Automatic rule-extraction for malware detection on mobile devices

Andrii Shalaginov

Masteroppgave
Master i Teknologi - Medieteknikk
30 ECTS
Avdeling for informatikk og medieteknikk
Høgskolen i Gjøvik, 2013

Avdeling for
informatikk og medieteknikk
Høgskolen i Gjøvik
Postboks 191
2802 Gjøvik

Department of Computer Science
and Media Technology
Gjøvik University College
Box 191
N-2802 Gjøvik
Norway

Automatic rule-extraction for malware detection on mobile devices

Andrii Shalaginov

2013/06/02

Abstract

Malware causes damage not only to personal computers, yet also to contemporary mobile devices. With growing performance and storage capabilities users of mobile devices tend to store more sensitive information than before. Additionally, mobile platforms allow to use charged telecom services via installed software applications for extending the functionality of devices. Beside certified application-distribution services, users can download applications from uncertified developers. The amount of applications have been increasing exponentially each year and part of them are distributed by third-party markets. Taking all these aspects into account, mobile devices have become attractive targets for attackers and their malicious software.

Mobile platforms possess restricted access to information and execution of applications. In order to be able to execute some functionality, applications require a user to provide a set of permissions. Another protection mechanism is commercial Anti-Virus (AV) software that uses so-called signatures. These signatures define indicators used for malicious applications recognition. The detection process of such software can be as simple as file names comparison or as complex as checking system artifacts. Sometimes signatures can be composed only as a result of advanced malware reverse engineering. Despite the fact of the existing protection solutions, there is still a challenge to detect malware automatically in dynamic environment. This is because the malware detection process involves evaluation of different factors, which accompany malware execution.

This study focuses on deriving fuzzy rules for malware detection automatically. Challenges of malware detection are many-fold and therefore we will focus on mobile devices in this study. We introduce precise artifacts that mobile malware leaves during execution. In this study a virtualized environment is involved in studying dynamic malware behavior. In addition, analysis of static malware attributes is performed. The goal is not only to derive malware detection rules automatically, yet also empower them with linguistic meaning that is understandable by human. The thesis will establish a method in, which combination of Artificial Neural Networks (ANN) and Fuzzy Logic (FL) is utilized for rules extraction. In result, such rules are human-explainable, which allows forensics analyst to use them in a court of law. Finally, the thesis presented here provides justification of how derived rules can be applied in an automated analysis of large amount of mobile malware.

Acknowledgements

This thesis concludes 2 years of study in the Master of Information Security study program at the Gjøvik University College.

I would like to express my deep gratitude to my supervisor, Prof. Katrin Franke, for her invaluable advices, supervision, shared experience and inspiration in machine learning and pattern recognition.

Thank you also to my colleagues, particularly Ruslan Puzyriov, Knut Borg and Heng Zhang for sharing ideas, valuable feedback and fruitful discussions. I would like to extend my thanks to all members of staff at Gjøvik University College for creating an excellent environment for scientific work and for conducting such thesis research.

Last but not least, I would like to thank my mother and my fiancé for believing in me; for their understanding and motivation during the work. Without their unwavering support, it would have been difficult to accomplish this work.

Andrii Shalaginov, 2013/06/02

Contents

Glossary

Android Package File	Compressed installation package used for Android applications distribution and installation
Black Box Testing	Testing and exploration of functionality of an application without knowledge about an entire structure
Emulator	A software/hardware solution that is able to imitate computer or device
Features	Single measurement of some parameter, also called attribute
Features Extraction	A process of deriving features from the raw measurable data or characteristics within a mobile device
Fuzzy Logic	A variant of the classical logic, which uses truth degree for each linguistic variable rather than simple binary true or false statements
Fuzzy Rule	A conditional IF-THEN statements that are composed from linguistic variables
Linguistic Terms	The discrete linguistic variable in fuzzy theory that can have truth degree (instead of classical true or false)
Linguistic Rules	In this study means fuzzy rules used for malware detection
Linguistic Variables	The variables in fuzzy logic theory, which can take linguistic terms as values. In this work security metrics are considered as linguistic variables
Membership Function	In fuzzy logic represents degree of truth that a given value belongs to some fuzzy term
Neuro-Fuzzy	Fuzzy logic theory that uses artificial neural network to derive the estimate and derive the rules
Rules Construction	A process of composing rules from the security metrics consist of two stages: all possible rules extraction and selection of the most relevant rules
Rules Mining	In this work rules mining means construction of essential fuzzy rules for malware detection
Security Metrics	A complex characteristic of some security-related domain that consists of several raw features

Security Metrics Construction	A process of composing selected raw features into a security metric according to a domain
Smartphones	A mobile device that usually has a mobile phone functionality with an installed operation system
Virtualization	A software/hardware solution for deploying virtual analogue of a software/hardware

Acronyms

ANN	Artificial Neural Networks
API	Application Programming Interface
APK	Android Package File
ARM	Advanced RISC Machine
AV	Anti-Virus
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DEX	Dalvik Executable Format
EM	Expectation Maximization
FL	Fuzzy Logic
GPU	Graphics Processing Unit
JNI	Java Native Interface
HDD	Hard Disk Drive
MF	Membership Function
ML	Machine Learning
NF	Neuro-Fuzzy
OS	Operating System
PC	Personal Computers
PR	Pattern Recognition
RAM	Random-Access Memory
SDK	Software Development Kit
SVM	Support Vector Machine
VM	Virtual Machine

List of Figures

List of Tables

Listings

1 Introduction

This Chapter provides an overview of the defined research area and questions to be answered. Additionally, methodology and contribution of this thesis are summarized.

1.1 Keywords

Mobile malware, automated malware detection & analysis, machine learning, rules extraction, neuro-fuzzy approach.

1.2 Covered Topic

In the recent decade a popularity of mobile devices like smartphones have increased considerably due to functional and computational abilities. They are much more portable and consume less energy in comparison with general Personal Computers. This fact extends their usage in business and home related activities such as surfing the Internet, purchasing goods, interacting with Internet banking, etc. During early 2000th, malicious software was associated mainly with Personal Computers (PC). Such software targets the computer systems, network infrastructure, sensitive private information as well as taking control over the computer system operation in general.

Currently a vast amount of malware has been developed on mobile platforms, which makes the user's sensitive information vulnerable to malicious actions. As malicious programs develop the signature-based Anti-Virus (AV) software develops as well. Signature-based detection utilizes a set of signatures or rules. The signatures can represent not only a MD5 hash sum[1] of the files, yet also text regular expressions, source code patterns, specific Application Programming Interface (API) calls, names, etc. Developers of most signature-based Anti-Virus (AV) programs aim to filter out all known malware. However, such rules are very specific and maintainable mostly by malware analysts or reverse engineers [1]. Moreover, the Anti-Virus (AV) software is not able to deal with dynamically changeable and proactive environment [2]. This is because detection signatures are composed manually and therefore sets of signatures are often outdated.

This thesis aims to develop a malware-detection method based on deriving of Neuro-Fuzzy (NF) rules using automated rule mining. Neuro-Fuzzy (NF) approach provides human understandable and explainable rules, which do not require additional post processing. Furthermore, in a court of law a judge and a jury may understand the reasoning behind the extracted rules, which is very important under computational forensics investigation of digital evidences.

1.3 Research area

In general, computers allow user to install additional software on it. These applications can be either benign or malicious. Benign applications are considered to be useful and serve as it is required. In contrast, malicious software or malware is intended to execute harmful operations

[1]MD5 is an obsolete cryptographic has sum algorithm, which is suitable for purposes of content comparison due to low computational complexity than modern SHA-2 or SHA-3

either against the computers or users sensitive information. In a historical perspective, the first malware sample appeared a few decades ago [3]. Since performance of the computers grew, they got cheaper and intentions to create malicious software increased. Starting with a simple password stealer (keylogger) they now are able to perform large-scale attacks against businesses and public people. This thesis includes analysis of what kind of information can be stolen from user on contemporary mobile Android devices.

