 Uncertainty and Degradation of Digital Evidence

Integrity and continuity of computer-based evidence is volatile and will degrade at an exponential rate [Janes 00]. This degradation begins the moment the relevant computer is used after an incident.

Very little of the potential evidence is tangible in the normal sense. Digital evidence is interpretive and must be transformed or abstracted from the collection of ones and zeroes it is in it lowest form before it is suitable for analysis [Palmer 02] (the complexity problem). This process can be error prone, both because of inaccurate tools and intent [Anonymous 02].

Uncertainties in evidence can thus not only stem from data corruption, loss or tampering, but also from what simultaneously are introduced by abstraction layer errors. In order to make sense to a human being, digital evidence has to be abstracted. For example; data on a disk is abstracted to a partition, which further is abstracted to a file system, where the content of one file can be abstracted to text in a word processing document. During this process, errors can be introduced because of abstraction errors or implementation errors [Carrier 03].

Generally speaking, the Abstraction Layer Error Problem then becomes the errors that are introduced by the layers of abstraction.

Because of this, we must try to estimate the uncertainty, or how closely the measured values approximate reality.

Even if this is successful, completely reliable sources of digital evidence are not believed to exist [Casey 02]. This makes digital evidence circumstantial, and difficult to use as the sole evidence.

In addition, we have seen a growing problem of "digital gloves" – programs like Evidence Eliminator™ designed to destroy evidence, and runefs designed to hide evidence [Anonymous 02]. Such programs make fewer pieces of evidence available, and force the investigation to be a more arson type[3].

If more information about digital forensics is needed, a good starting point would be the two books by Eoghan Casey listed in appendix 8.3.

1.6 Problem Statement

The problem an investigator faces is thus two-fold:

1. It is too time consuming to investigate on computers, especially old or partially destroyed evidence.

2. Potential evidence is lost or not recognised because it is partially overwritten by new data.

This leads us to the following research questions:

1. How can the investigation process be speeded up?

2. How can evidence be recognised even if it is partially overwritten?

[3] See chapter 1.2

In order to answer the research questions, the following research problems must be solved:

1. How can the quantity problem be solved?

2. Can it be solved easier by using current methods in different ways?

3. Are there ways of easing the complexity problem without the need for abstraction layers?

4. How can one be more certain that most or all evidence is found?

5. In case of lack of complete data, can the fragments be identified in any way?

6. How small can a fragment be in order to be able to identify which file it is a part of?

1.7 Demarcation to Static Environments

A recent survey indicates that technology and tools are considered important issues [Rogers et al 04]. We will therefore focus on technology, and the discussion in this thesis only relate to a statical technical environment, where the actions of the investigator do not have any potential to introduce changes to the data. A typical example of such an environment is a bit stream image of a hard disk drive.

1.8 Review of the State of the Art

As was mentioned earlier, the complexity problem is currently solved by tools that translate data through layers of abstraction until it can be understood by humans [Carrier 03].

The quantity problem can be eased by data reduction; by removing known data or by grouping data together. In our case it is facilitated by sorting files

by types, attributes, or access time or identifying duplicates [NIST 03] or known files by utilizing hash databases.

The latter is possible because a large amount of data consists of operating system and application files not relevant to the investigation. This makes it possible to use hash sets of known good files to eliminate these from review, and thus reduce the total number of files to be reviewed [Larson 02]. The same method can be used to flag known bad files for review [NIST 03]. One common source of hashes for known files is NIST's Reference Data Set[4]

Using this method, the smallest fragment that can be identified is a complete file. This works very well on whole and undamaged evidence files, but is of no use when we are dealing with fragmented, incomplete evidence.

Other reduction methods include focusing on data most probably created by the user, filtering duplicate files and identifying discrepancies [Casey 04].

The effectiveness of current tools is tied to the volume of data involved and the time available to examine it. Storage has grown so large that it is very difficult to examine every sector manually for data that may or may not be evidence [Stephenson 03]. The investigator is therefore often limited to searching for occurrences of text strings. Such a method assumes that it is known which phrases that might yield interesting information. This method seems outdated since the investigator is only able to see the tip of the ice berg, and much incriminating and exculpatory evidence is potentially overlooked [Anderson 01].

Investigators have also begun to question the effectiveness of making bit stream images of whole such systems [Culley 03], despite it being the most thorough option. Some even use statistical data sampling [Anderson 01] in order to identify where to start the search for relevant evidence.

Some even go as far as to claim that full-blown forensic examinations usually are unnecessary, and the investigation should cease when necessary evidences to prosecute have been found [Ferraro 04]. This means skipping searching for exculpatory evidence, and is what happened in the case with the U.K. man described at the beginning.

It has been pointed out that methods must address this issue without sacrificing the quality of the investigation [Janes 00]. Unfortunately, there seems to be a trend away from this.

One method of handling degradation of evidence is to utilize data carving [Casey 04], which means to look through residual data for sections matching known file headers and footers. This may be useful, but its main limitation is that it depends upon having intact headers. A variation of this method is to utilize low-level, short-range structures to classify files [de Vel 04]. This method seems to be able to identify the type of uncompressed file based on a 256 byte (half sector) fragment of the file. It is not able to say anything about the content of the file.

From the review, it seems clear that many of the research problems have been solved, but many still regard handling of fragmented evidence as a manual, labour intensive task.

1.9 Claimed Contributions

In order to preserve the highest investigative standards, we believe we can present a method to solve a part of the quantity problem relating to evidence fragments, and at the same time be able to avoid the complexity and abstraction layer error problems.

[4] NIST Special Database 28 from NIST's Standard Reference Data Group

This is done by applying the well known method of known file hash databases to fragments of files, adapting it so it can be used in the special case of slack space on a disk drive.

It will not give complete solutions to the research problems outlined earlier, but will contribute to some extent to all of them.

Obviously, the stakeholders are not only digital forensic professionals employed by corporations, police bureaus and intelligence agencies, but also people under wrongful suspicions.

1.10 Agenda

This report outlines the results from an implementation of a fragment hash database of real world data.

It starts with a brief account of the research method used and continues with describing the hashing algorithm chosen.

The next section describes how the data is gathered for the test database and how the different fragment databases are made. It follows up with some comments on how the fragment databases are evaluated.

A brief section describes the most important results before the validity and reliability of the experiment and the method is discussed and evaluated against proposed field-specific guidelines.

2 Materials and Methods

2.1 Choice of Methods

Because the research questions are mainly problem centric, a mixed research approach was chosen in order to solve them [Creswell 03]. This means combining both quantitative and qualitative methods involving a literature study and a quantitative experiment. Because of the enormous amount of data produced by the experiment, the quantitative part will be limited to simple statistics.

The hypothesis is that fragments are usable as identification of files. "Usable" is defined here as more than half of the fragments can identify a file uniquely.

In this field, methods for estimating sample sizes and confidence intervals are not very well researched into. This thesis will not try to find an answer to this, but chapter 2.3 explains how the experiment will be designed to try to compensate somewhat for this lack. To further ensure validity, the method and results will be evaluated against guidelines proposed by the community.

2.2 Use of Hashing Algorithms

Hashing is an extremely good way to verify the integrity of a sequence of data bits. It is a mathematical function which generates a hash value produced from the data bits. Two files with the same bit patterns hash to the same value when using the same algorithm. If the hashes for two files match, then it is a very high probability that the files are the same. Because of this, hashes are used as a primary verification tool to find identical files.

When NIST researched algorithms to use in their National Software Reference Library (NSRL), one of the alternatives evaluated was the Message Digest level 5 (MD5) algorithm [Boland et al 00]. MD5 is an improved version of MD4, and is intended for digital signature applications. It processes data in 512 bit (64 bytes) blocks, and outputs a 128-bits (16 byte) hash value.

This algorithm has since become the de-facto standard for identifying known files in the digital forensic field, and is therefore the one used in this experiment.

It would undoubtedly give more accurate results if we could populate the data with the exact contents of the files instead of the hashes. But this would raise ethical issues about handling of intellectual property for known good data, and possession and distribution for known bad data. In addition the storage requirement for the sample database would multiply as much as 32 times.

2.3 Generation of the Data Set

By using files containing purely random data, a fragment might be enough to uniquely identify a file, and the chance of a correct identification of would be possible to calculate. Real-world data is not random, so files from real systems will be used.

Although forensic investigations often are focused on illegal material and files, ethical and legal considerations made it necessary to only use data from regular operating systems and program.

The experiment will try to compensate somewhat for the lacking estimation of sample sizes and confidence intervals by using huge amount of data files.

Our test data was Microsoft Windows Trial Software and Service Pack for 3 architectures and 26 languages 228617, the complete Windows Platform SDK, which is the building stone for many software packages 8760, 4 complete Linux distributions of different revisions and 5 x86-architectures 623796, FreeBSD for 2 different non x86-archtectures 140406 and Sun Sorlaris Patches for 2 operating system versions and architectures 24277. Chapter 8.7 contains a detailed description of the test data

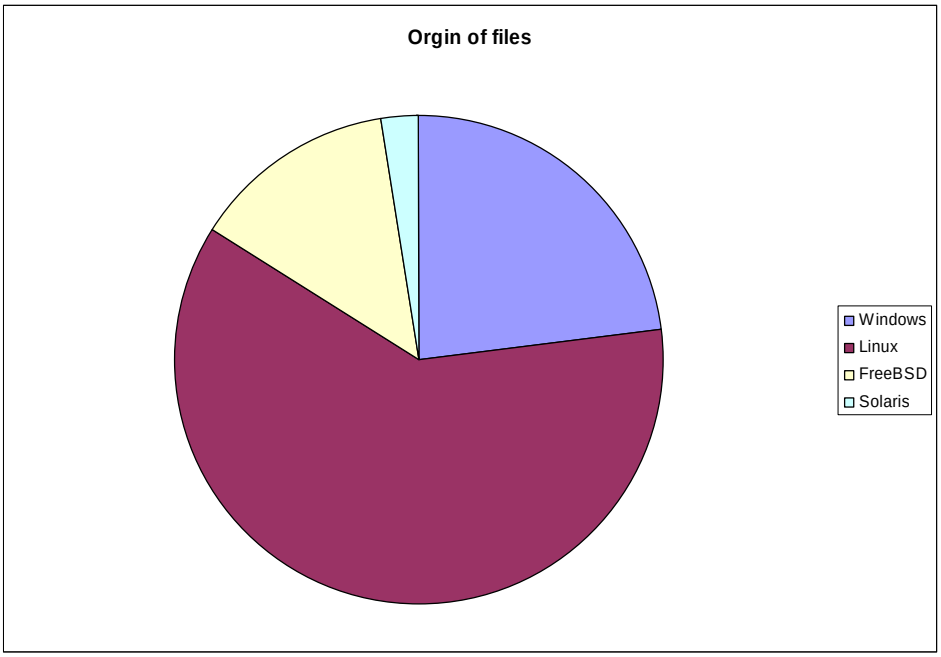

Figure 1 - The distribution of origin of the files in the test data base.

The data contain an overweight of similar data; the same program file compiled for different languages and architectures, and the same program or data file in different revisions. This simulated worst case scenario was done to limit the possibility for type I errors, and to get an impression of how the results would have been on a fully populated database.

All files were extracted to their fullest extent[5]. This means that all nested archives and compressed files were iteratively extracted until no more files were generated. In the process, no files were overwritten, and the original archive or compressed file was kept intact together with all intermediate files.

All files with sizes less than 2048 bytes and more than 16776704 bytes were then discarded.

[5] Some large archives were split into multiple parts in an uncommon way. With such archives, only the first part was extracted

File Extension	Instances
gz	70321
html	50376
dll	46666
dl_	40893
mo	36902
h	20029
png	15517
so	13600
bz2	13475
exe	12005
htm	10630
ex_	10183
o	9169
c	9127
pcf	8795
class	8200
docbook	7910
1	6332
cache	5811
xml	5627
inf	5571
3	5518
ko	5057
elc	4948
rpm	4513
a	4314
cpio	3888
java	3719
in_	3717
desktop	3440

Table 1 – The 30 most common file extensions of the files in the test database

The only real limitation of size would have been files of 512 bytes or more. This is because at least one full block is needed to produce a hash value. A minimum of 2048 was chosen to achieve at least four full continuous blocks, and this would in a future experiment indicate how many blocks would be needed to make a definitive identification. An upper boundary to limit the number of large archives in the databases was needed, and the number was chosen because it represented the maximum number of blocks expressible by an unsigned integer. Compressed archives are also files, but since they per

definition contain little redundant data and most likely would provide unique hash values, they were not very valuable paramount to include in the data.

A MD5 hash was generated of every file in order to identify duplicate files. The strength of the MD5 algorithm has been questioned, but since it is the de-facto standard in the field, and has been the subject of legal scrutiny, it was a natural choice for this project.

All duplicates were then discarded, but the location was noted for future reference. The 1 025 856 files then produced the data set of 625 529 unique, but similar files amounting a total of 61 798 345 340 bytes.

In addition of the overview found in appendix 8.2, appendix 8.7 contain a detailed inventory of the files used in the database.

2.4 Generation of the Hash Database

For each 512 byte block in each file, the MD5 hash was generated of the last 1, 2, 3, 4, 6, 8, 12, 16, 24, 32, 48, 64, 96, 128, 192, 256 and 386 bytes. The sample sizes were chosen because they divide a full block in half iteratively, and then divide each half once more. The hash was also generated for the block as a whole to able to judge how well the smaller blocks served as an identification of the block as a whole. If the files consisted of truly random data, 27 bits are needed to identify an exact block inside a file, so there was no need to calculate the database for 1, 2 and 3 byte blocks.

The hashes were sorted and placed in hash databases, one for each of the block sizes. Each database contained 120 411 278 hashes with references to the original file and the block position in it. This was more than six times the size proposed in the original project plan, a modification necessary to make the results more suitable for generalization.

2.5 Experiments on the Data Set

In order to find a fragment size that produced a balance between frequency of collisions (false positives) and feasibility for usage, the 18 populated databases were examined with simple counting and cross referencing. The number of unique identifications and the number of hits of non unique was counted:

- Relatively to a unique file[6] and the exact position in it.
- Relatively to a unique file.
- Relatively to a unique file name
- Relatively to a unique distribution/ database reference

When counting, the hash value for a stream of zeroes was removed from the data set because it is usually the default value on unused parts of the disk, and it is also very common - 10,8 to 13,6 times more common than the next extreme value.

The exact method used in counting can be found by examining the listings in appendix 8.5.

[6] With an unique file, we mean a file with a unique MD5 hash value

3 Results

After counting the number of duplicates in the full block reference database, it turned out that 41 803 042, one third, of the hashes were identical to at least one another. This means that two thirds of the fragments could be uniquely identified as belonging to a specific file. The distribution of the collisions can be seen in the table below.