The malware detection on mobile devices is in demand and recently a developing field within computer security. Mobile devices are gradually replacing PC. Modern mobile devices capable of storing vast amount of data. Among that data there are sensitive personal information such as passwords, mail and bank account details. This thesis shows that fraudulent schemes have migrated from PC to mobile devices already. Despite complex protection mechanisms in mobile Operating System (OS), user undertakes control of applications privileges. That makes a whole mobile device vulnerable to attacks in case of carelessness or mistake. Additionally, software markets are flooded by fake applications with malicious payloads [4, 5]. Moreover, common user can not distinguish malware from benign applications due to lack of knowledge [6]. The author focuses on human understanding of malware detection in this thesis.

The important step in malware analysis is a collection of specific attributes by which a malware can be characterized. Mobile malware reverse engineering provides comprehensive view on malware functionality instead of only deep analysis [7]. However, this process is mainly manual and based purely on knowledge of the analyst in contrast to Anti-Virus (AV) software. Anti-Virus (AV) software uses signatures databases, which are hardly interpretable by average person or by court of law [8]. Furthermore, signatures contain single measurements (or features) of parameters in time, which are discrete and targeted only on specific malware. This means that there should be used security metrics instead of pure discrete features. The study [9] has shown that security metrics are more suitable for human representation and abstraction of features. This is because features are mainly collected through statistical analysis while metrics are mapped by analyst. Influence of such metrics on malware detection and forensics soundness is described later in this thesis.

Machine Learning (ML) is a domain of computational intelligence, which allows to build adaptive and expert systems. Such systems can improve their performance and learn new as well as using already collected knowledge. Support Vector Machine (SVM) is one of the best classification methods, which provides high performance on different tasks [10]. Despite the fact that there were developed many powerful algorithms, we concentrate our work on the Neuro-Fuzzy (NF) approach. The biggest advantage of Neuro-Fuzzy (NF) is the possibility of a non-linear[2] statistical model construction for classification and regression purposes [4]. Additionally, Fuzzy Logic (FL) is understandable by human brain. Furthermore, Neuro-Fuzzy (NF) has considerable significance in dealing with large data analysis. The goal is also to utilize Neuro-Fuzzy (NF) as a tool for automated analysis.

There exist several approaches such as signature-based misuse and behavioral-based anomaly

[2]Means that the model uses nonlinear combination of the parameters and does not linearly and directly dependent on the input data. In other words we can state that the first derivative of non-linear model function is dependent on one or several parameters

detection [11, 12]. Because of various obfuscation, fragmentation and masking techniques, signature-based concept becomes less reliable than experimental behavioral-based according to recent studies. Machine Learning (ML) provides more flexibility in building detection rules than manual analysis. This is because Machine Learning (ML) utilizes statistics and learning instead of purely defining parameters for each malware [13]. Neuro-Fuzzy (NF) approach can be applied for extraction of detection rules based on malicious and benign application's statistics [14]. This thesis aims to extract human-explainable rules for malware detection.

1.4 Research questions

This research aims on construction of detection rules utilizing a Neuro-Fuzzy (NF) approach. Previous known work on Neuro-Fuzzy (NF) application in malware detection is designed only for PC [15]. This work is not applicable for usage on mobile devices. It means that our work makes innovative contribution to forensically sound malware detection on mobile devices. The author's previous work on automated mobile applications testing is used as to support the study [16]. Security metrics based on application's features (parameters) are introduced here. Such metrics enable both automated malware-detection rules and human comprehension. This thesis seeks to answer the following research questions:

1. What kind of security metrics could be applied for malware detection and what is the detection reliability of such metrics?

2. How do user profiles (various sensitive and private information stored on the device) affect malware behavior and what kind of data are stored/transmitted by malware?

3. Are the results of static and dynamic testing of mobile applications reliable for automated malware detection?

4. Is it possible to automatically extract corresponding advanced fuzzy rules and provide a fair detection rate?

1.5 Methodology to be used

In order to solve the research problems we organize the research as following. Thesis includes both justification of proposed theorems and proof-of-concept demonstration. For theoretical part we elaborate on foundation of theory for security metrics and motivate reasons for using it. For practical part the neural-network theory is combined with fuzzy-rule extraction that leads to a Neuro-Fuzzy (NF) approach. This is necessary for achieving reliable classification based on gained security metrics.

The practical part is conducted with an experimental setup that allows the author to test and to classify given software samples based on extracted metrics and fuzzy rules. As a mobile platform for our experiments the Android OS was selected. The rationale behind selecting the Android is that Android security model allows the user to install third party software. The user is also entitled to grand execution privileges to applications. Subsequently it provides more threats to user privacy.

The dataset used for experiment on mobile applications is partially based on the previously collected data during work on a testing laboratory [16]. Essentially, that dataset was supplemented by approximately 50% of benign applications. The practical part was performed in several tests. First of all malicious and benign applications are labeled and gathered into a single set. In addition the author completed it by different versions of applications targeted on different Android Application Programming Interface (API). Secondly, expert-defined features are extracted from each application. Thirdly, security metrics are composed from extracted features. As a result, train and test metrics sets are utilized in practical part. All datasets are provided together with thesis report (for details see the Appendix A).

During conducted experiments, quantitative and qualitative performance measures are collected and presented. Gathered and generated datasets are supplied on DVD along with the report. In summary, one can outline used methodology:

1. Collect malicious and benign applications.
2. Preprocess the applications dataset and extract features.
3. Create specific user profile (store sensitive information).
4. Implement and perform static and dynamic tests over dataset.
5. Extract features (parameters) for each application.
6. Construct security metrics using expert knowledge.
7. Implement fuzzy-neuro model.
8. Extract fuzzy rules from trained Artificial Neural Networks (ANN).
9. Estimate classification accuracy using extracted rules.

1.6 Justification, Motivation and Benefits

The amount of malware on mobile devices has been growing significantly since it first was discovered in 2010th. Few years ago privacy threats may have been considered minor due to storing limitation and complexity of the mobile programs. Yet now they are having impact on more sensitive data as well as converting a mobile phone to an attacker's remote terminal.

To the authors knowledge there exists no reliable signature-based solutions for automated testing and malware detection for mobile platforms. Researches have targeted on either automated testing or non-interpretable and non-automated malware detection.

In the authors work signature-based malware detection takes into consideration fuzzy patterns by which malicious and benign applications can be distinguished. Applying signature-based solutions one can bring fast and reliable detection of the malware sample in mobile system. Subsequently, fuzzy-rule utilization allows not only to detect malware, yet also to give humans an understandable form of signatures. The main difference of such approach from behavioral-based detection is the independence from user behavioral and centralized databases [17, 18]. Moreover, in contrast with behavioral-based it can manage without any user interaction. Furthermore, NF approach can be trained from data that applications generate during automated execution.

The author generated single user profile with specific sensitive information. As it is shown, the effect of this profile on malware detection can be neglected. This is because amount of applications that sends user's sensitive data is small in comparison to total amount on market.

After the project is accomplished and researched questions are solved, possible automated and human-understandable malware detection will be established. The system will be adaptable to any new applications properties and metrics. The research can provide significant aid in mobile malware detection and defense for large organizations, public and private software markets. Finally the proposed procedure will encourage future experimental behavioral-based methods using security metrics.

1.7 Limitations

This thesis is targeted on examining mobile application data, which are available under static and dynamic tests. This means that the author have a limited amount of time on probing and testing each particular application manually. To counter this problem automated routine is implemented in order to reduce the amount of necessary time. All possible features are extracted during the test cycle. However there is a probability to miss some of the related details due to automated testing procedure.

Due to privacy limitations it was impossible to perform large-scale data collection for user's profile study. In order to solve this problem we were working on producing single user profile based common data that users tend to store/utilize on mobile devices. It is reasonable to use only one user profile with stored information in the most popular applications. These applications are: phone book, calendar, browser, mail, messages and calls. We do not consider additional user profiles because their variance does not affect the malware detection process significantly. If there is no sensitive user-related information in the profile they does not influence the detection at all.

To our awareness there has been work on hardening of mobile applications and security solutions [19]. As it was examined in this work, application platforms and mobile markets have own protection mechanisms. In contrast to this, we emphasize on forensic soundness of automated malware detection using Machine Learning (ML), i.e. Neuro-Fuzzy (NF). Besides available hardening it is necessary to create additional means of protection. We focus our examination on Android OS (Operating System), because of exponential growing amount of malware for this platform. Furthermore this platform allows to distribute third-party application bypassing the official market Google Play [20]. Unlike this fact, iOS mobile OS represents itself very strict and conservative system in application installation domain [21]. For general user it is possible to install application on iOS file system just from App Store [22]. That is why only Android OS represents interest for us from perspective of this research.

1.8 Thesis Contribution

This thesis seeks to provide better justification behind human understandable malware detection rules. Using these rules, user can transfer knowledge gained about malware to another users. It will increase total security awareness of common mobile devices target customer auditory. Also it can be mentioned that such detection method is capable of performing large-scale investigation of mobile markets. One can say that this is a promising area of research in information security. Based on that, the contribution of master thesis can be outlined as following.

Initially, we focus on gaining new knowledge in a defined problem area in order to build reliable classification based on human understandable rules. For this we extract parameters from an application during its static and dynamic testing. These parameters are also called features. Using features, security metrics age going to be created with help of analyst knowledge. The metrics are more general and interpretable by users than specific and narrow technical features. Then proposed detection method using NF approach and based on extracted metrics is utilized. As a result, we obtain fuzzy rules that are both human explainable and reliable for malware detection. Moreover, this is applicable in automated applications testing in big datasets.

As a significant achievement of the thesis, one can highlight practical work on datasets and experimental setup. First of all, sizeable datasets and user profile are created, which are important for further research in this area. Among datasets there are collection of various mobile applications, extracted features and constructed metrics for each of the application. Additionally, examination of user profile and its influence on malware behavior is analyzed. It leads to developing of emulated sensitive information in every aspect of mobile platform usage. Finally, prototype is constructed for proving the concept and understanding of weaknesses / strengths of theory. To authors knowledge neither data set nor suitable automated environment were proposed before.

One can also mention implications and impact of performed theoretical studies and practical implementation. It was found that there exists a huge amount of possibilities to extract features. Some of the features are irrelevant for malware detection while others are very important. Performed research proved under theoretical foundation that human understandable rules can be extracted from automated analysis of mobile applications. From experimental point of view, the training time of the proposed method on metrics is considerably less that on raw features. Accuracy of detection based on features does not differ significantly. This thesis found that usage of virtualization and Graphics Processing Unit (GPU) acceleration provides significant speed-up of the test and execution process. In most cases emulator took less time to boot than physical devices with the same configuration. Additionally, it was noticed that many applications have concealed dependency on specific device hardware. Finally, extracted rules based on features are not so difficult to perceive as based on features.

1.9 Thesis structure

The thesis is organized to provide better understanding of the problem. First, theoretical background and proposed method are described. Then practical aspects are given. The work has following structure:

- Chapter 2 provides overview of the research area. Initially, malware with focus on mobile devices and related details are described. Then follow literature about security applications parameters and security metrics. After the application of ML approach in malware detection is presented. Finally, details are given that focus on fuzzy-rule extraction using ANN .

- Chapter 3 seeks theoretical answers on defined research questions. Firstly, extraction process of security metrics from available artifacts on mobile devices are presented. Secondly, justification behind utilization of ML and Pattern Recognition (PR) in malware detection is outlined. This chapter finalizes with proposed innovative method of fuzzy-rule extraction for malware analysis based on studied scientific literature in the Chapter 2.

- Chapter 4 consists of all practical results. Initially, construction of datasets process is described. Then security metrics and extracted detection rules are analyzed. Moreover, reliability of static ans dynamic tests and automated testing are discussed.
- Chapter 5 includes discussions of overall findings and implementation architecture concerns. In the end, important details and findings in implementation architecture are mentioned.
- Chapter 6 provides conclusions, theoretical and practical implications and suggestions for future work.

2 State of the art

The important step of each research is to get overview of existing methods and developed approaches for problem area. This chapter provides overview of the relevant scientific literature in order to lay foundation and support the contributions of the thesis.

2.1 Malware with focus on mobile devices

Mobile devices can be considered as an evolution of stationary PC. They inherited main components and building principles in both hardware and software perspectives. Despite the fact that mobile devices have own security-protection mechanisms the classical threats landscape can be projected substantially from PC to mobile devices. At the moment the main difference between these two platforms is access to paid services via mobile OS. This aspect makes portable platforms vulnerable to a new horizon of misuse attacks.

Closer look on architecture of mobile devices gives a clear understanding that it is hardly possible to launch unnoticeable background process because of resources limitations on mobile platform. In this case it is more likely for attackers to use fake popular applications or write own "valuable" software and mask malicious payload in such a way [4]. It means that malware activity will not be noticeable while executing benign functionality.

After extensive study, one can map normal PC environment with possible attacks to a *mobile devices environment* with corresponding components [23]:

1. Client $\rightarrow$ mobile devices
2. Network $\rightarrow$ wireless communication like GSM, WiFi, etc
3. Server $\rightarrow$ usually consumer web-based services

Moreover, one can highlight main *types of malicious actions* targeted against mobile platforms [24, 25, 4]:

1. Scam - misleading offers for premium rates services
2. Phishing - type of fraud aimed on gathering sensitive information such as passwords, bank account numbers, etc [26, 27, 28]
3. Spam - sending messages to persons in a contact list without user's authorization
4. SMS[1] Trojans - subscribe a mobile devices to send SMS to a premium rates services
5. Information stealer - steals specific sensitive data in order to sent it to attackers

[1]Short Messaging Service

6. Illegal positioning - sends GPS[2] data without user permission
7. Pop-up advertising (adware) - publish advertisements consistently on the screen
8. Botnets - hidden functionality that allows to connect to multiple bot-clients and execute massive spam or denial of service attacks [29]

Major amount of security attacks against mobile platforms are usually related to problems in access control that is granted by user to an application. Even though security-permissions model in mobile OS is well-thought-out and complex, users still has dangerous granting permissions role [30]. It means that a user can negligently provide such permissions to an application in a rush. In addition, there exist also a vulnerability that allows to send SMS even without requesting actual 'SEND_SMS' permission [31, 32]. Also taking into the consideration the fact that third-party applications can be installed on Android OS, one can conclude that mobile platforms are exposed to attacks.

2.1.1 Mobile OS and Markets protection

This subsection provides overview of the recent technologies and approaches in malware protection on mobile devices. There are mentioned the drawbacks and limitations of existing solutions.

Recent software and hardware architecture in mobile platforms provide standard security mechanisms in order to protect device and user's data from unauthorized actions. According to recent study of Android and iOS platforms security [33], they have following basic mechanisms:

- Device access control,
- Sensitive data reading/altering restriction by processes,
- Each application is provided by specific permissions (granted by user) [25],
- Limited interaction between hardware and software layers,
- Protection against various types of web-attacks.

Over the last year the total amount of applications on official Android Market have been achieved number in eleven billions and will be growing exponentially in nearest future [34]. As one can see from the statistics [35], Android and iOS occupies over 80% of the mobile platforms marketplace. Android shows more bigger grows of its share, taking up more than 50% of Smartphone OS Market in 2012.

The main difference between these two popular platforms is that Android OS allows to use third-party markets for applications distribution, while iOS has single App Store. One can also mention that Android is an open project, so manufacturers of devices can change the UI, which may affect security issues. Furthermore, Google Play does not provide sufficient security testing of all applications available on Market, just basic scan [36]. Client-side protection mechanisms for Android were mentioned before as well in recent studies [25, 33, 36, 37]. Of course, it is impossible to have trusted security level for applications from Black markets. Both platforms

[2]Global Positioning System

have some problems with upgrading to a new version of OS because of hardware or manufacturer limitations.

Now we want to concentrate more on market-side protection that can be found on official markets. Below we give some details of available protection routines:

Android Google offers to its customers a centralized market called Google Play where authorized users can submit their applications. All applications on the market has signature and pass through basic validation. If an application is found malicious after submission then it can be blocked. Yet this very depends on user reviews and comments [19]. In 2012 Google announced Bouncer systems that filters malware on Google Play Market [38]. Even taking into consideration Google statement about decreasing total number of installed malware there are still no details available about this system.

iOS Apple App Store represents well protected and safe market of mobile applications [37]. All entries are precisely checked before posting and users can be assured that applications are safe to use and does not contain malicious payloads or viruses [39]. Additionally, Apple does not reveal API, which reduces number of known vulnerabilities on this platform.