Files identified	Number of hashes	Unique hashes
Sum	120411278	88623418
1 (unique)	78838754	78697936
2 (usable)	12744746	6343505
3 (usable)	4321913	1426325
4 (usable)	2237552	552753
5 (usable)	2323359	459465
6 (usable)	1406662	228129
7 (usable)	1199157	166931
8 (usable)	1119482	137367
9 (usable)	670272	72708
10-	13349252	537180
100-	380757	1099
1000-	149230	19
10000-	1670142	1

Table 2 – Identification potential of in the test data for full 512 byte (one sector) blocks

Hashes with 2 to 9 collision were tagged as usable (or guessable). This means that one fragment might suggest 2 to 9 files, but if there are more fragments, one exact might be interpolated.

If we ignore the unique hashes and look only at the duplicates, the following table illustrates the identification potential relating to other attributes of the file.

	Positions in file	Distributions	File names
1 Unique	8299195	3530535	4676377
2 Duplicate	923851	3309392	4702783
3- Multiple (usable)	760050	2588805	605958
10-	30587	586531	29412
100-	1557		716

1000-	22		17
10000-	1		
Sum non-uniques	1716068	6484728	5338886

Table 3 – Other distribution of the duplicate hashes from the full 512 byte (one sector) blocks

For example does this mean that even if a fragment hash identifies multiple possible unique files, it may still identify only one file name.

Appendix 8.1 contains the full results for all fragment sizes. The following graph illustrates the identification rates for each size:

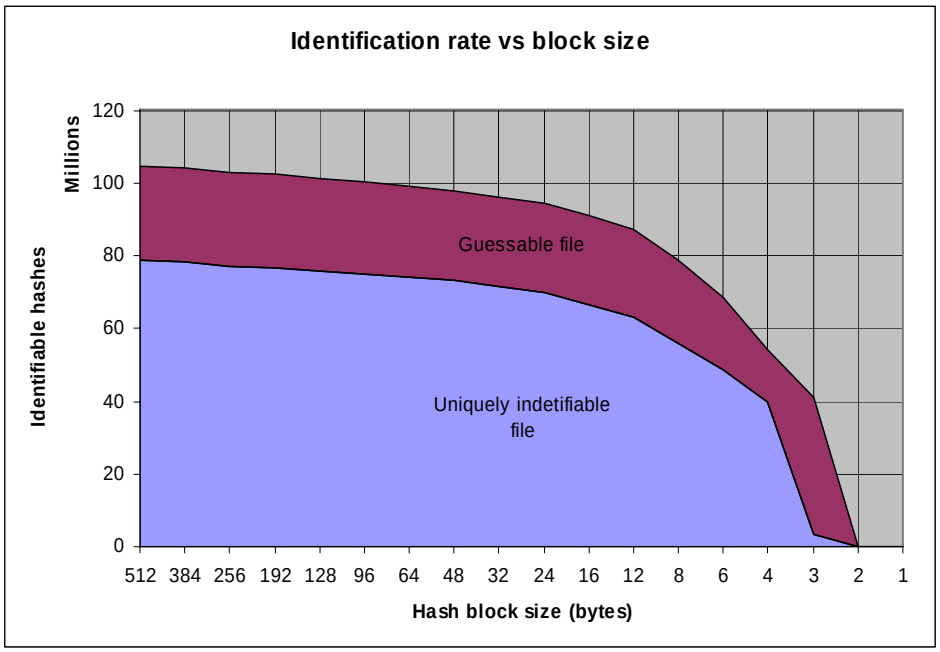

Figure 2 - The identification potential for each of the block sizes in the experiment

The identification potential stays relatively high, but starts to drop sharply for fragments smaller than 12 bytes.

.

4 Discussion

Although the test data for the database were carefully selected, they are still nothing more than convenience samples. Also, errors in the software and programs developed may have affected the validity of the results. This might limit to what extent it can be generalized.

By using raw data from the fragment directly, many facets of the validity problems are tackled. For example are errors associated with the forensic specific complexity and abstraction layer error problems completely avoided.

4.1 Selecting the Recommended Block Size

Based on the acceptation criteria outlined in chapter 2.1, the minimum block size usable for identification of files is somewhere between 8 and 12 bytes.

Since the MD5 hash value itself stored in the database is 16 bytes and it is calculated based on 32 byte data blocks, it would probably make no sense in using smaller fragments than 32 bytes without evaluating different hash functions.

Since the 32 byte block also only gives a mere 9% drop in the unique identification rates compared to the full block, it makes sense recommending it as the preferred size to use in a practical implementation of the fragment hash database.

A 32 byte fragment should be safely past margins of error since it can identify a file in more than 60% of the time despite of the large size and similarity of the test data.

4.2 Accounting for Duplicate Hashes

As shown in chapter 3, 35% of the full sector (512 bytes) blocks did not uniquely identify a file. Below are the findings that explain some of the reason behind this.

The tar file format was originally designed for tape devices capable of manipulating a block at a time. The default block size is 512 bytes [Schmidt 03], which is the same as the size used in hard disks and this experiment. In addition each non-full block is padded with zeroes to make a full 512 byte block.

Since each intermediate file during expansion was kept, any block from a file that was originating from one of the tar files in some form (tar, tgz, tbz2 and tar.Z) will most likely be found at least twice, and will not contribute to the exact identification figure.

This applies to the vast majority of files from Gentoo Linux (.tbz2) and FreeBSD (.tgz) distributions, and must therefore be a significant factor in determining the numbers and results.

Another factor is when a Windows executable file is localized, usually only the end of it is changed. Windows files existing in different languages will therefore count towards many duplicates in the hash database, without being a true duplicate.

A block of only zeros produced the most common duplicate hash. It accounted for 1 670 142 of the full block duplicates, but rises sharply as the sample size decreases. This is more than ten times more instances than the next extreme hash value. This could also somewhat be attributed to pad with zero of the tar file format [Schmidt 03].

On the other hand, the most common file type in the test data was the compression format gzip. Since the aim of most compression algorithm is to reduce the redundancy in files to make them smaller, this lead to slightly better average identification values than the method deserves.

It is reasonable to believe that as the hash database grows, the duplicate ratio will grow with it, but it is not possible to predict at what rate without further experiments.

4.3 Evaluation Against Proposed Guidelines

The majority of efforts in Digital Forensics has focused on development and use of tools [Whitcomb 02], while ignoring the theoretical foundations thereof. This has lead to attacks against the reliability and validity of this method.[Carrier et al 03].

Brian Carrier and Sarah Mocas have both tried to address the theoretical foundation for their development and have provided a set of guidelines to verify the soundness of a method or tool [Mocas 04] [Carrier 03]. They overlap somewhat, but can be summarized in the following eight criteria:

1. **Non-interference** - That evidence is not altered, or is altered in an identified manner.

2. **Usability** – That the method or tool provides output at a level of abstraction that is understandable by the investigator, and it is so clear and accurate that it can not be interpreted incorrectly.

3. **Comprehensive** – That access to all output data is provided. This is so an investigator is able to find both inculpatory and exculpatory evidence.

4. **Accuracy** – That the method used to gain usability must be an accurate translation trough the layers of abstraction, or if not, that the margin of error is known. This is to ensure that the evidence presented is what it claims. This can be thought of as a reliability issue in other scientific contexts.

5. **Deterministic** – That the process is reproducible, and always produces the same output, given the same data and transformation rules. Also a reliability issue.

6. **Verifiable** – That the result is possible to verify using alternative methods

7. **Minimization** – That the minimum necessary data was examined to provide the output.

When assessing the proposed method using these guidelines, one gets the following results:

Requirement	Satisfied	Comment
Non-interference	Yes	We are operating in a statically technical environment, and the original data is only used to perform lookups in databases
Usability	Yes	The investigator is presented with a list of all files that contains the matching byte pattern
Comprehensive	Yes	as above
Accuracy	Yes	Since the method operates on the lowest level of abstraction, no translation is necessary
Deterministic	Yes	Since both the hash algorithms and database lookup is deterministic, so is the method. If, however, the database is updated with new known data, the output contains more results.
Verifiable	Yes	An investigator is able to manually verify a byte

		pattern to an original file.
Minimization	Yes	Only the number of bytes of each sector needed to produce a hash value is used.

Table 4 – Summary of requirement satisfaction for the proposed method.

This indicates that the proposed method is sound and does not contradict the current best practices in the field.

4.4 Levels of Uncertainty

The validity of evidence is important, and in order to find a balance between the need for estimates of certainty in digital evidence versus the cost and complexity of calculating uncertainty, one of the pioneers in digital forensics, Eoghan Casey, proposed a scale for categorizing levels of certainty in digital evidence [Casey 02]. This scale ranges form C0 - Erroneous/ Incorrect to C6 – Certain, which is considered inconceivable at the moment.

Trying to describe the level of certainty of the proposed method using this scale, indicates level C2 - Somewhat Uncertain. This is because there is only one source of evidence and that source is not protected against tampering.

But if the method is used on several blocks and gives the same result, the certainty level rises to C4 - Probable. This is because there are independent sources of evidence that agree. However, the evidence is still not protected against tampering.

4.5 Resistance to Tampering

It is important to note that the method only identifies the block from which the hash is taken. There are no guarantees that the rest of the slack space is from the same dataset. Because of this, one must be careful of jumping to quick conclusion without comparing the finding to other evidence.

One can also imagine that if a method like the one described in this thesis becomes widespread, the electronic glove programs will be modified to place fragments of trojans into slack space in addition to the techniques in use today.

This suggests that the method should not be used as sole conclusive evidence or in a system to automatically filter data.

4.6 Feasibility for a Real World Implementation

The known files database delivered with AccessData's Forensic Toolkit is approximately 70 MB in size. The average size of the files in the database in this experiment was just below 99 000 bytes (193 fragments). Based on this, one could assume that the size of a reasonable complete fragment database would be around 28GB

If using similar calculations on the 7,198,856 unique files in version 2.3 of the NIST NSRL hash database, one could estimate that the fragment database would be around 44GB.

None of these sizes is so large that it should prevent practical implementations.

5 Conclusion and Further Work

Many solutions to the research problems were pointed out in the introduction, and the experiment also suggested in depth solutions to problems 2, 3, 5 and 6.

The author of this research was not able to provide a good answer to how one could be sure that most or all evidence is recognised, but the research indicates that using the proposed method will bring us one step closer to the final solution.

It also showed that this method seems superior to classification by low-level, short-range structures [de Vel 04] or data carving [Casey 04] when looking at fragments of known files.

However, since it might be inaccurate and susceptible to falsification of evidence, it might best be used to complement other methods.

Maybe the best compromise between resource usage and legal protection is to start out with searching for just enough evidence to prosecute as suggested by [Ferraro 04], but switch to a full-blown forensic investigation if any of the quick hash based methods indicates presence of malicious code.

5.1 Further Work

When examining the results of the experiment, some natural follow up questions materialized.

It might be useful to analyze the data to find out how the identification ratio changed based on the origin and type of files in the database. This would give answer to if the method is still usable if, for example, all files originate from a 32bit Windows system or if all files are image files.

A prototype robust enough for general access implementing this method could be deployed in order to get data on how helpful it is to the community.

It might be interesting to look at to what extent the more advanced techniques used by anti virus scanners can be used to identify fragments more effectively and accurate than using fragment hashes.

One could also explore the way NTFS compressed files are stored on disk[7] and examine if it is possible to do something similar with hashes for those structures. Today, no tool exists that is able to analyze NTFS compressed slack space, and it is believed that it is impossible to do so without the accompanying Master File Table entry [Sanderson 02].

It might also be possible to explore if and how it can be adapted to network traffic, either in a general manner or in a protocol specific manner.

[7] NTFS compresses files in units of 16 clusters (8 kB minimum), each which can consist of either compressed or uncompressed data. After compression, the remaining space in the unit is padded with zeroes, and it is not used until the disk runs out of space

6 Acknowledgements

The author wishes to thank Telenor ASA for the support, and being granted time to conduct work on this thesis, and his advisor, Einar Snekkernes for invaluable advice during the process.

I would also like to thank my wife, Agnes, and kids, Miko, Jan and Maria who have not complained too much during the many and long evenings and weekends used to complete this thesis.

7 References

[Anonymous 02] Anonymous. (July 2002). Defeating Forensic Analysis on Unix. Phrack, Volume 0x0b, Issue 0x3b, Phile #0x06 of 0x12. Retrieved on May 30. 2004 from http://phrack.org/gogetit/phrack59/p59-0x06.txt

[Anderson 01] Anderson, Michael R. (May 2001). Hard Disk Drives - Bigger is Not Better: Increasing Storage Capacities, The Computer Forensics Dilemma. New Technologies Armor, Inc. Retrieved May 28. 2004 from http://www.forensics-intl.com/art14.html

[Boland et al 00] Boland, Tim. Fisher, Gary. (June 2000). Selection of Hashing Algorithms. Retrieved May 28. from http://www.nsrl.nist.gov/documents/hash-selection.doc

[Bryson et al 02] Bryson Curt. Stevens, Scott. (2002). Tool testing and Analytical Methodology. Casey, Eoghan. Handbook of Computer Crime Investigation, London: Academic Press.

[Carrier 02] Carrier, Brian. (June 2002). An Investigator's Guide to File System Internals. Presentation at FIRST 2002, 24 Jun 2002, Hawaii, USA.

[Carrier 03] Carrier, Brian. (2003). Defining Digital Forensic Examination and Analysis Tools Using Abstraction Layers. Journal of Digital Evidence, WINTER 2003, Volume 1, Issue 4.

[Carrier et al 03] Carrier, Brian. Spafford, Eugene H. (2003). Getting Physical with the Digital Investigation Process . International Journal of Digital Evidence. Fall 2003, Volume 2, Issue 2

[Casey 02] Casey, Eoghan. (2002). Error, Uncertainty, and Loss in Digital Evidence. Journal of Digital Evidence, Summer 2002, Volume 1, Issue 2.

[Casey 03] Casey, Eoghan. (July 2003). Arson, Archaeology, and Computer Crime Investigation. Computer Fraud & Security, Volume 2003, Issue 7, Pages 12-15.

[Casey 04] Casey, Eoghan. (2004). Digital Evidence and Computer Crime: Forensic Science, Computers and the Internet. (Second Edition). London: Academic Press. Chapters 8-10.