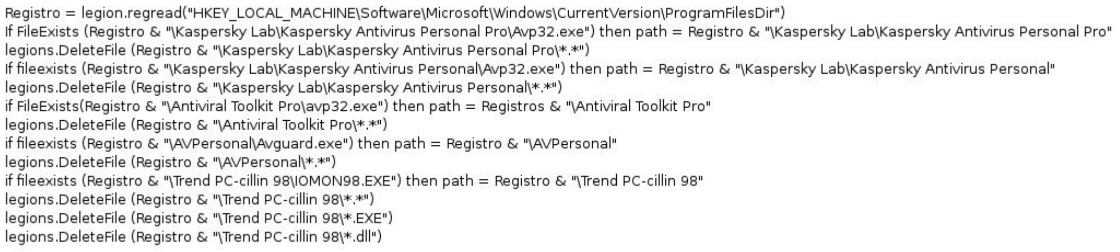
Windows Mobile Windows Phone Apps store [40] offers only around hundred of applications in comparison with to million on Google Play. Only after validation and subscription procedure, a developer can submit applications on this market. The submission process includes validation and certification of the applications. According to Microsoft MSDN [41], the security policies and certificates inside the devices are checked before launching the applications. Based on the input and configuration, the OS provides normal or privileged execution. However, we can say that popularity of this platform is falling down dramatically, which caused decrease in mobile marketshare to a few percents in total.

2.1.2 Commercial AV software

The problem that occurs in mobile devices protection from malicious software is a lack of a strong and a comprehensive internal solution. Contemporary AV programs for PC show detection rates over 90% because of well-studied drawbacks and vulnerabilities over the past few decades [42]. In case of mobile platforms, such programs are not so complex and mostly can not show confident detection rate. As we see from the testing of various mobile AV solutions, only around 25% of them can produce more than 90% detection rate [43]. It can be explained by pure study of all possible vulnerabilities, bugs and coding errors due to time constraints. That is why, examination of recent mobile malicious software needs manual processing by corresponding specialists and takes much time.

Drawbacks of existing signature-based solutions

With growing amount of new variants of viruses and zero-day attacks, classical signature-based AV software becomes less efficient for malware detection [1, 2, 44]. This is caused by the fact that such kind of AV software relies on statical signature sets, which are filled and updated by developer company. This is one problem that escalates difficulties in signatures composing, due to lackness of special knowledges in the field. Prior to signatures composing, it is necessary to obtain and perform very deep reverse engineering of the malware sample. The second problem, which appears after signatures composing is its complexity. Common information technologies aware user can not imagine the whole picture of malware-detection process. As an example we can consider mass-mailer written in Visual Basic. It has a functionality to disable popular AV solution as it is presented in the Figure 1.



```
Registro = legion.regread("HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\ProgramFilesDir")
If FileExists (Registro & "\Kaspersky Lab\Kaspersky Antivirus Personal Pro\Avp32.exe") then path = Registro & "\Kaspersky Lab\Kaspersky Antivirus Personal Pro"
legions.DeleteFile (Registro & "\Kaspersky Lab\Kaspersky Antivirus Personal Pro\*.*")
If fileexists (Registro & "\Kaspersky Lab\Kaspersky Antivirus Personal\Avp32.exe") then path = Registro & "\Kaspersky Lab\Kaspersky Antivirus Personal"
legions.DeleteFile (Registro & "\Kaspersky Lab\Kaspersky Antivirus Personal\*.*")
if FileExists(Registro & "\Antiviral Toolkit Pro\avp32.exe") then path = Registros & "\Antiviral Toolkit Pro"
legions.DeleteFile (Registro & "\Antiviral Toolkit Pro\*.*")
if fileexists (Registro & "\AVPersonal\Avguard.exe") then path = Registro & "\AVPersonal"
legions.DeleteFile (Registro & "\AVPersonal\*.*")
if fileexists (Registro & "\Trend PC-cillin 98\IOMON98.EXE") then path = Registro & "\Trend PC-cillin 98"
legions.DeleteFile (Registro & "\Trend PC-cillin 98\*.*")
legions.DeleteFile (Registro & "\Trend PC-cillin 98\*.EXE")
legions.DeleteFile (Registro & "\Trend PC-cillin 98\*.dll")
```

Figure 1: Sample of AV killer code in Visual Basic

In order to detect this malicious software, the ClamAV AV uses logical signatures [45] of the following format:

$Worm.Godog; Target : 0; ((0|1|2|3)\&(4)); (0); (1); (2); (3); (4)$

Despite the fact that such signature can detect malware it is hardly understandable and uninterpretable without special knowledge of the field. Moreover, in court of law it is impossible to use such rules without additional description and scientific justifications.

2.2 Security Metrics

Considering inefficiency of using pure features in signature-based malware detection, our decision is to concentrate on security metrics. Risen security-threat landscape indicates that there should be applied more advanced transformation of measurable features available for application. Taking this fact into consideration, we concluded that security metrics may help to solve problem and improve malware-detection process. Security metrics were considerably studied last decade. They can be treated as human interpretations of raw measurable parameters, which are turned into valuable information [9, 46, 47].

To our knowledge, there were no scientific and practical work on security-metrics construction for mobile devices. Furthermore, such concept have not appeared, yet in the area of mobile malware detection.

Influence of user profile

Recent mobile devices allow to store vast amount of information in a comparatively very limited physical size. It makes them irreplaceable in ratio of portability / volume of stored private information. Undoubtedly, this fact makes them vulnerable to various privacy threats such that malware or hidden information stealing.

Examination of stored information is vital for understanding how user's profile (this information) affects malware behavior. The main obstacle for transition from classical signature-based solutions to a novel anomaly-detection mechanisms lies in discovery dependency between user's profile and malware success. It may happen that malware triggers only under special circumstances and availability of certain information.

Based on general user's normal day web activity examination, there can be dedicated few major domains [24]:

- Internet
- Social network [48]
- Content services
- Search engines
- Messaging
- Calling services

According to study [24], social networking and media resources take more attention by general user. This is caused by the fact that usually such resources propose own ready-to-use applications that considerably simplifies navigation. Despite usability, it makes possibility to fake such application and build-in malicious payload. Furthermore, due to the same appearance, general user has no ability to distinguish between genuine and fake applications [29]. However, extracted security metrics may reveal significant differences interpretable by common user.

2.3 Machine learning and pattern recognition in malware detection

Classical signature-based solution are no more efficient for successful malware detection because of encryption, polymorphism and other obfuscation methods. Recently problem of malware detection has been migrating to a machine learning and a PR domains. It allows to perform not only general statical checking and comparison routine, yet also to analyze a program in the dynamic environment. The key feature of Machine Learning (ML) is that it allows to build educable and adjustable system based on collected data. Furthermore, it is possible to achieve sufficient level of detection of malware samples even without human interaction.

For successful detection, there should be present two parts: learning algorithm and training labeled dataset with malicious and benign applications [10]. The example of general ML approach is shown in the Figure 2.

During training phase, a statistical model is going to be learned from input training dataset. After learning process has been performed, a statistical model can be used for classification or prediction of a new unlabeled input pattern.

Overview of existing approaches. Pros & Cons

There was performed a significant work on ML approaches adaptation for malware detection on mobile devices [49, 50, 51, 52]. According to taxonomy in the ML book, one distinguish following general ML techniques [10]:

- *Supervised* - learning of statistical model from labeled dataset or other known information for supervision.
- *Unsupervised* - learning from unlabeled dataset, where the task of ML is to reveal hidden or unknown patterns without any additional information.

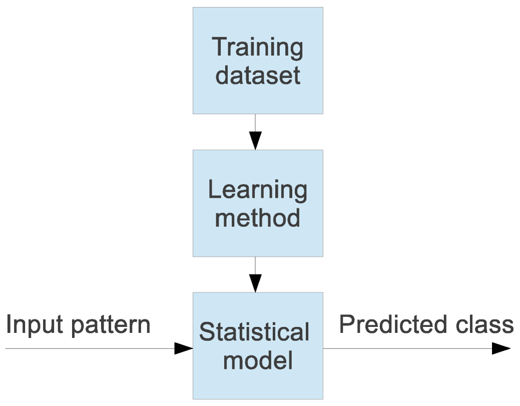


Figure 2: General ML approach

However, in real case it is more suitable to use mixed approaches. Such models combines supervised and unsupervised techniques for getting more reliable results. Despite the advantage of using mixed approaches, there are some difficulties with ML in malware detection. Initially, features and attributes have to be extracted from applications. Then extracted data have to be preprocessed for further using. Finally, qualitative and quantitative criteria of the end of learning process need to be defined.