[CERT 03] CERT® Coordination Center (CERT/CC). (January 2003). CERT/CC Statistics 1988-2003. Retrieved June 4. 2004 from http://www.cert.org/stats/

[Creswell 03] Creswell , John W. (2003). Research Design: Qualitative, Quantitative and Mixed Methods Approaches. Second Edition, SAGE Publications

[Culley 03] Culley, Adrian. (June 2003). Computer forensics: past, present and future. Information Security Technical Report, Volume 8, Issue 2, Pages 32-36.

[Ferraro 04] Ferraro, Monique Mattei. Russell, Andrew. (February 2004). Current issues confronting well-established computer-assisted child exploitation and computer crime task forces. Digital Investigation, Volume 1, Issue 1, Pages 7-15.

[Gary 01] Palmer, Gary. (November 2001). A Road Map for Digital Forensic Research. Technical Report DTR-T0010-01, DFRWS. Report from the First Digital Forensic Research Workshop (DFRWS).

[Hosmer 02] Hosmer, Chet. (Spring 2002). Proving the Integrity of Digital Evidence with Time. International Journal of Digital Evidence, Volume 1, Issue 1.

[Janes 00] Janes, Simon. (June 2000). The Role of Technology in Computer Forensic Investigations. Information Security Technical Report, Volume 5, Issue 2, Pages 43-50.

[Kruse et al 01] Kruse, Warren G. Heiser, Jay G. (September 2001). Computer Forensics: Incident Response Essentials. Addison Wesley Professional

[Larson 02] Larson, Troy. (2002). The other Side of Civil Discovery. Casey, Eoghan. Handbook of Computer Crime Investigation, London: Academic Press. Page 43.

[Microsoft 01] Microsoft. (August 2001). Microsoft Knowledge Base Article - 231756: The Convert.exe Tool Uses 512-Byte Clusters. Retrieved May 30. 2004 from http://support.microsoft.com/default.aspx?scid=kb;en-us;231756

[Microsoft 04a] Microsoft. (April 2004). MSDN: Supported file systems. Retrieved May 30. 2004 from http://msdn.microsoft.com/library/en-us/fileio/base/supported_file_systems.asp

[Microsoft 04b] Microsoft. (2004). Cluster Size. Microsoft Windows XP Resource Kit, Part II Desktop Management, Chapter 13. Retrieved May 30. 2004 from http://www.microsoft.com/resources/documentation/Windows/XP/all/reskit/en-us/prkc_fil_lxty.asp

[Mocas 04] Mocas, Sarah. (February 2004). Building theoretical underpinnings for digital forensics research. Digital Investigation, Volume 1, Issue 1, Pages 61-68.

[NIST 01] NIST CFTT. (October 2001). Disk Imaging Tool Specification, v3.16.

[NIST 03] NIST. (August 2003). National Software Reference Library (NSRL): Project Overview. Retrieved May 28. 2004 from http://www.nsrl.nist.gov/Project_Overview.htm

[Olsen et al 04] Olsen, Inger Anne. Rapp, Ole Magnus. (May 2004). Søker blant 850.000 barnepornobilder. Aftenposten Nettutgaven. May 27. 2004 00:12. Retrieved June 5. 2004 from http://www.aftenposten.no/nyheter/iriks/article797019.ece

[Palmer 02] Palmer, Gary L. (2002). Forensic Analysis in the Digital World. Journal of Digital Evidence, Spring 2002, Volume 1, Issue 1.

[Politidirektoratet 03] Politidirektoratet. (April 2003). Nasjonal trusselvurdering 2003.

[Reith et al 02] Reith, M., Carr, C., Gunsch., G. (2002). An examination of digital forensic models. International Journal of Digital Evidence, Volume 1, Issue 3.

[Rogers et al 04] Rogers, Marcus K. Seigfried, Kate (February 2004). The future of computer forensics: A needs analysis survey. Computers & Security, Volume 23, Issue 1, Pages 12-16.

[Sanderson 02] Sanderson, Paul. (October 2002). NTFS Compression - a forensic view. Revised January 3. 2003. Retrieved May 28. 2004 from http://www.sandersonforensics.co.uk/Files/NTFS%20compression%20white%20paper.pdf

[Schwartz 03] Schwartz, John. (2003). Acquitted Man Says Virus Put Pornography On Computer. New York Times, August 11, 2003, Late Edition - Final , Section C , Page 1 , Column 5.

[Schmidt 03] Schmidt, Marco. (June 2003). TAR archive file format. Retrieved June 4. 20004 from http://www.geocities.com/marcoschmidt.geo/tar-archive-file-format.html

[Stephenson 02] Stephenson, Peter. (December 2002). Analysis and Correlation. Computer Fraud & Security, Volume 2002, Issue 12, Pages 16-18.

[Stephenson 03] Stephenson, Peter. (June 2003). A comprehensive approach to digital incident investigation. Information Security Technical Report, Volume 8, Issue 2, Pages 42-54.

[de Vel 04] de Vel, Olivier. (June 2004). File classification using byte sub-stream kernels. Digital Investigation, Volume 1, Issue 2, Pages 150-157.

8 Appendices

Local Gjøvik University College regulations state that all information that could be useful to the next generation of thesis writers or necessary to duplicate the results should be included. This section therefore contains some detail information that normally would have been considered irrelevant.

8.1 Detailed result tables

Identification when using 512 byte hash blocks (full sector)

Dup-licates	Identify only file				Identify file and position in it			
	Total hashes	%	Unique hashes	%	Total hashes	%	Unique hashes	%
Unique	78838754	65	78697936	88	78608124	65	78608124	88
2	12744746	10	6343505	7	12785760	10	6392880	7
3	4321913	3	1426325	1	4308729	3	1436243	1
4	2237552	1	552753	0	2257192	1	564298	0
5	2323359	1	459465	0	2291330	1	458266	0
6	1406662	1	228129	0	1395822	1	232637	0
7	1199157	0	166931	0	1161286	0	165898	0
8	1119482	0	137367	0	1117816	0	139727	0
9	670272	0	72708	0	671040	0	74560	0
10-	13349252	11	537180	0	13101138	10	547809	0
100-	380757	0	1099	0	633642	0	2852	0
1000-	149230	0	19	0	228155	0	117	0
10000-	1670142	1	1	0	1851244	1	7	0
Sum	120411278	100	88623418	100	120411278	100	88623418	100

Table 5 – Identification when using 512 byte hash blocks (full sector)

Identification when using 384 byte hash blocks

Dup-licates	Identify only file				Identify file and position in it			
	Total hashes	%	Unique hashes	%	Total hashes	%	Unique hashes	%
Unique	78230679	64	78068152	88	77962933	64	77962933	88
2	12682208	10	6308709	7	12734390	10	6367195	7
3	4323112	3	1424930	1	4311039	3	1437013	1
4	2244545	1	552171	0	2261236	1	565309	0
5	2319504	1	458386	0	2286920	1	457384	0
6	1387000	1	225796	0	1385226	1	230871	0
7	1195281	0	166059	0	1156050	0	165150	0
8	1177849	0	142127	0	1157928	0	144741	0
9	670214	0	72570	0	671211	0	74579	0
10-	13651514	11	546298	0	13374601	11	558010	0
100-	398953	0	1206	0	683634	0	3116	0
1000-	192591	0	32	0	259137	0	127	0
10000-	1937828	1	1	0	2166973	1	9	0

Identification when using 384 byte hash blocks

Dup-licates	Identify only file				Identify file and position in it			
	Total hashes	%	Unique hashes	%	Total hashes	%	Unique hashes	%
Sum	120411278	100	87966437	100	120411278	100	87966437	100

Table 6 - Identification when using 384 byte hash blocks

Identification when using 256 byte hash blocks (half sector)

Dup-licates	Identify only file				Identify file and position in it			
	Total hashes	%	Unique hashes	%	Total hashes	%	Unique hashes	%
Unique	77261593	64	77088202	88	76978324	63	76978324	88
2	12565884	10	6246027	7	12609254	10	6304627	7
3	4303405	3	1416394	1	4284894	3	1428298	1
4	2205853	1	542747	0	2232192	1	558048	0
5	2311005	1	456055	0	2276180	1	455236	0
6	1356172	1	220975	0	1360236	1	226706	0
7	1188566	0	165083	0	1150478	0	164354	0
8	1182844	0	142648	0	1163568	0	145446	0
9	668518	0	72166	0	666711	0	74079	0
10-	14162066	11	562013	0	13831233	11	574993	0
100-	545316	0	1337	0	749317	0	3420	0
1000-	446409	0	39	0	292004	0	142	0
10000-	2213647	1	1	0	2816887	2	14	0
Sum	120411278	100	86913687	100	120411278	100	86913687	100

Table 7 - Identification when using 256 byte hash blocks (half sector)

Identification when using 192 byte hash blocks

Dup-licates	Identify only file				Identify file and position in it			
	Total hashes	%	Unique hashes	%	Total hashes	%	Unique hashes	%
Unique	76758949	63	76572085	88	76453656	63	76453656	88
2	12501370	10	6209957	7	12544630	10	6272315	7
3	4287654	3	1409496	1	4266795	3	1422265	1
4	2198001	1	540374	0	2229984	1	557496	0
5	2303985	1	454136	0	2268260	1	453652	0
6	1332910	1	217369	0	1340946	1	223491	0
7	1183005	0	163934	0	1143002	0	163286	0
8	1167136	0	143072	0	1168336	0	146042	0
9	666505	0	71769	0	663516	0	73724	0
10-	14582860	12	574176	0	14192227	11	588067	0
100-	482051	0	1459	0	807109	0	3708	0
1000-	253270	0	39	0	336797	0	153	0
10000-	2693582	2	1	0	2996020	2	12	0
Sum	120411278	100	86357867	100	120411278	100	86357867	100

Table 8 - Identification when using 192 byte hash blocks

Identification when using 128 byte hash blocks (quarter of sector)

Dup-licates	Identify only file				Identify file and position in it			
	Total hashes	%	Unique hashes	%	Total hashes	%	Unique hashes	%
Unique	75919817	63	75710767	88	75578292	62	75578292	88
2	12393734	10	6148063	7	12430978	10	6215489	7
3	4261949	3	1399633	1	4241091	3	1413697	1
4	2194234	1	538652	0	2234728	1	558682	0
5	2284060	1	449813	0	2251205	1	450241	0
6	1307850	1	212383	0	1317240	1	219540	0
7	1169292	0	161671	0	1128918	0	161274	0
8	1177707	0	144021	0	1179912	0	147489	0
9	663676	0	71279	0	660159	0	73351	0
10-	15248633	12	594888	0	14797177	12	610452	0
100-	621035	0	1828	0	974723	0	4354	0
1000-	294467	0	40	0	368964	0	163	0
10000-	2874824	2	1	0	3247891	2	15	0
Sum	120411278	100	85433039	100	120411278	100	85433039	100

Table 9 - Identification when using 128 byte hash blocks (quarter of sector)

Identification when using 96 byte hash blocks

Dup-licates	Identify only file				Identify file and position in it			
	Total hashes	%	Unique hashes	%	Total hashes	%	Unique hashes	%
Unique	75238990	62	75011088	88	74867954	62	74867954	88
2	12297507	10	6093770	7	12329036	10	6164518	7
3	4245277	3	1391450	1	4218288	3	1406096	1
4	2190347	1	536784	0	2236636	1	559159	0
5	2269957	1	446298	0	2237435	1	447487	0
6	1288473	1	209104	0	1302096	1	217016	0
7	1160040	0	160134	0	1120434	0	160062	0
8	1182990	0	144315	0	1187176	0	148397	0
9	663256	0	71047	0	658116	0	73124	0
10-	15748353	13	611313	0	15290626	12	628573	0
100-	791439	0	2207	0	1117093	0	4954	0
1000-	88347	0	39	0	436994	0	196	0
10000-	3246302	2	2	0	3409394	2	15	0
Sum	120411278	100	84677551	100	120411278	100	84677551	100

Table 10 - Identification when using 96 byte hash blocks

Identification when using 64 byte hash blocks (double MD5 block size)

Dup-licates	Identify only file				Identify file and position in it			
	Total hashes	%	Unique hashes	%	Total hashes	%	Unique hashes	%
Unique	74120169	61	73852777	88	73689859	61	73689859	88
2	12147145	10	6005366	7	12165990	10	6082995	7
3	4219217	3	1377805	1	4180620	3	1393540	1

Identification when using 64 byte hash blocks (double MD5 block size)

Dup-licates	Identify only file				Identify file and position in it			
	Total hashes	%	Unique hashes	%	Total hashes	%	Unique hashes	%
4	2197277	1	536485	0	2251376	1	562844	0
5	2250874	1	441506	0	2218295	1	443659	0
6	1265846	1	204566	0	1282860	1	213810	0
7	1147655	0	157501	0	1105608	0	157944	0
8	1193089	0	144967	0	1202280	0	150285	0
9	663467	0	70519	0	654462	0	72718	0
10-	16553266	13	636475	0	16067182	13	656755	0
100-	1045477	0	2830	0	1373645	1	6154	0
1000-	184533	0	52	0	576727	0	270	0
10000-	3423263	2	2	0	3642374	3	18	0
Sum	120411278	100	83430851	100	120411278	100	83430851	100

Table 11 - Identification when using 64 byte hash blocks (double MD5 block size)

Identification when using 48 byte hash blocks

Dup-licates	Identify only file				Identify file and position in it			
	Total hashes	%	Unique hashes	%	Total hashes	%	Unique hashes	%
Unique	73177667	60	72874001	88	72693505	60	72693505	88
2	12012957	9	5926559	7	12019574	9	6009787	7
3	4201711	3	1369142	1	4158909	3	1386303	1
4	2205978	1	536044	0	2262220	1	565555	0
5	2235892	1	437565	0	2202930	1	440586	0
6	1257114	1	202138	0	1274700	1	212450	0
7	1138513	0	155606	0	1096074	0	156582	0
8	1201206	0	145531	0	1214144	1	151768	0
9	667147	0	70431	0	655956	0	72884	0
10-	17153161	14	656229	0	16685870	13	679794	0
100-	1336301	1	3414	0	1603271	1	7161	0
1000-	258077	0	59	0	684191	0	324	0
10000-	3565554	2	2	0	3859934	3	22	0
Sum	120411278	100	82376721	100	120411278	100	82376721	100

Table 12 - Identification when using 48 byte hash blocks

Identification when using 32 byte hash blocks (MD5 block size)

Dup-licates	Identify only file				Identify file and position in it			
	Total hashes	%	Unique hashes	%	Total hashes	%	Unique hashes	%
Unique	71483901	59	71119651	88	70908002	58	70908002	88
2	11762878	9	5777179	7	11737588	9	5868794	7
3	4199690	3	1361451	1	4145874	3	1381958	1
4	2231050	1	538092	0	2293064	1	573266	0
5	2224026	1	432939	0	2186635	1	437327	0
6	1255918	1	200583	0	1277814	1	212969	0

Identification when using 32 byte hash blocks (MD5 block size)

Dup-licates	Identify only file				Identify file and position in it			
	Total hashes	%	Unique hashes	%	Total hashes	%	Unique hashes	%
7	1123888	0	152825	0	1082886	0	154698	0
8	1222107	1	147204	0	1241432	1	155179	0
9	681394	0	71260	0	668547	0	74283	0
10-	18190322	15	688437	0	17694187	14	718241	0
100-	1807724	1	4760	0	2061446	1	9303	0
1000-	449320	0	84	0	855606	0	416	0
10000-	3779060	3	2	0	4258197	3	31	0
Sum	120411278	100	80494467	100	120411278	100	80494467	100

Table 13 - Identification when using 32 byte hash blocks (MD5 block size)

Identification when using 24 byte hash blocks

Dup-licates	Identify only file				Identify file and position in it			
	Total hashes	%	Unique hashes	%	Total hashes	%	Unique hashes	%
Unique	69922027	58	69510933	88	69274232	57	69274232	87
2	11534461	9	5644251	7	11484710	9	5742355	7
3	4219338	3	1361208	1	4151706	3	1383902	1
4	2270444	1	543661	0	2330296	1	582574	0
5	2227813	1	431291	0	2185300	1	437060	0
6	1278758	1	202176	0	1297026	1	216171	0
7	1125642	0	151941	0	1080583	0	154369	0
8	1250778	1	149258	0	1268880	1	158610	0
9	701301	0	72800	0	688014	0	76446	0
10-	18973354	15	712383	0	18468747	15	748433	0
100-	2214836	1	5931	0	2451088	2	11253	0
1000-	617944	0	104	0	1040549	0	499	0
10000-	4074582	3	3	0	4690147	3	36	0
Sum	120411278	100	78785940	100	120411278	100	78785940	100

Table 14 - Identification when using 24 byte hash blocks

Identification when using 16 byte (128 bits) hash blocks (MD5 result size)

Dup-licates	Identify only file				Identify file and position in it			
	Total hashes	%	Unique hashes	%	Total hashes	%	Unique hashes	%
Unique	66618438	55	66141842	87	65871539	54	65871539	87
2	10988915	9	5339005	7	10878410	9	5439205	7
3	4293138	3	1374331	1	4202799	3	1400933	1
4	2356008	1	557449	0	2411596	2	602899	0
5	2247864	1	431242	0	2196000	1	439200	0
6	1360530	1	212221	0	1369782	1	228297	0
7	1158948	0	154490	0	1108051	0	158293	0
8	1323600	1	155296	0	1336024	1	167003	0
9	759840	0	77775	0	738504	0	82056	0

Identification when using 16 byte (128 bits) hash blocks (MD5 result size)

Dup-licates	Identify only file				Identify file and position in it			
	Total hashes	%	Unique hashes	%	Total hashes	%	Unique hashes	%
10-	20476117	17	760809	1	19919239	16	807646	1
100-	3238150	2	9043	0	3468545	2	15897	0
1000-	1055940	0	160	0	1470682	1	653	0
10000-	4533790	3	3	0	5440107	4	45	0
Sum	120411278	100	75213666	100	120411278	100	75213666	100

Table 15 - Identification when using 16 byte (128 bits) hash blocks (MD5 result size)

Identification when using 12 byte hash blocks

Dup-licates	Identify only file				Identify file and position in it			
	Total hashes	%	Unique hashes	%	Total hashes	%	Unique hashes	%
Unique	63032525	52	62518912	87	62236887	51	62236887	87
2	10395134	8	5018509	7	10224498	8	5112249	7
3	4335720	3	1380649	1	4222479	3	1407493	1
4	2440657	2	572756	0	2480620	2	620155	0
5	2239550	1	426055	0	2179160	1	435832	0
6	1447695	1	224878	0	1457334	1	242889	0
7	1208348	1	159894	0	1155343	0	165049	0
8	1402808	1	162797	0	1403408	1	175426	0
9	815875	0	83018	0	790110	0	87790	0
10-	21970291	18	810251	1	21336486	17	865095	1
100-	4394526	3	12599	0	4624407	3	20785	0
1000-	1700648	1	245	0	1858682	1	849	0
10000-	5027501	4	4	0	6441864	5	68	0
Sum	120411278	100	71370567	100	120411278	100	71370567	100

Table 16 - Identification when using 12 byte hash blocks

Identification when using 8 byte hash blocks

Dup-licates	Identify only file				Identify file and position in it			
	Total hashes	%	Unique hashes	%	Total hashes	%	Unique hashes	%
Unique	55897006	46	55401434	87	55135192	45	55135192	86
2	9002234	7	4312923	6	8768614	7	4384307	6
3	4195432	3	1327283	2	4058067	3	1352689	2
4	2436133	2	566835	0	2441052	2	610263	0
5	2087731	1	391866	0	2013345	1	402669	0
6	1533445	1	235519	0	1515330	1	252555	0
7	1257461	1	164506	0	1192541	0	170363	0
8	1484682	1	170840	0	1473232	1	184154	0
9	882079	0	88873	0	847431	0	94159	0
10-	24723570	20	892846	1	23839285	19	955123	1
100-	7769745	6	23548	0	7840274	6	33986	0
1000-	2803115	2	586	0	3292225	2	1525	0

Identification when using 8 byte hash blocks

Dup-licates	Identify only file				Identify file and position in it			
	Total hashes	%	Unique hashes	%	Total hashes	%	Unique hashes	%
10000-	6338645	5	10	0	7994690	6	84	0
Sum	120411278	100	63577069	100	120411278	100	63577069	100

Table 17 - Identification when using 8 byte hash blocks

Identification when using 6 byte hash blocks

Dup-licates	Identify only file				Identify file and position in it			
	Total hashes	%	Unique hashes	%	Total hashes	%	Unique hashes	%
Unique	48824193	40	48358481	87	48126117	39	48126117	86
2	7214663	5	3410355	6	6910630	5	3455315	6
3	3764957	3	1183597	2	3611286	2	1203762	2
4	2187984	1	502292	0	2155832	1	538958	0
5	1802476	1	332582	0	1715160	1	343032	0
6	1455033	1	220859	0	1420410	1	236735	0
7	1206869	1	155797	0	1129296	0	161328	0
8	1429175	1	164100	0	1417216	1	177152	0
9	847321	0	84057	0	807354	0	89706	0
10-	25677835	21	894799	1	24621269	20	961728	1
100-	11996700	9	36224	0	11492986	9	47730	0
1000-	5807638	4	1474	0	6481827	5	2950	0
10000-	8196434	6	28	0	10521895	8	132	0
Sum	120411278	100	55344645	100	120411278	100	55344645	100

Table 18 - Identification when using 6 byte hash blocks

Identification when using 4 byte (32 bits) hash blocks (theoretical minimum)

Dup-licates	Identify only file				Identify file and position in it			
	Total hashes	%	Unique hashes	%	Total hashes	%	Unique hashes	%
Unique	39941903	33	39675949	88	39541108	32	39541108	88
2	4812812	3	2268050	5	4568212	3	2284106	5
3	2799446	2	881072	1	2669538	2	889846	1
4	1542418	1	353355	0	1489880	1	372470	0
5	1237203	1	226906	0	1162635	0	232527	0
6	1128163	0	170954	0	1075356	0	179226	0
7	960184	0	123402	0	887733	0	126819	0
8	1223850	1	142734	0	1198968	0	149871	0
9	670678	0	66119	0	627777	0	69753	0
10-	22804125	18	731806	1	21331344	17	779985	1
100-	18366696	15	54725	0	16836608	13	66891	0
1000-	14326218	11	3775	0	14310144	11	6028	0
10000-	10597582	8	62	0	14711975	12	279	0
Sum	120411278	100	44698909	100	120411278	100	44698909	100

Table 19 - Identification when using 12 byte (32 bits) hash blocks (theoretical minimum for this experiment if pure random data in database)

Identification when using 3 byte hash blocks (Unusable)

Dup-licates	Identify only file				Identify file and position in it			
	Total hashes	%	Unique hashes	%	Total hashes	%	Unique hashes	%
Unique	3561948	2	3548638	23	3541127	2	3541127	23
2	7813451	6	3885212	25	7741490	6	3870745	25
3	9056887	7	2996212	19	8945304	7	2981768	19
4	7550777	6	1864748	12	7422856	6	1855714	12
5	5188173	4	1020644	6	5085890	4	1017178	6
6	3268430	2	531482	3	3190068	2	531678	3
7	2060707	1	283857	1	2003169	1	286167	1
8	1427786	1	170273	1	1380744	1	172593	1
9	1110222	0	116656	0	1068894	0	118766	0
10-	17791741	14	615052	4	16338037	13	644677	4
100-	19357356	16	56920	0	17589951	14	66876	0
1000-	21011142	17	5026	0	18198143	15	7044	0
10000-	21212658	17	196	0	27905605	23	583	0
Sum	120411278	100	15094916	100	120411278	100	15094916	100

Table 20 - Identification when using 3 byte hash blocks

Identification when using 2 byte (16 bits) hash blocks (Unusable)

Dup-licates	Identify only file				Identify file and position in it			
	Total hashes	%	Unique hashes	%	Total hashes	%	Unique hashes	%
Unique	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0
10-	0	0	0	0	0	0	0	0
100-	40531490	33	53609	81	35674559	29	49537	75
1000-	36740701	30	11405	17	32292008	26	14929	22
10000-	43139087	35	522	0	52444711	43	1070	1
Sum	120411278	100	65536	100	120411278	100	65536	100

Table 21 - Identification when using 2 byte (16 bits) hash blocks (Unusable)

Identification when using 1 byte (8 bits) hash blocks (Unusable)

Dup-licates	Identify only file				Identify file and position in it			
	Total hashes	%	Unique hashes	%	Total hashes	%	Unique hashes	%
Unique	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0
10-	0	0	0	0	0	0	0	0
100-	0	0	0	0	0	0	0	0
1000-	0	0	0	0	0	0	0	0
10000-	120411278	100	256	100	120411278	100	256	100
Sum	120411278	100	256	100	120411278	100	256	100

Table 22 - Identification when using 1 byte (8 bits)hash blocks (Unusable)

8.2 Detailed database contents

Ref	Count	Ref	Count	Ref	Count	Ref	Count	Ref	Count
g40x8	80411	24ja	5448	24tr	3393	x1ru	2413	46en	1116
m923	76314	ss8rs	4673	24pt	3392	x1fr	2412	cs2en	1101
m922	65175	e3en	4478	24sv	3392	x1es	2410	on3cs	530
g41p3	64731	e23en	3925	24nl	3391	x1nl	2410	on3fr	514
g41i6	64730	spp3en	3627	e54en	3384	x1de	2409	on3ja	492
m103	60638	24fr	3458	24el	3218	x1hu	2409	on3sp	485
f521al1	58273	24es	3457	24ar	3215	x1it	2409	on3ko	480
g41at	56780	24de	3456	24he	3214	x1pt	2409	on3it	478
f521sp1	55964	24en	3452	24da	3213	x1br	2408	s23en	478
m101	53450	ss8rx	3434	24fi	3213	x1pl	2408	on3ge	471
m102	51087	24ko	3432	23en	3204	x1sv	2408	on3br	452
m921	50461	24hk	3425	24no	3197	x1tr	2408	on3en	451
srv03sp1	17093	24cn	3424	22en	2563	x1cs	2407	on3ct	448
x0922207	16437	24it	3410	x1en	2500	x1fi	2407	lc3en	219
x0945916	16033	24hu	3409	x1ja	2441	x1da	2404	s74en	189
f521al2	13121	24tw	3409	x1cn	2427	x1no	2404	mm	16
f521sp2	13048	24br	3408	x1ch	2424	x1ar	2401	logs	3
psdk	8760	24cs	3408	x1tw	2423	x1he	2400		
ss9rs	8516	24pl	3394	x1el	2416	21en	1919		
ss9rx	7654	24ru	3394	x1ko	2413	bt4en	1448		

Table 23 – Detailed distribution of source for the 1 025 856 files used in creating the database

Ext	Count	Ext	Count	Ext	Count	Ext	Count	Ext	Count
gz	70321	a	4314	3x	1630	ps	785	cp_	512
html	50376	cpio	3888	gpd	1463	hl_	764	9	504

dll	46666	java	3719	ch_	1400	cat	759	scm	501
dl_	40893	in_	3717	tbz2	1294	jar	740	dat	499
mo	36902	desktop	3440	tar	1283	exp	705	rb	498
h	20029	pyc	3221	makefile	1221	patch	675	as_	497
png	15517	el	3171	idl	1206	afm	674	sp_	496
so	13600	sys	3117	5	1201	n	664	xsl	493
bz2	13475	txt	2811	pl	1186	file	648	vf	490
exe	12005	0	2732	36432*	1155	pfb	647	pp_	487
htm	10630	ppd	2691	tfm	1083	pdb	641	xul	482
ex_	10183	gif	2651	js	1051	oc_	635	tbz	466
o	9169	pyo	2609	cgi	1047	sty	630	info	462
c	9127	py	2386	4	1040	cab	624	ttf	452
pcf	8795	8	2226	tex	1000	7	611	css	447
class	8200	res	2212	readme	970	tcl	586	dxf	440
docbook	7910	2	2205	asp	967	fo_	577	vim	434
1	6332	pm	1983	mf	910	scr	566	tt_	423
cache	5811	changelog	1979	wav	908	sc_	562	aspx	407
xml	5627	sy_	1961	level	860	htm_1033	554	glade	405
inf	5571	pkg-plist	1943	stw	848	lc_time	554		
3	5518	xpm	1915	jpg	835	en	550		
ko	5057	ebuild	1815	ocx	830	conf	535		
elc	4948	hlp	1769	pod	805	cpl	533		
rpm	4513	chm	1676	3qt	786	properties	524		

Table 24 – All file extensions occurring more than 400 times in the database

8.3 Suggested introductory Literature

The last year has seen quite a few books on Digital Forensics, especially targeted towards incident response on a corporate environment. The following two books by Eoghan Casey is written for a technical audience in a law enforcement environment, and are widely used and referenced in this setting. They are therefore a natural recommendation as introductory literature in this field.

- Casey, Eoghan: Handbook of Computer Crime Investigation, 2002, Academic Press, London

- Casey, Eoghan: Digital Evidence and Computer Crime - Forensic Science, Computers and the Internet, 2004, Academic Press, London

8.4 Practical Problems

This chapter outlines some of the practical problems encountered during the experiment. It is only of interest for those attempting to duplicate the results or perform similar experiments.