After extensive study of existing ML approaches, one can highlight disadvantages and advantages of the following procedures, which are more appropriate for the task [10, 49, 50, 52]:

- **SVM** - binary supervised classification procedure.

  Pros: Robustness and Generalization. Understandable classification process.

  Cons: Performs well on linearly-separable[3] classes data, otherwise requires complex non-linear transformation by means of kernels.

- **ANN** - non-linear simulation of learning and decision making activity in human brain. Particularly Self-organizing map (SOM) procedure is a variant of ANN.

  Pros: Flexible in case of non-linear classes, supports high degree of complexity.

  Cons: Complex and incomprehensible, which is hard to interpret.

- **K-means** - unsupervised learning procedure.

  Pros: Fast partitional clustering

  Cons: Significantly depends on initial centroids and as results may provide quite different results.

- **K-Nearest Neighbor** - supervise binary classification procedure. Utilizes lazy learning for decision making.

[3]Basically means that the hyperplane of format $y = a \cdot x + b$ can be used for classes separation. In simple words, the line separates instances of the first and the second classes, if depicted. From the other side, derivative of such hyperplane function's is not be dependent on input parameters

Pros: Fast performance, understandable decision making process.

Cons: Inappropriate in case of outliers and noised components in training data set.

- **Naive Bayes Classifier** - simple probabilistic classifier for multi-class problems.

  Pros: Helpful in case of missing features in data set, relatively fast.

  Cons: Uses initial assumption that features are strongly independent.

- **Fuzzy-Neuro** - statistical model that utilized both ANN and fuzzy logic and provides human interpretable reasoning of the defined problem.

  Pros: Allows to model non-linearity of high degree in classification problems. Extracts human understandable linguistic fuzzy rules from complex ANN.

  Cons: Learning and optimization of extracted rules for malware detection can be time-consuming in case of big data set.

One can conclude that NF is the most appropriate procedure for malware detection in case if it should be interpreted by human. Furthermore, one can extract linguistic rules as a result of processing collected data set by this procedure. This approach has already presented it successful perspective in previous works related to rules extraction [53].

2.4 Malware detection & analysis using neuro-fuzzy

Mentioned in previous section ML approaches are well-studied and have powerful theoretical basis behind them. However, the main drawback of most approaches is almost uninterpretable final statistical model. Despite successful usage in malware detection, it is hard to extract human understandable patterns or rules. The only exception from this provide NF, which output is set of fuzzy rules based on the model as it is shown in the Figure 3. Furthermore, NF inference systems are adoptive and can be learned in live environment without complete re-training.

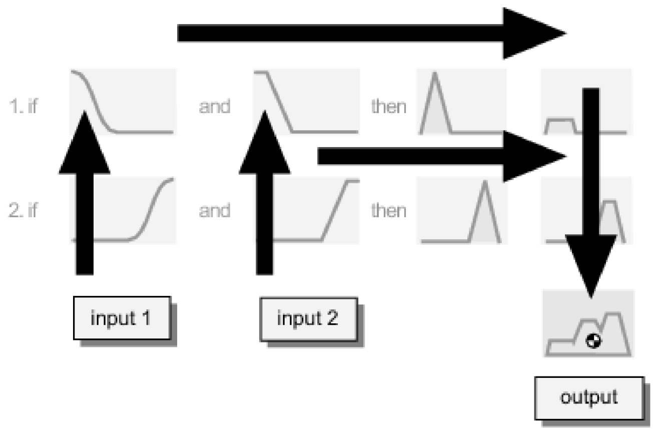


Figure 3: Model of fuzzy rules construction in neuro-fuzzy [54]

After extensive study of literature, one can be concluded that NF inference systems can be applied in information security and computational forensics. The study [14] presents that adaptive

NF resolves problem of malicious executable *.exe* files detection. Authors outlined good performance and accuracy in malware detection. Second study [13] emphasizes on tolerance to noised and mistaken data and adaptive capabilities.

In the research [15] it was proposed a three fuzzy sets (benign, suspicious and malicious) detection method using neuro-fuzzy. Despite the fact that method has high detection rate, rules are understandable to an expert only in a very narrow domain. Multiple network-related features are used in NF anomaly-detection system in [55] . Such system gains high detection rate on more than 40 extracted feature. However, features used in rules like "srv-diff-host-rate" are not understandable for unaware user. This fact makes rules applicable in a narrow specialists environment.

Taking into consideration advantages of NF adaptive systems, our proposal is to use security metrics [56]. Initially, extracted rules based on these metrics are understandable for both common and expert users. Then, gained knowledge can help to detect, analyze and prevent malware. Finally, it will increase total awareness and understanding about malicious software among mobile device users.

NF has a significant application perspective in computational forensics and information security, as it is comprehensively studied in the researches [57, 58]. Also these researches arise the question whether FL can be applied in the dynamically changing data. In a comparative study [59] it was proved the advantage of using fuzzy logic with human reasoning and powerful ANN for novel attacks detection. The master thesis is going to prove that it is possible to apply it in live environment without loss of forensic soundness. In the studied literature, NF has not been applied for human interpretable malware detection on mobile devices.

3 Methodology

This chapter provides description of the methodology that was used for the research. We give overview of scientific methods and approaches that were used in order to retrieve answers on defined research questions. First of all, security metrics construction process is described. Then, overview of ML in perspective of the malware detection is given. Finally, we concentrate on fuzzy-rule extraction for malware detection.

3.1 Theoretical surveying of defined problem area

We are planning to utilize ML and PR as a concept for designing of self-learning systems and computational intelligence. This aspect is crucial in malware detection and can provide sufficient confidence without human interaction.

Existing malware classification methods using Linear-Discriminant Analysis or Support-Vector Machine can provide reliable classification. They are appropriate for usage in linearly-separable metrics space for both benign and malicious applications. It means that these methods can have errors on training set in case of not linearly separable data. Therefore, SVM offer additional transformation by means of kernel or multiple kernel solutions. This is a complex task that requires extensive examination of the input data's statistical characteristics. One can also use ANN as classification algorithm that utilize non-linear transformation of the data intrinsically. Finally, the strategy of Machine Learning (ML) usage for malware classification can be described as following:

1. *Training phase*
    - 1.1. Perform automated data gathering from static and dynamic phases of each application testing
    - 1.2. Extract features from gathered testing data
    - 1.3. Evaluate each feature merit in the particular security metric
    - 1.4. Build and estimate each metric's value based on the features
    - 1.5. Model ANN and estimate it parameters using fuzzy approach. The each rule weight is adjusted by means of one-dimensional optimization procedures such that gradient descent or golden section search
    - 1.6. Statistical relationship is binding between input and output for each metric by means of fuzzy patches using Gaussian Membership Function (MF)
    - 1.7. Extract all possible combinations of FL rules that can characterize both classes
    - 1.8. Tune obtained rules by evaluating of ANN neurons weights
2. *Testing phase*

2.1. Construct classification model using derived fuzzy rules

2.2. Estimate membership of an unclassified application to the set of rules in classification model

2.3. Make a final decision with respect to gained knowledge

3.2 Retrieving of security metrics from applications testing process

This Section is devoted to security-metrics construction. These metrics are human understandable indicators of security assets, by which any application can be treated from Information Security perspective.

Our strategy for solving defined problem is to apply ML approach based on NF for fuzzy rules extraction. This rules further will be used as a basis for classification model. Also they include not only encoded malicious software specific details, yet also are understandable for general human brain in quantitative domain.

Malicious and benign applications are going to be gathered in order to get training dataset with equal distribution of samples from both classes.

3.2.1 User profiles creation

Android emulator [30] provides a powerful tool to create and use customizable runtime images. When Android emulator is launching, it uses few important images with following information: kernel, system, SD[1] card, user and cache data [60]. Obviously, we concentrate our attention on user data image. Here we should distinguish two types of images with user information [30]:

- **userdata.img** - copy of system user-related initialization data,
- **userdata-qemu.img** - Android SDK writes user specific data and runtime session information on this image.