Approximately 360 GB disk space was used to produce the 60 GB database of unique files. This was mainly due to wide extensive use of batch processing, downloading all, extracting all, and then inserting all into the database. A more resource friendly method would have been to download, extract and insert the archives one by one.

Running several of the jobs was time consuming, especially the extracting and fragment check summing. It was not uncommon to have to wait for several days before the results were ready. In this scenario, debugging was very important, since a bug in a job not only could result in that the result of the job was unusable, but also that results from previous jobs were damaged. In this case, it seemed like the bottle neck on the desktop systems was mainly disk IO.

Directory depth and file names are limited to 256 bytes on a Windows NTFS volume, so this lead to some problems during extracting of deeply nested packages.
As an example; to fully extract all levels of the archive apache2-mod_php-2.0.48_4.3.4-1mdk.i586.rpm, the xAll script needed to make the directory structure apache2-mod_php-2.0.48_4.3.4-1mdk.i586.rpm.EXP\apache2-mod_php-2.0.48_4.3.4-1mdk.i586.cpio.gz.EXP\apache2-mod_php-2.0.48_4.3.4-1mdk.i586.cpio.EXP.
A next version of the script could be designed to choose the destination path in a different way.

The size of the data sets involved made them impossible to process with statistic software on the hardware available to the author. This meant that all processing had to be done by specially developed software.

The use of Microsoft Word as the text processor presented difficult problems. It was probably due to lack of knowledge of the tool, but it was a challenge to achieve the desired results when deadlines were fast approaching.

8.5 Script and Program Listings

8.5.1 xAll – Extract All

The following JPSoft 4NT 5.00 Batch Script was used to recursively extract archive files. It makes use of the multi-format archiver 7-zip to do the actual extraction. Archives are extracted to a directory named after the archive with an extra extension .EXP for further expansion.

```
@ECHO OFF
ECHO eXtract ALL in a subdirectory
ECHO This program must be run several times to extract all levels
ECHO.

SETLOCAL

SET ZZ=7z.exe

UNALIAS *
SET LISTF=%@UNIQUE[ %TEMP% ]
SET LOGF=%_CWD\xAll.LOG

ECHO Secarcing for non empty files with no description ...
DIR /F /S /A:-D /H /[s1] /I"[]" /[!DESCRIPT.ION]* > %LISTF%

ECHO Processing %@EVAL[ %@LINES[ "%LISTF%" ] + 1 ] files in
"%LISTF%" ...
ECHO Processing %@EVAL[ %@LINES[ "%LISTF%" ] + 1 ] files in
"%LISTF%" >> "%LOGF%"

FOR %F in (@"%LISTF%") (
  ECHO Extracting "%F%" ...
  ECHO Extracting "%F%" >> "%LOGF%"
```

```
  SET EXTF=%@FILENAME["%F%"]
  SET OUTD=%EXTF%.EXP
  PUSHD %@PATH["%F%"]
  %ZZ% x -aou -y -o"%OUTD%" "%EXTF%" >>& "%LOGF%"
  DESCRIBE "%EXTF%" /D"Extracted COUNT=%@FILES[
"%OUTD%\*" ] RC=%? DIR=%OUTD%"
  POPD
)

DEL %LISTF% >& NUL

ECHO Log of operations is in file "%LOGF%" .

ENDLOCAL
```

8.5.2 MD5sort – Sort files according to hash and find uniques.

The following JPSoft 4NT 5.00 Batch Script was used to sort the extracted files into a directory structure based on it's MD5 hash value. This was done both to be able to discard duplicate files, but also to facilitate quick lookup of the original file and information about the original origin. The directory structure was choosen because of the way NTFS is designed to sort directory entries by name [Casey 04]. This is also the exact way Microsoft itself has choosen to implement large file storage applications.

The script needs a MD5 checksum file in md5sum-format as input.

```
@ECHO OFF
ECHO Sort files according to MD5 hash value
ECHO USAGE: MD5sort {MD5file} {DestinationPath}
ECHO.

EVENTLOG MD5sort is starting with parameters %0$
SETLOCAL
SETDOS /D0
UNALIAS *

SET MD5FILE=%1
SET DESTINATIONPATH=%2

SET LOGF=%_CWD\MD5sort.LOG
```

```
FOR %line in (@%MD5FILE%) DO (
  SET MD=%@WORD[0,%line]
  SET FN=%@WORD["*",1,%line]
  SET
DP=%DESTINATIONPATH%\%@LEFT[1,%MD%]\%@INSTR[1,1,%
MD%]\%@INSTR[2,1,%MD%]\%MD%
  SET FDT=%@FILEDATE["%FN%",,4] %@FILETIME["%FN%",,s]
  MD /N /S %DP% >& NUL
  IF "%_?" == "0" (
    MOVE /H "%FN%" %DP%\content >>& %LOGF%
    SET DEST=%DP%\content
    ECHO CompanyName     %@VERINFO[%DEST%,CompanyName]
-  %DP%\verinfo
    ECHO FileDescription %@VERINFO[%DEST%,FileDescription] --
%DP%\verinfo
    ECHO FileVersion     %@VERINFO[%DEST%,FileVersion]     --
%DP%\verinfo
    ECHO ProductName     %@VERINFO[%DEST%,ProductName]     --
%DP%\verinfo
    ECHO ProductVersion  %@VERINFO[%DEST%,ProductVersion]  --
%DP%\verinfo  )
  ECHO %FN%%=t%FDT% >> %DP%\paths
)

ECHO Log of operations is in file "%LOGF%" .

ENDLOCAL
EVENTLOG MD5sort has finished
```

8.5.3 MD5frag – Calculate MD5 checksums of all fragments

The following .NET 1.1 C# program calculates MD5 hashes of all fragments used in this experiment.

The output is in ASCII format with one record on each line to make it possible to sort the tables using utilities preinstalled with operating systems. In order to make the sorting jobs manageable on regular desktop hardware, the output is preliminary sorted in files based on the first byte of the MD5 checksum. When all 256 files have been sorted, it is possible to just merge the files to create the main fragment database.

The program needs a list of all content files to process as input.

```
using System;
using System.IO;
using System.Security.Cryptography;

namespace MD5frag
{
 class ClassMain
 {
  [STAThread]
  static void Main(string[] args)
  {
   MD5 mMD5 = new MD5CryptoServiceProvider();
   byte[] bHash;
   byte[] bBlock = new Byte[512];

   StreamWriter[] swWriters512 = CreateWriters( "table.512.fragments." );
/*Repeated for every block size*/
   StreamWriter[] swWriters1  = CreateWriters( "table.1. fragments." );

   int iLineCounter = 0;

   Stream sFileList = File.OpenRead( args[0] );
   StreamReader srFileList = new StreamReader( sFileList );

   string sFileDataName = srFileList.ReadLine();
   while (sFileDataName != null)
   {
    string sFileDataHash = sFileDataName.Substring( 6, 32 );
    if (iLineCounter++ % 10 == 0 )
     Console.WriteLine( "Processing {0}.", sFileDataHash);

    Stream sFileData = File.OpenRead( sFileDataName );
    BufferedStream bsFileData = new BufferedStream( sFileData );

    int iCounter = 0;
    while ( bsFileData.Read( bBlock, 0, 512) == 512 )
    {
     bHash = mMD5.ComputeHash( bBlock );
     swWriters512[ bHash[0] ].WriteLine( "{0} {1} {2,5}", HashString( bHash ),
sFileDataHash, iCounter );

/*Repeated for every block size*/

     bHash = mMD5.ComputeHash( bBlock, 499, 1 );
     swWriters1 [ bHash[0] ].WriteLine( "{0} {1} {2,5}", HashString( bHash ),
sFileDataHash, iCounter );

     iCounter++;
    }
    bsFileData.Close();
    sFileDataName = srFileList.ReadLine();
```

```
      }
      srFileList.Close();

      CloseWriters( swWriters1 );
/*Repeated for every block size*/
      CloseWriters( swWriters512 );

    }

    static StreamWriter[] CreateWriters( string sFilename )
    {
      StreamWriter[] swWriters = new StreamWriter[256];

      for( int iCounter=0; iCounter <= 255 ; iCounter++ )
      {
        swWriters[ iCounter ] =
          new StreamWriter( sFilename + iCounter.ToString( "x2" ) );
      }
      return swWriters;
    }

    static void CloseWriters( StreamWriter[] swWriters )
    {
      for( int iCounter=0; iCounter <= 255 ; iCounter++ )
      {
        swWriters[iCounter].Close();
      }
    }

    static string HashString( byte[] hash )
    {
      return
        hash[0].ToString("x2") +
        hash[1].ToString("x2") +
        hash[2].ToString("x2") +
        hash[3].ToString("x2") +
        hash[4].ToString("x2") +
        hash[5].ToString("x2") +
        hash[6].ToString("x2") +
        hash[7].ToString("x2") +
        hash[8].ToString("x2") +
        hash[9].ToString("x2") +
        hash[10].ToString("x2") +
        hash[11].ToString("x2") +
        hash[12].ToString("x2") +
        hash[13].ToString("x2") +
        hash[14].ToString("x2") +
        hash[15].ToString("x2");
    }

  }
}
```

8.5.4 MD5stats – Identify duplicate fragments

The following .NET 1.1 C# program identifies fragments with the same MD5 checksum. It logs them to a separate file so they can be analyzed later.

It makes a special note if all the fragments are from the same position in different files. This makes it easier to spot cases where the same checksum is due to different language versions or revisions of the same file.

It also keeps track of which file the various fragments are from, making it possible to say something about why this particular fragment was a duplicate.

The program needs the main fragment database as input.

```
using System;
using System.IO;
using System.Collections;
using System.Collections.Specialized;

namespace MD5stats
{
 /// <summary>
 /// Summary description for Class1.
 /// </summary>
 class ClassMain
 {
  /// <summary>
  /// The main entry point for the application.
  /// </summary>
  [STAThread]
  static void Main(string[] args)
  {
   long lLineTotalCounter = 0;
   long lUniqueHashesTotalCounter = 0;
   long lDuplicateTotalCounter = 0;

   string sHashBlockOld = ""; // The last block hash
   string sHashFileOld = ""; // The last file hash

   int iDuplicateCount = 0; // Reset the number of duplicates for this hash

   StringDictionary sdDistributions = new StringDictionary();
   StringDictionary sdFileNames = new StringDictionary();
   StringDictionary sdFileHashes = new StringDictionary();
   StringDictionary sdBlocks = new StringDictionary();
```

```
  Stream sFileList = File.OpenRead( args[0] );
  StreamReader srFileList = new StreamReader( sFileList );

  Stream sFileDuplicates = File.OpenWrite( args[0] + ".duplicates" );
  StreamWriter swFileDuplicates = new StreamWriter( sFileDuplicates );

  Stream sFileDuplicatesNames = File.OpenWrite( args[0] + ".duplicates.names" );
  StreamWriter swFileDuplicatesNames = new StreamWriter( sFileDuplicatesNames );


  string sHashPair = srFileList.ReadLine();
  while (sHashPair != null)
  {
   string sHashBlock = sHashPair.Substring( 0, 32 );
   string sHashFile = sHashPair.Substring( 33, 32 );
   int iBlock = Int32.Parse( sHashPair.Substring( 66 ) );

   lLineTotalCounter++;

   //if (lLineTotalCounter % 4096 == 0 )
   //Console.WriteLine( "Block {0}, file {1}", sHashBlock, sHashFile );

   if ( sHashBlock != sHashBlockOld ) // A new block
   {
    if ( iDuplicateCount >= 2 ) // Are there some data to save (was the last one a duplicate)?
    {
     // HashBlock DuplicateCount FileHashCount BlockCount DistCount FileNameCount
     swFileDuplicates.WriteLine( "{0} {1,7} {2,7} {3,7} {4,7} {5,7}", sHashBlockOld,
iDuplicateCount, sdFileHashes.Count, sdBlocks.Count, sdDistributions.Count,
sdFileNames.Count );
     // HashBlock Dists FileNames Blocks
     swFileDuplicatesNames.WriteLine( "{0}\t{1}\t{2}\t{3}", sHashBlockOld,
KeysString( sdDistributions ), KeysString( sdFileNames ), KeysString( sdBlocks ) );
    }
    else
    {
     lUniqueHashesTotalCounter++;
    }

    iDuplicateCount = 1; // Reset the number of duplicates for this hash

    sdDistributions.Clear();
    sdFileNames.Clear();
    sdFileHashes.Clear();
    sdBlocks.Clear();
   }
   else // The same block as the last time (duplicate)
   {
    if (iDuplicateCount == 1) // It has not been buildt before
    {
     GetDistsAndNames( sHashFileOld, sdDistributions, sdFileNames );

     if ( ! sdFileHashes.ContainsKey( sHashFileOld ) )
      sdFileHashes.Add( sHashFileOld, "" );
```

```
    }

    lDuplicateTotalCounter++;
    iDuplicateCount++;

    if (sHashFile != sHashFileOld) // A new file
    {
     GetDistsAndNames( sHashFile, sdDistributions, sdFileNames );

     if ( ! sdFileHashes.ContainsKey( sHashFile ) )
      sdFileHashes.Add( sHashFile, "" );
    }


    if ( ! sdBlocks.ContainsKey( iBlock.ToString() ) )
     sdBlocks.Add( iBlock.ToString(), "" );
   }

   sHashBlockOld = sHashBlock; // Save the last block hash
   sHashFileOld = sHashFile;  // Save the last file hash

   sHashPair = srFileList.ReadLine();
  }

  swFileDuplicatesNames.Close();
  swFileDuplicates.Close();
  srFileList.Close();

  Console.WriteLine( "{0} lines processed. {1} were duplicates. {2} unique/single
instaces.", lLineTotalCounter, lDuplicateTotalCounter, lUniqueHashesTotalCounter );
 }


 static void GetDistsAndNames( string sHash, StringDictionary sdDistributions,
StringDictionary sdFileNames )
 {
  Stream sFilePaths = File.OpenRead( "F:\\Uniques\\" + sHash[0] + "\\" + sHash[1] + "\\"
+ sHash[2] + "\\" + sHash + "\\paths" );
  StreamReader srFilePaths = new StreamReader( sFilePaths );
  string sPaths = srFilePaths.ReadLine();
  while (sPaths != null)
  {
   string[] sSplit = sPaths.Split( new Char[] {'\\','\t'} );
   string sDistribution = sSplit[0]; // Get the distribution/ database reference name
   string sFileName = sSplit[ sSplit.Length - 2]; // Get the file name
   if (sFileName == "[Content]") // One of the special names
    sFileName = sSplit[ sSplit.Length - 3]; // Get a more descriptive file name

   if ( ! sdDistributions.ContainsKey( sDistribution ) )
    sdDistributions.Add( sDistribution, "" );
   if ( ! sdFileNames.ContainsKey( sFileName ) )
    sdFileNames.Add( sFileName, "" );
```

```
   //Console.WriteLine( "Adding hash {0}: {1}, {2}, {3}.", sHash, sFileName,
sdDistributions.Count, sdFileNames.Count );

    sPaths = srFilePaths.ReadLine();
   }
   srFilePaths.Close();
  }


  static string KeysString( StringDictionary sdKeys )
  {
   string sKeys = "";
   foreach ( DictionaryEntry deEntry in sdKeys )
    sKeys = sKeys + deEntry.Key + " ";
   return sKeys;
  }

 }
}
```

8.5.5 DBstats – Simple statistics about hash database content

The following .NET 1.1 C# program sums up simple statistics about the fragments in the database.