Our aim is to create user information that will not be not lost after emulator relaunch. Therefore, for this experiment we created *userdata-qemu.img* image with the information under emulated Android 2.2. This image can be used in later Android emulator [30] versions due to backward capabilities. Following data were generated in popular and necessary applications, which are used commonly on mobile devices (for more information see the Appendix B):

1. *Phone book*
   ```
   Ivan Petrov 476-666-66
   ivan.petrov@gmail.com
   Teknologiveien
   Gjovik
   Norway
   2815
   HiG
   ```
   See the Figure 51 for details.

[1] Secure Digital memory card - popular format of non-volatile storage: https://www.sdcard.org

2. *Browser history*
   `yandex.ru`
   `ria.ru`
   `tsn.ua`

   See the Figure 47 for details.

3. *Calls*
   `to Ivan Petrov`
   `9.43 pm`
   `Friday 22,`
   `2013`

   See the Figure 48 for details.

4. *SMS*
   `to Ivan Petrov:`
   `Testimon`
   `9.44pm`

   See the Figure 49 for details.

5. *Mail boxes*
   `cmatlis@ukr.net`
   `password: cmatlis1`
   `name: Andrii`

6. *Sent mails*
   `to: andrii.shalaginov@hig.no`
   `topic: Androidlab`
   `body: Qwerty`

   See the Figure 50 for details.

3.2.2 Artifacts

Before extracting features from application and its tests results we need to look into artifacts inside mobile platforms, which can be examined. As we mentioned before, iOS is a conservative and more protected system. That is why we examine Android platform more strongly and precisely for artifacts seeking.

After extensive study of Android platform, we decided to concentrate on some of the platform's discoverable data. Basically these artifacts in bunch allow to characterize each application as unique and unconventional instance under dynamic and statical analysis.

In security features extraction our attention is focused on the next artifacts that are closely related to an application [5]:

1. Android Android Package File (APK) [30]

   1.1. Total size of package

   1.2. Entropy of different parts of package (helps to reveal encrypted information)
   1.3. Length of package name, length, etc
   1.4. Permissions
   1.5. Specific version requirements to Android platform.

2. *Computational resources*
   2.1. Central Processing Unit (CPU) utilization[2]
   2.2. Actual and virtual memory consumption [61]
   2.3. Amount of generated threads[3] during execution
   2.4. Occupied memory on device's storage

3. *Stored information*
   3.1. Volatile Random-Access Memory (RAM) memory
   3.2. Application's own folder[4]
      3.2.1. SQLite[5] databases
      3.2.2. Different files
      3.2.3. Binary libraries with Java Native Interface (JNI)
      3.2.4. Web cache and browser cookies
      3.2.5. Shared preferences
   3.3. External memory (like SD card)
   3.4. Different log files

4. *Application execution*
   4.1. Frequently requested host
   4.2. Type of user's sensitive information, which was transmitted
   4.3. User and system functions calls traces
   4.4. Log of the application launching process
   4.5. Stolen and transmitted IMEI[6] or UDID[7] codes

Usage of defined above artifacts significantly depends on application functionality, purposes and possible intentions. It may happen that application presents only static information on the screen and resources consumption is non-changeable as a result.

[2]In this particular case shows percentage of maximal possible CPU computational capacity, which is using by the running applications
[3]In computational theory, thread means smallest part of the data processing that can be assign by the CPU. One running process (application) can contain several threads, which can be executed in parallel and independently
[4]All information related to installed applications inside Android device is stored in the folder '/data/data/application'
[5]SQLite - lightweight relational database management system without client-server separation of the architecture
[6]International Mobile Equipment Identity
[7]Universally Unique Identifier

3.2.3 Nature of data

Since artifacts have different information nature and expression, our task is to process them properly for further using. Computer-based realization of statistical methods can hold naturally only numerical data so all used features are transformed from high level to low level numerical data. Additionally, this data needs to be scaled and normalized in order to achieve generalization. So, the task is to perform preliminary statistical pre-processing of raw security features for getting appropriate classification results.

For defined problem one can distinguish following possible data types, which can appear during application testing [62]:

- *According to quality measure:*
  1. Categorical
     1.1. Nominal (unsorted)
     1.2. Ordinal (can be sorted according to some criteria; have central tendency)
- *According to quantity measure:*
  1. Binary
  2. Continuous
  3. Discrete
  4. Range

As an example of categorical data we can consider requested permissions. Each of the permissions has following string format 'RECEIVE_SMS' that briefly describes the feature or a functionality of the device that an applications intends to use. In order to extract security-related meaning we decided to assign degree of security risk that such permission can cause according to studies [63, 64]. Therefore, we performed mapping as it is depicted in the Table 2. It was empirically defined 5 levels of risks: 0 - low, 1 - medium, 2 - high, 3 - dangerous, 4 - critical. The higher the number assigned to a permission, the more damage it can cause.

In additional, it should be mentioned that non-numerical features have to be transformed in corresponding numerical values with entire meaning preserving. This is done by utilizing probabilistic modeling of each item appearance in nominal and ordinal features types.

3.2.4 Features extraction

Most of the contemporary mobile applications have a complex and an object-oriented nature with implementation of sophisticated functionality. In addition to this, various code obfuscation, fragmentation and slicing techniques were invented, which make manual reverse engineering process difficult, time-consuming and sometimes infeasible [12, 65]. That is why our proposal is to concentrate first on "black box" tests (to be discussed in the Section 3.3) of the application. It gives an opportunity to develop unified test concepts that suits for different application testing.

Initially, features that are extracted based on pure static tests. During this phase an application is not executed. Finally, feature extraction from dynamic tests is performed in protected emulated environment .

Permission name	**Risk Level**
READ_EXTERNAL_STORAGE	1
WRITE_EXTERNAL_STORAGE	1
READ_SMS	2
SEND_SMS	2
RECEIVE_SMS	2
READ_CONTACTS	2
WRITE_CONTACTS	2
WRITE_SECURE_SETTINGS	3
AUTHENTICATE_ACCOUNTS	3
PROCESS_OUTGOING_CALLS	3
READ_LOGS	3
BILLING	4
ADD_SYSTEM_SERVICE	4

Table 2: Mapping categorical permissions names to numerical risk levels

During static applications testing following goals are achieved:

- *Code structure traversal*

  Disassembled Android Dalvik Executable Format (DEX) file can be characterized by different features: amount of implemented functions, functions calls, required libraries or resources, used variables, stored in functions predefined information, etc.

- *Processing of accompanying to application information*

  Following descriptive statistics features to be gathered: entropy (gives understanding whether stored data is encrypted or compressed by means of bigger value of entropy [66]), seeking for frequency of specific keywords (function names or locations), ip addresses, etc.

- *Calculation of different application's parts sizes*

  Reveals hidden and unusual data that are stored inside application package. May contain malicious payloads.

After artifacts analysis, security features are constructed manually based on expert knowledge. This is a crucial task in ML that allows us to utilize automated application testing and analysis on prepared data. After features construction it was empirically chosen several domains for building corresponding security metrics. Each of the domains can be described by several relevant features. Then, RELIEF method was applied to weight features in security metrics.

There are a lot of possible artifacts and we decided to concentrate on the most important and relevant for the defined problem. Following security-related features are going to be derived during statical and dynamical tests including continuous behavioral analysis:

Feature name	**Description**
id_featureSet	Identity number of processed features during statical and analysis test of application

id_app	Identity number of an application in database
id_test	Identity number of performed test
sdkVersion	Minimal version of software development kit (Android API version), needed for application execution
targetSdkVersion	Target version of software development kit (Android API version), necessary for successful application execution and running all developed functionality without restrictions
app_label_length	Length of the application's label
package_name_length	Length of the application's label (usually has a format 'com.application')
filesize	Size of APK installation package file
permissions_highest	The highest numerical level of permissions [!!!link to description]
permissions_avg	Average numerical level of permission
permissions_number	Amount of permissions requested by application
pull_data_size	Amount of application's own data that were stored and collected during dynamic test cycle in application's folder '/data/data/application'
log_launch_size	Size of pulled log file that contains verbose system output during application launch process
cpu_usage_peak	Peak value of CPU utilization during dynamical test of application
cpu_usage_avg	Average value of CPU utilization during dynamical test of application
cpu_usage_stdev	Standard deviation value of CPU utilization during dynamical test of application
thr_usage_peak	Maximal amount of created threads by an application during dynamic test
thr_usage_avg	Average amount of threads created by an application during dynamical test
thr_usage_stdev	Standard deviation of amount of threads created by an application during dynamical test
vss_usage_peak	Maximal size of virtual memory used by an application during dynamic test
vss_usage_avg	Average size of virtual memory used by an application during dynamic test
vss_usage_stdev	Standard deviation of size of virtual memory used by an application during dynamic test
rss_usage_peak	Maximal size of resident memory used by an application during dynamic test
rss_usage_avg	Average size of resident memory used by an application during dynamic test
rss_usage_stdev	Standard deviation of size of resident memory used by an application during dynamic test
shared_prefs	Number of shared preferences XML files in '/data/data/share_prefs' folder
shared_prefs_size	Size of all shared preferences files
databases	Number of stored databases in '/data/data/databases'
databases_size	Size of all stored databases

files	Number of stored databases in '/data/data/files'
files_size	Size of all stored files
package_entropy	Shannon entropy of the APK installation package [66]. The Entropy is a measure of an information uncertainty, which helps to reveal encrypted information inside the file
package_number_files	Total number of files in APK package
manifest_size	Size of APK configuration file 'AndroidManifest.xml'
res_folder_size	Size of the folder with additional resources such that interface and multimedia elements
assets_folder_size	Size of directory with application's assets
classes_dex_size	Size of Dalvik Executable Format (DEX) files (already compiled Java classes)
classes_Dex_entropy	Shannon's entropy of Dalvik Executable Format (DEX) files
execution_time	

Table 3: Labels and description of the security features

The assumption regarding relevance of all features is not always true and depends usually on the application functionality. Therefore, we try to select most stable and robust ones.

3.2.5 Security metrics construction

During statical analysis of given application and dynamic testing, we are able to extract multiple features (or measurements), which describe each particular application. Metrics are more related to human interpretation than low-level raw input features [9]. Our task is to create metrics that are repeatable, measurable and can characterize some domain by bunch of raw features. It means that previously mentioned features need to be converted into a numerical metrics, which are expressed in form of mathematical equations.

In order to assess influence of malware on such important information-security assets like Confidentiality, Integrity and Availability we need to perform related security-metrics construction. One can consider extracted features as a characteristics of some domains of mobile platform. So, they can be composed into single security metric, which has much more significance than single feature. Moreover, security metrics are treated as particular domain's property.

Our proposal for security metric is to utilize linear combiner over features, which are in the same domain and have similar meaning:

$$Metric_i = \frac{1}{N} \cdot w_k \cdot \sum_{k=1}^{N} Feature_{k_i} \tag{3.1}$$

Where $Metric_i$ - corresponding security metric, $Feature_{k_i}$ - a feature that belongs to this metric, w_k - weight of a particular feature in the security metric and N - amount of features that are related to the metric.

As an example, CPU utilization can be characterized by a single security metric:

$$CPUMetric = \sum_{k=1}^{3} cpu\_usage_{m_k} \cdot w_k \tag{3.2}$$

Where $CPUMetric$ - is a security metrics that describes CPU usage, w_i - weight of each particular feature in this metric and set of corresponding features is $m = \{avg, stdev, peak\}$.

Weights define significance and level of influence of each particular feature on security metric. The optimization procedure along with expert knowledge needs to be applied in order to find optimal weights. After study of corresponding feature-selection approaches for binary, RELIEF was chosen as the most appropriate method [10, 67]. It provides feature selection and attributes quality evaluation for defined binary classification problem. What is more, the RELIEF estimates not only correlation between the features, yet also interrelation. Additionally, RELIEF shows the importance of particular attribute in distinguishing between instances from opposite classes (for the details, please see the Figure 3.3). Nevertheless, feature selection problem and particularly RELIEF are open for the future study.

$$W_i = W_i - \mathrm{diff}\,(x_i, nearHit)^2 + \mathrm{diff}\,(x_i, nearMiss)^2 \qquad (3.3)$$

where x_i - an input pattern, W_i - merit (weight) of corresponding feature, $nearHit$ - nearest example for the same class, $nearMiss$ - nearest example from the opposite class.

The features were divided into several domains: statical tests data, dynamical tests data, resources usage, permissions, SDK details. Then, RELIEF was applied to each domain that gives features weights.

3.3 Malware detection using ML

ML an PR were defined previously as a powerful and suitable tool for malware detection problem. In this Chapter we support idea for ML usage by considering different sides of the detection process. In this thesis, an application testing is considered as a "black box" process [68]. It means that there used only low-level observable information, yet not an abstract-level information such that functionality or implementation details. "Black box" paradigm to be used in dynamic tests is shown in the Figure 4.

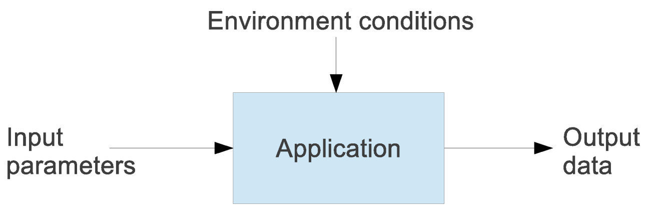


Figure 4: "Black Box" testing scheme

By utilizing mentioned paradigm we want to dig into two following problems:

- Features extraction from available observation in "black box" testing process, which is necessary for detection.
- Reliable malware detection as automated process using ML instead of non-automated analysis like androidguard [69].

3.3.1 Dynamic-focused methods

Dynamic "black box" test routine targeted on collecting behavioral observations in volatile dynamic environment. Generally it includes next steps: start data logging process, application installation and launch, user activity simulation, application stop and installation, stop logging process. User simulation allows to execute predefined combinations of actions in application to be tested. Additionally, network is simulated in order to capture live network traffic, which is generated by the application.

In addition to static test, dynamic test has to be executed for achievement of more comprehensive understanding of general intentions and functionality of an application. Basically dynamic application test is an execution in emulated or real environment with pre-selected sequence of actions. It is a powerful instrument that helps to collect data, which characterizes an application over period of observation. Information flow tracking, function calls and network traffic are going to be extracted. Eventually, features extraction from different behavioral observation is done. We mention some of the possible statistical models, which cover different domains of behavioral analysis.

Probabilistic models

Dynamic behavioral test is a stochastic process that is defined by an application and execution conditions. In spite of the fact that test initial test conditions are equal, the observable data varies. Probabilistic models are important for modeling some covert process and prediction future actions. The most powerful approaches, which found application in malware detection are Hidden Markov Model, Gaussian Mixture Model, n-gram, etc.

For example, probabilistic model can be used for detection patterns in log files. Log files are verbose output of some events. Based on frequency of such events and their order in sequence, we can predict whether application is trying to execute malicious operation or such operations are benign. Example of such log file obtained while launching Android application is presented in the Appendix D.

In real world, no absolutely correct prior information is available. Therefore, this information defined from empirical considerations either extracted from collected statistics. The idea behind utilization of probabilistic models is forecasting nature of some process. It means that unknown or missing data can be easily modeled by such models.

Time series

Some of the behavioral data observable in dynamic test represent sequence of simple events (or measures) like CPU or memory utilization. Such sequences are called time series and usually are taken within equal time ranges. It can be processor load, memory usage, network traffic, etc. For analysis it may be required to have more previous observations than already exist. Due to software and hardware delays, the resources usage measurements can be obtained once in 3-10 seconds while communicating with emulator. Also it might be required to predict future value in a time range, which is unavailable in previously accumulated information. Therefore, time series analysis might be used as a powerful supplementary tool.

From observed data during dynamic tests we use cubic spline[8] approximation model in order

[8]Approximation by piece-wise polynomial function with defined degree

to predict values of consumption of the resources for past events and future. Samples of the implementation is C++ is presented in the Appendix F. The results are depicted on the Figure 5, where X-axis - time in seconds and Y-axis - CPU utilization. As it can be seen, the red points in the Figure represent predicted values.