The program needs the main fragment database as input.

```
using System;
using System.IO;

namespace DBstats
{
 /// <summary>
 /// Summary description for Class1.
 /// </summary>
 class ClassMain
 {
  /// <summary>
  /// The main entry point for the application.
  /// </summary>
  [STAThread]
  static void Main(string[] args)
  {
   long[] lTFilesT = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}; // 0=total, 1-9=numbers,
10=10-99, 11=100-999, 12=1000-9999, 13=10000-
   long[] lTFilesU = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
   long[] lTFilesPosT = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
   long[] lTFilesPosU = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
/*   long[] lTFilesU = new Int64[14];
   long[] lTFilesPosT = new Int64[14];
   long[] lTFilesPosU = new Int64[14];*/
```

```
Console.WriteLine( "Processing file {0} ...", args[0] );

Stream sFileList = File.OpenRead( args[0] );
StreamReader srFileList = new StreamReader( sFileList );

// Initalize variables
string sHashPair = srFileList.ReadLine();
long lCFiles = 1; // Count the first instance of a file
long lCFilesPos = 1; // Count the first instance of a file and position pair
string sHashBlockOld = sHashPair.Substring( 0, 32 );
string sHashFileOld = sHashPair.Substring( 33, 32 );

// Read the first line to be processed
sHashPair = srFileList.ReadLine();

while (sHashPair != null)
{
 string sHashBlock = sHashPair.Substring( 0, 32 );
 string sHashFile = sHashPair.Substring( 33, 32 );

 if ( sHashBlock != sHashBlockOld ) // A new block
 {
  TableIncrement( lTFilesPosT, lTFilesPosU, lCFilesPos, lCFilesPos );
  TableIncrement( lTFilesT, lTFilesU, lCFilesPos, lCFiles );

  lCFilesPos = 1; // Count the first instance of a file and position pair
  lCFiles = 1; // Count the first instance of a file

  sHashBlockOld = sHashBlock; // Save the last block hash
  sHashFileOld = sHashFile;  // Save the last file hash
 }
 else // The same block as the last time (duplicate)
 {
  lCFilesPos ++;

  if (sHashFile != sHashFileOld) // A new file
  {
   lCFiles ++;

   sHashFileOld = sHashFile;  // Save the last file hash
  }
 }

 sHashPair = srFileList.ReadLine();
}

// Save off last data
TableIncrement( lTFilesPosT, lTFilesPosU, lCFilesPos, lCFilesPos );
TableIncrement( lTFilesT, lTFilesU, lCFilesPos, lCFiles );

srFileList.Close();
```

```
  Stream sFileStat = File.OpenWrite( args[0] + ".sq.TXT" );
  StreamWriter swFileStat = new StreamWriter( sFileStat );

  swFileStat.WriteLine( "Quick statistics for file {0}.", args[0] );

  swFileStat.WriteLine( "Identify both file and position" );
  TableWrite( lTFilesPosT, lTFilesPosU, swFileStat );

  swFileStat.WriteLine( "Identify file only" );
  TableWrite( lTFilesT, lTFilesU, swFileStat );

  swFileStat.WriteLine( "" );
  swFileStat.Close();
 }

 static void TableIncrement( long[] lTableTotal, long[] lTableUnique, long lHashCnt, long
lInstanceCnt )
 {
  // 0=total, 1=unique // 0=total, 1-9=numbers, 10=10-99, 11=100-999, 12=1000-9999,
13=10000-
  lTableTotal[0] += lHashCnt;
  lTableUnique[0] ++;

  if (lInstanceCnt < 10)
  {
   lTableTotal[lInstanceCnt] += lHashCnt;
   lTableUnique[lInstanceCnt] ++;
  }
  else if (lInstanceCnt < 100)
  {
   lTableTotal[10] += lHashCnt;
   lTableUnique[10] ++;
  }
  else if (lInstanceCnt < 1000)
  {
   lTableTotal[11] += lHashCnt;
   lTableUnique[11] ++;
  }
  else if (lInstanceCnt < 10000)
  {
   lTableTotal[12] += lHashCnt;
   lTableUnique[12] ++;
  }
  else
  {
   lTableTotal[13] += lHashCnt;
   lTableUnique[13] ++;
  }
 }

 static void TableWrite( long[] lTableTotal, long[] lTableUnique, StreamWriter swFile )
 {
  // 0=total, 1=unique // 0=total, 1-9=numbers, 10=10-99, 11=100-999, 12=1000-9999,
13=10000-
```

```
  swFile.WriteLine( "Desc\tTotal\tTotal %\tUnique\tUnique %");
  swFile.Flush();
  swFile.WriteLine( "Sum\t{0}\t100\t{1}\t100", lTableTotal[0], lTableUnique[0] );
  swFile.Flush();

  for (int iCnt=1; iCnt < 10; iCnt++ )
   swFile.WriteLine( "{0}\t{1}\t{2}\t{3}\t{4}", iCnt, lTableTotal[iCnt], lTableTotal[iCnt]
* 100 / lTableTotal[0], lTableUnique[iCnt], lTableUnique[iCnt] * 100 / lTableUnique[0] );

  swFile.WriteLine( "10-\t{0}\t{1}\t{2}\t{3}", lTableTotal[10], lTableTotal[10] * 100 /
lTableTotal[0], lTableUnique[10], lTableUnique[10] * 100 / lTableUnique[0] );
  swFile.WriteLine( "100-\t{0}\t{1}\t{2}\t{3}", lTableTotal[11], lTableTotal[11] * 100 /
lTableTotal[0], lTableUnique[11], lTableUnique[11] * 100 / lTableUnique[0] );
  swFile.WriteLine( "1000-\t{0}\t{1}\t{2}\t{3}", lTableTotal[12], lTableTotal[12] * 100 /
lTableTotal[0], lTableUnique[12], lTableUnique[12] * 100 / lTableUnique[0] );
  swFile.WriteLine( "10000-\t{0}\t{1}\t{2}\t{3}", lTableTotal[13], lTableTotal[13] * 100
/ lTableTotal[0], lTableUnique[13], lTableUnique[13] * 100 / lTableUnique[0] );

 }

}
}
```

8.5.6 SourceStat – Simple Statistics about the source of files

The following .NET 1.1 C# program counts and produces simple statistics based on the source of the fragments in the hash database.

The program needs the main fragment database as input

```
using System;
using System.IO;
using System.Collections;
using System.Collections.Specialized;

namespace SourceStat
{
 /// <summary>
 /// Summary description for Class1.
 /// </summary>
 class ClassMain
 {
  /// <summary>
  /// The main entry point for the application.
  /// </summary>
  [STAThread]
  static void Main(string[] args)
  {
   long lUniqueCounter = 0;
```

```
  long lTotalCounter = 0;

  StringDictionary sdDistributions = new StringDictionary();
  StringDictionary sdExtensions = new StringDictionary();


  Stream sFileList = File.OpenRead( args[0] );
  StreamReader srFileList = new StreamReader( sFileList );

  string sLine = srFileList.ReadLine();
  while (sLine != null)
  {
   string sHash = sLine.Substring( 6, 32 );

   lUniqueCounter ++;
   lTotalCounter += GetDistsAndExts( sHash, sdDistributions, sdExtensions );

   sLine = srFileList.ReadLine();
  }
  srFileList.Close();

  Stream sFileStat = File.OpenWrite( "C:\\Master\\Sources.Stat.TXT" );
  StreamWriter swFileStat = new StreamWriter( sFileStat );

  swFileStat.WriteLine( "Information about a total of {0} files, {1} uniques.",
lTotalCounter, lUniqueCounter );

  swFileStat.WriteLine( "Distributions" );
  TableWrite( sdDistributions, swFileStat );

  swFileStat.WriteLine( "File Extensions" );
  TableWrite( sdExtensions, swFileStat );

  swFileStat.WriteLine( "" );
  swFileStat.Close();

 }

 static void TableWrite( StringDictionary sdDict, StreamWriter swFile )
 {
  swFile.WriteLine( "Entry\tCount");

  foreach ( DictionaryEntry deEntry in sdDict )
  {
   swFile.WriteLine( "{0}\t{1}", deEntry.Key, deEntry.Value );
  }
 }

 static long GetDistsAndExts( string sHash, StringDictionary sdDistributions,
StringDictionary sdExtensions )
 {
  StringDictionary sdDstTmp = new StringDictionary();;
  StringDictionary sdExtTmp = new StringDictionary();;
```

```
  long lPathsCounter = 0;
  Stream sFilePaths = File.OpenRead( "F:\\Uniques\\" + sHash[0] + "\\" + sHash[1] + "\\"
+ sHash[2] + "\\" + sHash + "\\paths" );
  StreamReader srFilePaths = new StreamReader( sFilePaths );
  string sPaths = srFilePaths.ReadLine();
  while (sPaths != null)
  {
   lPathsCounter++;
   string[] sSplit = sPaths.Split( new Char[] {'\\','\t'} );
   string sDistribution = sSplit[0]; // Get the distribution/ database reference name
   string sFileName = sSplit[ sSplit.Length - 2]; // Get the file name
   if (sFileName == "[Content]") // One of the special names
    sFileName = sSplit[ sSplit.Length - 3]; // Get a more descriptive file name

   sSplit = sFileName.Split( new Char[] {'.'} );
   string sExtension = sSplit[ sSplit.Length - 1]; // Get the extension
   if (sExtension == "exp") // One of the special names
   {
    sExtension = sSplit[ sSplit.Length - 2] + ".exp"; // Get a more descriptive extension
    if (sExtension == "tbz.exp" || sExtension == "aa.exp")
    {
     sExtension = "tar";
    }
   }

   if ( !sdDstTmp.ContainsKey( sDistribution ) )
    sdDstTmp.Add( sDistribution, "1" );

   if ( !sdExtTmp.ContainsKey( sExtension ) )
    sdExtTmp.Add( sExtension, "1" );

   sPaths = srFilePaths.ReadLine();
  }
  srFilePaths.Close();

  foreach ( DictionaryEntry deEntry in sdDstTmp )
   if ( sdDistributions.ContainsKey( deEntry.Key.ToString() ) )
    sdDistributions[ deEntry.Key.ToString() ] = IncString( sdDistributions[
deEntry.Key.ToString() ] );
   else
    sdDistributions.Add( deEntry.Key.ToString(), "1" );

  foreach ( DictionaryEntry deEntry in sdExtTmp )
   if ( sdExtensions.ContainsKey( deEntry.Key.ToString() ) )
    sdExtensions[ deEntry.Key.ToString() ] = IncString( sdExtensions[
deEntry.Key.ToString() ] );
   else
    sdExtensions.Add( deEntry.Key.ToString(), "1" );

  return ( lPathsCounter );

 }

 static string IncString( string sValue )
```

```
 {
  long lValue = Int64.Parse( sValue ) + 1;
  return ( lValue.ToString() );
 }

 }
}
```

8.6 The Multiple Format Extraction Tool 7-Zip

The extraction tool used was the open source project 7-Zip from http://www.7-zip.org/ .

Since it can expand and extract files at least from files of types MSI, CAB, ??_, RPM, DEB TAR, CPIO, GZIP, BZIP2, ZIP, RAR and ARJ, and also was able to correctly identify the type of archive without relying on file extension, it was very suitable as a universal extraction tool for this project.

8.7 Bill of Materials

The following lists the archive files used to populate the database for this experiment. The archives themselves were not a part of the database, but all the files they produced when recursively expanded were .

Microsoft Windows Server 2003, Enterprise Edition, for 64-bit Extended Systems. 17093 files.
File sized 519030784, dated 07.09.03 07:17. Database reference srv03sp1.
http://download.microsoft.com/download/c/9/b/c9ba876f-414f-4dd1-bf25-a84891278b69/srv03sp1_usa_1069_amd64fre_ads.iso

Microsoft Windows Server 2003, Enterprise Edition, 32-bit. 16437 files.
File sized 569346048, dated 22.04.03 01:06. Database reference x0922207.
http://download.microsoft.com/download/5/2/b/52beb621-1a93-41bb-a57d-ea5c7aef7f76/x09-22207.iso

Microsoft Windows Server 2003, Enterprise Edition, for 64-bit Itanium-Based Systems. 16033 files.

File sized 596054016, dated 22.04.03 02:03. Database reference x0945916. http://download.microsoft.com/download/4/c/c/4cc5a2fe-c8cf-4f58-8d4b-b21e76f5bcf9/x09-45916.iso

Microsoft Windows XP Service Pack 1a Network Installation, Arabic. 2401 files.
File sized 128887392, dated 27.01.03 22:20. Database reference x1ar. http://download.microsoft.com/download/d/a/2/da2a084c-3349-4cce-96be-d8f79126f58a/xpsp1a_ar_x86.exe

Microsoft Windows XP Service Pack 1a Network Installation, Portuguese (Brazilian). 2408 files.
File sized 129117792, dated 27.01.03 21:02. Database reference x1br. http://download.microsoft.com/download/5/8/6/5862ce05-4993-4a0c-82f8-576909fe3a77/xpsp1a_br_x86.exe

Microsoft Windows XP Service Pack 1a Network Installation, Chinese (Simplified). 2424 files.
File sized 147856480, dated 27.01.03 21:03. Database reference x1cn. http://download.microsoft.com/download/a/4/9/a49ca543-097f-4e57-b8e5-ee1090f7f022/xpsp1a_cn_x86.exe

Microsoft Windows XP Service Pack 1a Network Installation, Chinese (Hong Kong)
File sized 129710688, dated 04.04.03 22:55. Database reference x1cn. http://download.microsoft.com/download/e/7/d/e7da5bdc-8407-4739-9a5e-0967987e941a/xpsp1a.exe

Microsoft Windows XP Service Pack 1a Network Installation, Czech
File sized 129511008, dated 27.01.03 21:05. Database reference x1cs. http://download.microsoft.com/download/2/b/5/2b5361c9-c38b-402e-8066-f830e82bfffc/xpsp1a_cs_x86.exe

Microsoft Windows XP Service Pack 1a Network Installation, Danish File sized 129143904, dated 27.01.03 21:04. Database reference x1da. http://download.microsoft.com/download/5/8/7/5876b6c3-995f-43f0-9580-32d92f91205a/xpsp1a_da_x86.exe

Microsoft Windows XP Service Pack 1a Network Installation, German File sized 129049184, dated 27.01.03 20:56. Database reference x1de. http://download.microsoft.com/download/a/4/c/a4ce1a3b-fac8-4597-be33-c10ced905c3b/xpsp1a_de_x86.exe

Microsoft Windows XP Service Pack 1a Network Installation, Greek File sized 129707616, dated 27.01.03 21:17. Database reference x1el. http://download.microsoft.com/download/d/4/b/d4bf1ef1-3a5b-4013-a745-5779c11dad1b/xpsp1a_el_x86.exe

Microsoft Windows XP Service Pack 1a Network Installation, English File sized 131170400, dated 24.01.03 22:18. Database reference x1en. http://download.microsoft.com/download/5/4/f/54f8bcf8-bb4d-4613-8ee7-db69d01735ed/xpsp1a_en_x86.exe

Microsoft Windows XP Service Pack 1a Network Installation, Spanish File sized 129186400, dated 27.01.03 21:03. Database reference x1es. http://download.microsoft.com/download/0/d/d/0ddcd740-012b-4e45-84d0-52d63dbec578/xpsp1a_es_x86.exe

Microsoft Windows XP Service Pack 1a Network Installation, Finnish File sized 129474656, dated 27.01.03 22:21. Database reference x1fi. http://download.microsoft.com/download/1/8/5/1853ee20-2800-4e99-a9b6-012165be675c/xpsp1a_fi_x86.exe

Microsoft Windows XP Service Pack 1a Network Installation, French File sized 129283168, dated 27.01.03 20:59. Database reference x1fr.

http://download.microsoft.com/download/c/5/5/c5582838-ac1a-4edb-ade5-035c949edd69/xpsp1a_fr_x86.exe

Microsoft Windows XP Service Pack 1a Network Installation, Hebrew File sized 128788064, dated 27.01.03 22:22. Database reference x1he. http://download.microsoft.com/download/2/4/5/245ef5d9-a09c-4ed1-8fea-f5ba8449992c/xpsp1a_he_x86.exe

Microsoft Windows XP Service Pack 1a Network Installation, Hungarian File sized 129642080, dated 27.01.03 20:55. Database reference x1hu. http://download.microsoft.com/download/e/2/2/e221c26d-492c-4856-b644-2a2e7d9ce2e3/xpsp1a_hu_x86.exe

Microsoft Windows XP Service Pack 1a Network Installation, Italian File sized 129048672, dated 27.01.03 22:22. Database reference x1it. http://download.microsoft.com/download/6/a/7/6a727e5c-c84b-45ef-b943-b0080acfe352/xpsp1a_it_x86.exe

Microsoft Windows XP Service Pack 1a Network Installation, Japanese File sized 131510368, dated 27.01.03 20:59. Database reference x1ja. http://download.microsoft.com/download/c/8/0/c8092bab-f1b8-4119-8bcf-2e8dab528c25/xpsp1a_ja_x86.exe

Microsoft Windows XP Service Pack 1a Network Installation, Korean File sized 129086560, dated 27.01.03 20:50. Database reference x1ko. http://download.microsoft.com/download/3/2/3/32360c0e-b42c-4903-b7ab-b93f601d4893/xpsp1a_ko_x86.exe

Microsoft Windows XP Service Pack 1a Network Installation, Dutch File sized 129155168, dated 27.01.03 21:00. Database reference x1nl. http://download.microsoft.com/download/8/2/8/828e2216-e87c-4a0e-a056-c83da3e781f2/xpsp1a_nl_x86.exe

Microsoft Windows XP Service Pack 1a Network Installation, Norwegian
File sized 128893536, dated 27.01.03 21:05. Database reference x1no.
http://download.microsoft.com/download/2/0/c/20c376d4-664b-4fbb-8c94-a91f5839ca2d/xpsp1a_no_x86.exe

Microsoft Windows XP Service Pack 1a Network Installation, Polish
File sized 129326176, dated 27.01.03 21:14. Database reference x1pl.
http://download.microsoft.com/download/8/1/4/814a9d5f-54e0-43ee-b1b5-5509101f3e7b/xpsp1a_pl_x86.exe

Microsoft Windows XP Service Pack 1a Network Installation, Portuguese
File sized 129053280, dated 27.01.03 21:18. Database reference x1pt.
http://download.microsoft.com/download/7/7/f/77f8e1de-8c5d-4b5e-9de8-23d58506b20e/xpsp1a_pt_x86.exe

Microsoft Windows XP Service Pack 1a Network Installation, Russian
File sized 129240672, dated 27.01.03 21:23. Database reference x1ru.
http://download.microsoft.com/download/f/d/7/fd7a4d93-1cf6-40ac-93c9-c99338aa95ec/xpsp1a_ru_x86.exe

Microsoft Windows XP Service Pack 1a Network Installation, Swedish
File sized 128775264, dated 27.01.03 22:11. Database reference x1sv.
http://download.microsoft.com/download/b/a/5/ba56e88a-a9eb-4ab5-ab8e-b50f43d42b1b/xpsp1a_sv_x86.exe

Microsoft Windows XP Service Pack 1a Network Installation, Turkish
File sized 129104968, dated 26.02.03 04:31. Database reference x1tr.
http://download.microsoft.com/download/8/f/6/8f6ab9e4-9a82-4b25-8d14-6f0718cc2174/xpsp1a_tr_x86.exe

Microsoft Windows XP Service Pack 1a Network Installation, Chinese (Traditional)
File sized 129694816, dated 27.01.03 21:02. Database reference x1tw.

http://download.microsoft.com/download/5/8/b/58b518f7-94d6-4bd7-a333-4a55e4e2c662/xpsp1a_tw_x86.exe

Microsoft Windows 2000 Service Pack 4 Network Installation, Arabic
File sized 117102392, dated 20.06.03 22:28. Database reference 24ar.
http://download.microsoft.com/download/3/f/8/3f8a308a-eae1-4a2d-927d-1f76d3e60442/w2ksp4_ar.exe

Microsoft Windows 2000 Service Pack 4 Network Installation, Portuguese (Brazilian)
File sized 133702888, dated 21.06.03 01:37. Database reference 24br.
http://download.microsoft.com/download/f/a/0/fa0ac2ef-53b3-4ff0-af13-ba383610e31e/w2ksp4_br.exe

Microsoft Windows 2000 Service Pack 4 Network Installation, Chinese (Simplified)
File sized 134405296, dated 20.06.03 21:51. Database reference 24cn.
http://download.microsoft.com/download/4/1/4/4140e2e0-0ad9-4438-ac52-da0e0429c0e6/w2ksp4_cn.exe

Microsoft Windows 2000 Service Pack 4 Network Installation, Czech
File sized 133962744, dated 21.06.03 23:32. Database reference 24cs.
http://download.microsoft.com/download/f/d/2/fd2b1d2a-16c8-43ba-b805-518dbff8a438/w2ksp4_cs.exe

Microsoft Windows 2000 Service Pack 4 Network Installation, Danish
File sized 116951736, dated 21.06.03 01:09. Database reference 24da.
http://download.microsoft.com/download/7/c/7/7c7b3abd-2bfc-4e0d-b808-f4abdfc1c5f1/w2ksp4_da.exe

Microsoft Windows 2000 Service Pack 4 Network Installation, German
File sized 135945992, dated 20.06.03 10:53. Database reference 24DE.

http://download.microsoft.com/download/C/3/5/C35591E3-52B6-4BE0-95D3-EC82FA01CE12/W2KSP4_DE.EXE

Microsoft Windows 2000 Service Pack 4 Network Installation, Greek File sized 117420152, dated 21.06.03 01:00. Database reference 24el. http://download.microsoft.com/download/1/3/5/135cb641-c2a7-4e1b-95a0-69f75dc31674/w2ksp4_el.exe

Microsoft Windows 2000 Service Pack 4 Network Installation, English File sized 135477136, dated 20.06.03 10:30. Database reference 24EN. http://download.microsoft.com/download/E/6/A/E6A04295-D2A8-40D0-A0C5-241BFECD095E/W2KSP4_EN.EXE

Microsoft Windows 2000 Service Pack 4 Network Installation, Spanish File sized 135157672, dated 21.06.03 01:21. Database reference 24es. http://download.microsoft.com/download/0/d/0/0d02572f-e667-4bd7-a2d7-4b1a4328a711/w2ksp4_es.exe

Microsoft Windows 2000 Service Pack 4 Network Installation, Finnish File sized 116970400, dated 21.06.03 01:02. Database reference 24fi. http://download.microsoft.com/download/e/7/a/e7ae1b1f-3ef9-43f3-8df6-4bdb954e52c0/w2ksp4_fi.exe

Microsoft Windows 2000 Service Pack 4 Network Installation, French File sized 135498520, dated 21.06.03 02:02. Database reference 24fr. http://download.microsoft.com/download/c/f/0/cf065ae8-0c96-47f5-ba2b-3220da77076e/w2ksp4_fr.exe

Microsoft Windows 2000 Service Pack 4 Network Installation, Hebrew File sized 117048040, dated 20.06.03 22:23. Database reference 24he. http://download.microsoft.com/download/f/0/6/f06f92c9-0ba8-4522-aa5c-28de26efa7ab/w2ksp4_he.exe

Microsoft Windows 2000 Service Pack 4 Network Installation, Chinese (Hong Kong)
File sized 134670224, dated 20.06.03 21:44. Database reference 24hk.
http://download.microsoft.com/download/c/c/a/ccafe1fa-c4a2-4851-a2a9-eb973e2c3acb/W2KSP4_hk.EXE

Microsoft Windows 2000 Service Pack 4 Network Installation, Hungarian
File sized 134008064, dated 03.07.03 03:55. Database reference 24HU.
http://download.microsoft.com/download/f/2/f/f2f70a3e-943b-4ef3-806b-91a4f2d12df6/W2KSP4_HU.EXE

Microsoft Windows 2000 Service Pack 4 Network Installation, Italian
File sized 133231760, dated 21.06.03 02:01. Database reference 24it.
http://download.microsoft.com/download/4/3/9/4399958a-cae3-48cf-9fc3-2a36daab3a5e/w2ksp4_it.exe

Microsoft Windows 2000 Service Pack 4 Network Installation, Japanese NEC
File sized 136766896, dated 20.06.03 13:23. Database reference 24ja.
http://download.microsoft.com/download/9/7/8/978cd2a3-94df-4308-8889-1d7d30bb7623/W2KSP4_ja.EXE

Microsoft Windows 2000 Service Pack 4 Network Installation, Japanese
File sized 138817112, dated 20.06.03 12:45. Database reference 24ja.
http://download.microsoft.com/download/a/0/0/a008c376-a5fc-4219-9ba7-7b61d1236fc0/W2KSP4_ja.EXE

Microsoft Windows 2000 Service Pack 4 Network Installation, Korean
File sized 135493952, dated 20.06.03 21:24. Database reference 24KO.
http://download.microsoft.com/download/d/3/2/d32db0d3-9420-4568-b0ed-57a3060fb1ea/W2KSP4_KO.EXE

Microsoft Windows 2000 Service Pack 4 Network Installation, Dutch
File sized 133457176, dated 21.06.03 01:11. Database reference 24nl.

http://download.microsoft.com/download/1/b/7/1b7b823b-e760-4aa3-99d6-d9843cf71e7f/w2ksp4_nl.exe

Microsoft Windows 2000 Service Pack 4 Network Installation, Norwegian File sized 116836560, dated 21.06.03 01:20. Database reference 24no. http://download.microsoft.com/download/2/4/c/24ce4080-c34c-4636-bc01-8eb900f5021e/w2ksp4_no.exe

Microsoft Windows 2000 Service Pack 4 Network Installation, Polish File sized 134248176, dated 21.06.03 02:48. Database reference 24pl. http://download.microsoft.com/download/a/3/2/a32416d3-941d-424a-9310-21951f74216f/w2ksp4_pl.exe

Microsoft Windows 2000 Service Pack 4 Network Installation, Portuguese File sized 133672824, dated 21.06.03 02:18. Database reference 24pt. http://download.microsoft.com/download/a/7/4/a74b413f-d85c-4c0f-8170-2bd9b9ac7a7f/w2ksp4_pt.exe

Microsoft Windows 2000 Service Pack 4 Network Installation, Russian File sized 133952584, dated 21.06.03 03:46. Database reference 24ru. http://download.microsoft.com/download/6/1/1/611e63b8-95ec-40f4-b96c-e676f96d1a95/w2ksp4_ru.exe

Microsoft Windows 2000 Service Pack 4 Network Installation, Swedish File sized 134273616, dated 21.06.03 03:42. Database reference 24sv. http://download.microsoft.com/download/4/2/6/4261d8ed-167d-4b45-9757-c0f2828e94fa/w2ksp4_sv.exe

Microsoft Windows 2000 Service Pack 4 Network Installation, Turkish File sized 133527376, dated 21.06.03 03:01. Database reference 24tr. http://download.microsoft.com/download/6/a/d/6ad2327b-836f-4e38-9a83-d305793a9a8d/w2ksp4_tr.exe

Microsoft Windows 2000 Service Pack 4 Network Installation, Chinese (Traditional)
File sized 134668304, dated 20.06.03 21:57. Database reference 24tw.
http://download.microsoft.com/download/0/c/9/0c9ff7be-5ac4-4b44-badf-783dfc53b307/W2KSP4_tw.EXE

Microsoft Windows 2000 Service Pack 3 Network Installation, English
File sized 130978672, dated 23.07.02 19:23. Database reference 23en.
http://download.microsoft.com/download/win2000platform/SP/SP3/NT5/EN-US/W2Ksp3.exe

Microsoft Windows 2000 Service Pack 2 Network Installation, English
File sized 106278016, dated 07.05.01 14:34. Database reference 22en.
http://download.microsoft.com/download/win2000platform/SP/SP2/NT5/EN-US/W2KSP2.exe

Microsoft Windows 2000 Service Pack 1 Network Installation, English
File sized 87326656, dated 25.07.00 03:11. Database reference 21en.
http://download.microsoft.com/download/win2000platform/SP/SP1/NT5/EN-US/sp1network.exe

Microsoft Windows NT 4.0 Service Pack 6a Full Install, Standard Version for Intel (x86) platform, English Language Version
File sized 35720696, dated 20.11.99 02:50. Database reference 46en.
http://download.microsoft.com/download/winntsp/Install/6.0a/NT4/EN-US/sp6i386.exe

Microsoft Exchange 2000 Server and Exchange 2000 Enterprise Server Service Pack 3, English
File sized 173117728, dated 16.07.02 21:00. Database reference e23en.
http://download.microsoft.com/download/exchangeentserver/SP/3/NT5/en-us/EX2KSP3_server.exe

Microsoft Exchange Server 5.5 Service Pack 4, English
File sized 23735760, dated 30.10.00 20:36. Database reference e54en.
http://download.microsoft.com/download/exch55/SP/4/NT45/EN-US/SP4_550I.EXE
File sized 16816080, dated 30.10.00 20:37. Database reference e54en.
http://download.microsoft.com/download/exch55/SP/4/NT45/EN-US/SP4_55CI.EXE
File sized 15764944, dated 30.10.00 20:37. Database reference e54en.
http://download.microsoft.com/download/exch55/SP/4/NT45/EN-US/SP4_55client1I.exe

Microsoft SQL Server 2000 Service Pack 3a, English
File sized 45687344, dated 16.05.03 03:25. Database reference s23en.
http://download.microsoft.com/download/8/7/5/875e38ea-e582-4ee2-9485-b459cd9c0082/sql2kasp3.exe
File sized 72514184, dated 24.07.03 01:53. Database reference s23en.
http://download.microsoft.com/download/8/7/5/875e38ea-e582-4ee2-9485-b459cd9c0082/sql2kdesksp3.exe
File sized 57807120, dated 16.05.03 03:26. Database reference s23en.
http://download.microsoft.com/download/8/7/5/875e38ea-e582-4ee2-9485-b459cd9c0082/sql2ksp3.exe

Microsoft SQL Server 7.0 Service Pack 4, English
File sized 43843304, dated 23.04.02 03:13. Database reference s74en.
http://download.microsoft.com/download/sql70/SP/7.00.1063/W98NT42KMeXP/EN-US/sql70sp4.exe

Microsoft BizTalk Server 2004 Trial, English
File sized 151749328, dated 30.01.04 04:31. Database reference bt4en.
http://download.microsoft.com/download/4/1/2/4121c8be-6034-4cae-ab1b-74caa1644ed1/BTS2004Eval_EN.exe

Microsoft Commerce Server 2002, English
File sized 169531816, dated 04.04.02 02:37. Database reference cs2en.
http://download.microsoft.com/download/commerceserver2002/eval/1.0/NT5/EN-US/CS2002Evaluation.exe

Microsoft Live Communications Server 2003 Standard Edition Trial Version, English
File sized 50835640, dated 18.08.03 21:19. Database reference lc3en.
http://download.microsoft.com/download/a/8/b/a8baa907-3c15-46f1-b510-cc031c297cd5/LiveCommServer03_USA_RTM_Eval.exe

Microsoft Office SharePoint Portal Server 2003 - Evaluation Edition, English
File sized 301920864, dated 08.09.03 19:06. Database reference spp3en.
http://download.microsoft.com/download/7/7/d/77d791f5-49ed-4eb6-8672-133e39555c1c/SPS2003.exe

Microsoft Exchange Server 2003 Enterprise Edition Evaluation, English
File sized 122340456, dated 25.06.03 06:58. Database reference e3en.
http://download.microsoft.com/download/4/1/9/4195df06-8694-4d7b-ad94-a0003f139e97/E3EEVAENG.EXE

Microsoft OneNote 2003 (Office) 60-Day Trial, Brazilian Portugese
File sized 78284344, dated 17.09.03 21:18. Database reference on3Br.
http://download.microsoft.com/download/9/f/0/9f03a77b-f10b-4583-98ea-9ea15c2d0c95/OneNote.exe

Microsoft OneNote 2003 (Office) 60-Day Trial, Chinese Simplified
File sized 247750200, dated 09.09.03 03:15. Database reference on3CS.
http://download.microsoft.com/download/d/6/2/d62ddd42-ba16-4a8d-b775-d36ce90d6900/OneNote.exe

Microsoft OneNote 2003 (Office) 60-Day Trial, Chinese Traditional
File sized 131291192, dated 09.09.03 03:05. Database reference on3CT.

http://download.microsoft.com/download/b/a/c/bac50b46-d1f3-40fb-9447-db6855cdd029/OneNote.exe

Microsoft OneNote 2003 (Office) 60-Day Trial, English
File sized 79259704, dated 09.09.03 02:50. Database reference on3En.
http://download.microsoft.com/download/9/d/e/9decc1cf-3066-495c-bb8a-265e5d395312/OneNote.exe

Microsoft OneNote 2003 (Office) 60-Day Trial, French
File sized 104526392, dated 17.09.03 20:10. Database reference on3Fr.
http://download.microsoft.com/download/8/7/d/87d250c4-03f0-4b35-a230-7db753434322/OneNote.exe

Microsoft OneNote 2003 (Office) 60-Day Trial, German
File sized 90468408, dated 09.09.03 03:06. Database reference on3Ge.
http://download.microsoft.com/download/d/5/9/d599bfa5-c1b4-4098-ba78-becfacd87ec1/OneNote.exe

Microsoft OneNote 2003 (Office) 60-Day Trial, Italian
File sized 92341304, dated 17.09.03 13:23. Database reference on3It.
http://download.microsoft.com/download/9/0/F/90F7E507-6E77-43A3-A0EC-A2A7BC1FCD9D/OneNote.exe

Microsoft OneNote 2003 (Office) 60-Day Trial, Japanese
File sized 152461368, dated 09.09.03 03:11. Database reference on3Ja.
http://download.microsoft.com/download/a/1/b/a1bce172-fd3a-4feb-ad86-0da0040b7ee6/OneNote.exe

Microsoft OneNote 2003 (Office) 60-Day Trial, Korean
File sized 123904568, dated 09.09.03 03:11. Database reference on3Ko.
http://download.microsoft.com/download/1/3/9/13924aaa-cf3d-4510-952a-62d81dcc7391/OneNote.exe

Microsoft OneNote 2003 (Office) 60-Day Trial, Spanish
File sized 88638008, dated 17.09.03 21:43. Database reference on3Sp.
http://download.microsoft.com/download/6/d/6/6d69123f-5d39-4f42-be8b-0f69162d134b/OneNote.exe

Microsoft Windows Platform SDK, English
File sized 26221140, dated 02.04.03 02:28. Database reference psdk.
http://download.microsoft.com/download/platformsdk/sdk/update/win98mexp/en-us/3790.0/FULL/PSDK-FULL.1.cab
File sized 26221140, dated 02.04.03 02:28. Database reference psdk.
http://download.microsoft.com/download/platformsdk/sdk/update/win98mexp/en-us/3790.0/FULL/PSDK-FULL.2.cab
File sized 26221140, dated 02.04.03 02:29. Database reference psdk.
http://download.microsoft.com/download/platformsdk/sdk/update/win98mexp/en-us/3790.0/FULL/PSDK-FULL.3.cab
File sized 26221140, dated 02.04.03 02:29. Database reference psdk.
http://download.microsoft.com/download/platformsdk/sdk/update/win98mexp/en-us/3790.0/FULL/PSDK-FULL.4.cab
File sized 26221140, dated 02.04.03 02:31. Database reference psdk.
http://download.microsoft.com/download/platformsdk/sdk/update/win98mexp/en-us/3790.0/FULL/PSDK-FULL.5.cab
File sized 26221140, dated 02.04.03 02:31. Database reference psdk.
http://download.microsoft.com/download/platformsdk/sdk/update/win98mexp/en-us/3790.0/FULL/PSDK-FULL.6.cab
File sized 26221140, dated 02.04.03 02:32. Database reference psdk.
http://download.microsoft.com/download/platformsdk/sdk/update/win98mexp/en-us/3790.0/FULL/PSDK-FULL.7.cab
File sized 26221140, dated 02.04.03 02:32. Database reference psdk.
http://download.microsoft.com/download/platformsdk/sdk/update/win98mexp/en-us/3790.0/FULL/PSDK-FULL.8.cab
File sized 26221140, dated 02.04.03 02:32. Database reference psdk.

http://download.microsoft.com/download/platformsdk/sdk/update/win98mexp/en-us/3790.0/FULL/PSDK-FULL.9.cab
File sized 26221140, dated 02.04.03 02:28. Database reference psdk.
http://download.microsoft.com/download/platformsdk/sdk/update/win98mexp/en-us/3790.0/FULL/PSDK-FULL.10.cab
File sized 26221140, dated 02.04.03 02:29. Database reference psdk.
http://download.microsoft.com/download/platformsdk/sdk/update/win98mexp/en-us/3790.0/FULL/PSDK-FULL.11.cab
File sized 26221140, dated 02.04.03 02:29. Database reference psdk.
http://download.microsoft.com/download/platformsdk/sdk/update/win98mexp/en-us/3790.0/FULL/PSDK-FULL.12.cab
File sized 24343136, dated 02.04.03 02:28. Database reference psdk.
http://download.microsoft.com/download/platformsdk/sdk/update/win98mexp/en-us/3790.0/FULL/PSDK-FULL.13.cab

Mandrake Linux 10, for i586, CD 1
File sized 728651776, dated 04.03.04 14:40. Database reference m101.
http://bo.mirror.garr.it/mirrors/Mandrake/iso/Mandrakelinux-10.0-Community-Download-CD1.i586.iso

Mandrake Linux 10, for i586, CD 2
File sized 728797184, dated 04.03.04 14:41. Database reference m102.
http://bo.mirror.garr.it/mirrors/Mandrake/iso/Mandrakelinux-10.0-Community-Download-CD2.i586.iso

Mandrake Linux 10, for i586, CD 3
File sized 728829952, dated 04.03.04 14:42. Database reference m103.
http://bo.mirror.garr.it/mirrors/Mandrake/iso/Mandrakelinux-10.0-Community-Download-CD3.i586.iso

Mandrake Linux 10 Live CD, for i586
File sized 646500352, dated 07.01.04 17:55. Database reference mm.
http://bo.mirror.garr.it/mirrors/Mandrake/iso/MandrakeMove-i586.iso

Mandrake Linux 9.2, for i586, CD 1
File sized 683642880, dated 23.11.03 02:20. Database reference m921.
http://bo.mirror.garr.it/mirrors/Mandrake/iso/Mandrake92-cd1-inst.i586.iso

Mandrake Linux 9.2, for i586, CD 2
File sized 731797504, dated 19.10.03 09:25. Database reference m922.
http://bo.mirror.garr.it/mirrors/Mandrake/iso/Mandrake92-cd2-ext.i586.iso

Mandrake Linux 9.2, for i586, CD 3
File sized 728948736, dated 19.10.03 13:10. Database reference m923.
http://bo.mirror.garr.it/mirrors/Mandrake/iso/Mandrake92-cd3-i18n.i586.iso

Gentoo Linux 2004.0, for x86
File sized 721184768, dated 01.03.04 16:09. Database reference g40x8.
http://ftp.du.se/pub/os/gentoo/releases/x86/2004.0/livecd/universal/install-x86-universal-2004.0.iso

Gentoo Linux 2004.1, for AMD Athlon
File sized 646451200, dated 17.04.04 20:33. Database reference g41at.
http://ftp.du.se/pub/os/gentoo/releases/x86/2004.1/packagecd/athlon-xp/packages-athlon-xp-2004.1.iso

Gentoo Linux 2004.1, for i686
File sized 649799680, dated 17.04.04 21:35. Database reference g41i6.
http://ftp.du.se/pub/os/gentoo/releases/x86/2004.1/packagecd/i686/packages-i686-2004.1.iso

Gentoo Linux 2004.1, for pentium 3
File sized 650061824, dated 17.04.04 22:35. Database reference g41p3.
http://ftp.du.se/pub/os/gentoo/releases/x86/2004.1/packagecd/pentium3/packages-pentium3-2004.1.iso

FreeBSD 5.2.1 Technology point release January 2004, for Alpha, CD 1
File sized 643694592, dated 25.02.04 04:51. Database reference f521al1.
ftp://ftp.no.freebsd.org/pub/FreeBSD/releases/alpha/ISO-IMAGES/5.2.1/5.2.1-RELEASE-alpha-disc1.iso

FreeBSD 5.2.1 Technology point release January 2004, for Alpha, CD 2
File sized 304316416, dated 25.02.04 03:38. Database reference f521al2.
ftp://ftp.no.freebsd.org/pub/FreeBSD/releases/alpha/ISO-IMAGES/5.2.1/5.2.1-RELEASE-alpha-disc2.iso

FreeBSD 5.2.1 Technology point release January 2004, for Sparc 64, CD 1
File sized 603619328, dated 24.02.04 15:54. Database reference f521sp1.
ftp://ftp.no.freebsd.org/pub/FreeBSD/releases/sparc64/ISO-IMAGES/5.2.1/5.2.1-RELEASE-sparc64-disc1.iso

FreeBSD 5.2.1 Technology point release January 2004, for Sparc 64, CD 2
File sized 272662528, dated 24.02.04 14:08. Database reference f521sp2.
ftp://ftp.no.freebsd.org/pub/FreeBSD/releases/sparc64/ISO-IMAGES/5.2.1/5.2.1-RELEASE-sparc64-disc2.iso

Sun Solaris 8 Patches, from 27.04.2003, for Sparc
File sized 127879981, dated 27.04.04 22:53. Database reference ss8rs.
ftp://sunsolve.sun.com/pub/patches/8_Recommended.zip

Sun Solaris 8 Patches, from 27.04.2003, for x86
File sized 59245794, dated 27.04.04 22:53. Database reference ss8rx.
ftp://sunsolve.sun.com/pub/patches/8_x86_Recommended.zip

Sun Solaris 9 Patches, from 27.04.2003, for Sparc
File sized 140228892, dated 27.04.04 22:09. Database reference ss9rs.
ftp://sunsolve.sun.com/pub/patches/9_Recommended.zip

Sun Solaris 9 Patches, from 27.04.2003, for x86
File sized 83056377, dated 27.04.04 22:50. Database reference ss9rx.
ftp://sunsolve.sun.com/pub/patches/9_x86_Recommended.zip

Higly compressible text log files from own systems
File sized 6623343. Database referemce logs.
File sized 9556500. Database referemce logs.
File sized 2214092. Database referemce logs.

Index

Abstraction Layer Error Problem 13
abstraction layer errors 12
archaeology 8
arson investigation 8
Complexity Problem 12
data carving 16
digital gloves 13
Exculpatory evidence 10
hash sets 15
Inculpatory evidence 10
known good files 15
Quantity Problem 12