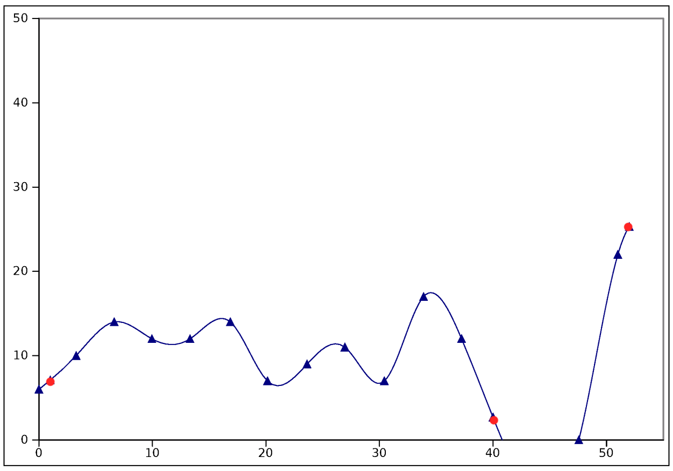
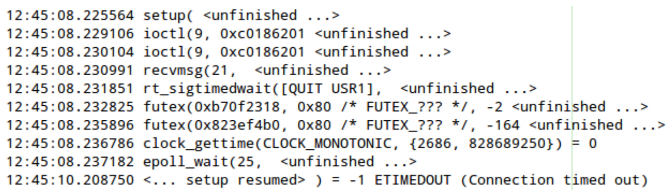


Figure 5: Sample of approximation usage. Blue points - obtained values, red points - approximation of previous existing interval values (at 1 and 40) and future one (at 52)

Additionally, for this purposes can be used different methods with outliers and missing data correction: Extrapolation, Linear Prediction, Alignment Prediction, Kalman Filter [70]. Finally, we would like to emphasize that time series analysis is a promising area of ML, which can find application in malware detection.

Information-based dependencies

Dynamic behavior of an launched application can be characterized by function calls during execution. As it was studied before [71], all application call a huge number of functions during launch and running process. Such sequences of user's or system's functions calls may produce calls graphs. Similar obfuscated malware implies same functionality and uses the same functions, yet in slightly different calling order. In the Figure 6, the function calls of initial installation of the Android application is depicted.



```
12:45:08.225564 setup( <unfinished ...>
12:45:08.229106 ioctl(9, 0xc0186201 <unfinished ...>
12:45:08.230104 ioctl(9, 0xc0186201 <unfinished ...>
12:45:08.230991 recvmsg(21,  <unfinished ...>
12:45:08.231851 rt_sigtimedwait([QUIT USR1],  <unfinished ...>
12:45:08.232825 futex(0xb70f2318, 0x80 /* FUTEX_??? */, -2 <unfinished ...>
12:45:08.235896 futex(0x823ef4b0, 0x80 /* FUTEX_??? */, -164 <unfinished ...>
12:45:08.236786 clock_gettime(CLOCK_MONOTONIC, {2686, 828689250}) = 0
12:45:08.237182 epoll_wait(25,  <unfinished ...>
12:45:10.208750 <... setup resumed> ) = -1 ETIMEDOUT (Connection timed out)
```

Figure 6: Android application function calls sequence obtained from strace [72]

Such function call traces can be expressed by means of the graphs represented in the Figure 7. The obtained graph can based in information-based dependencies comparison for similar patterns detection.

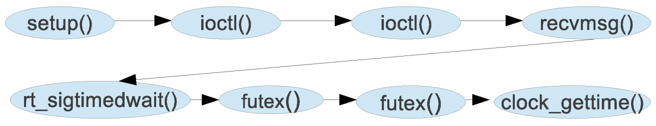


Figure 7: Graph of the Android application function calls

Therefore, graph-based dependency matching techniques might be used for pattern allocation in dynamically tested applications. It means that by comparing such patterns, malware detection method can be enforced and protected against polymorphism and obfuscation.

3.3.2 Feasibility of building automated malware detection expert system

Due to constraints in common user's knowledge it is not possible to prevent malware even with sophisticated detection solution. This problem appears because malware-prevention systems usually requires human interaction in critical situations.

The expert system as a solution for mentioned problem can use both knowledge database and user interface for interaction as it is shown on the Figure 8. However, the challenge of expert systems utilization is to achieve appropriate trade-off between required operator knowledge and detection rate [73]. User interaction has to be kept as little as possible for performance and stability of detection considerations.

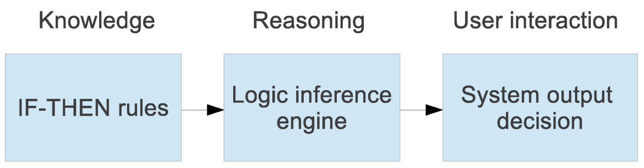


Figure 8: Expert system model

By looking toward theory, we are considering a possibility of making on-line and trainable malware detection expert system. We are proposing to use FL as a flexible mechanism in dealing with behavioral data processing. Such an expert system is suing previously collected and learned knowledge for performing detection without human interaction. Even taking into consideration complexity of such system it allows to optimize malware prevention process.

Since malware is developed usually by unknown people or organizations, it appears hard to obtain comprehensive entire structure and implemented functionality. Hence, manual malware reverse engineering is the most appropriate approach for getting such sort of knowledge. As malware reverse engineer intends to manually disassemble and analyze any kind of programs. Although, such methods has a high level of efficiency, precise manual processing is infeasible under time constraints and numerous samples. Thereafter, detection signatures should be produced.

Taking into account mentioned challenges, our proposal is to utilize ML as both program analysis and malware detection procedure. Initially, all possible features are extracted utilizing statistical methods. Finally, detection procedure is performed based on extracted decision rules. This is elaboration on the author's previous work on automated comprehensive mobile applications analysis and detection [16].

Perspective in big data analysis

As it was mentioned before, manual applications testing and analysis are not appropriate for examination of large collections of these applications. Current applications databases such that mobile app markets (like Google Play [20] or Apple Store [22]) and entire company's application pool are growing exponentially and continuously. Additionally, they are widely distributed and rapidly changing environments with proactive nature. Most of the existing basic ML solutions are targeted on off-line data processing and learning from training set [49]. In case if training set is changing, statistical model should be retrained from the beginning. These are the challenges that appears while dealing with big data analysis [74].

For slackness of defined challenges our goal is to use ANN learning for rules extraction. ANN is trained from input pattern gradually, which allows to utilize more recent data and apply forgiveness principle over "old" data. It implies that detection rules extraction is appropriate for using in a huge data on-line analysis with dynamic nature. In contrast to off-line methods, ANN helps to reduce overall operation complexity and training time for building corresponding statistical models. It can be also added that proposed malware detection utilizes ML over both dynamical and statistics test data unlike existing Google Bouncer technology [38].

3.4 Analysis and neuro-fuzzy rules extraction for malware detection

In this Section the author gives overview of existing fuzzy logic rules construction and their significance in expert systems. Furthermore, our own procedure for binary classification problem with rules tuning is presented.

Basically, fuzzy rules that are going to be studied further represent linguistic patterns extracted using neuro-fuzzy approach:

$$\mathrm{IF\ \ METRIC1} = A\ \ \mathrm{AND\ \ METRIC2} = B)\ \ => \ \mathrm{CLASS} = [benign|malicious] \tag{3.4}$$

Where A and B can be any linguistic quantitative terms.

Human understandable decision can be treated as scientific knowledge derived from scientific methodology according to the Daubert criteria. Therefore, this is a crucial part of forensics investigation and understanding of malware's success conditions can be resolved by proposed methodology.

3.4.1 Overview of the procedure

Basically, through this work we are seeking for malware detection approach, which could provide both reliable detection process and native human understanding of this process. From studied researches [13, 15, 14], it can be concluded that FL is a more appropriate ML procedure for this task. The FL represents itself a powerful and well studied area of classic logic, which is widely used in computer-based decision making systems [75, 76]. It combines both adjustable features sets and deducible by human reasoning and conclusion.

By contrast with classical mathematical logic, FL utilizes linguistic variables in conclusion making process. Transition from numerical, continuous or ordinal values features to a linguistic variables is performed [76]. This operation utilizes various statistical methods in order to obtain numerical parameters, which characterize each term of a corresponding variable. The linguistic variables represents fuzzy sets of a particular raw input feature or security metric [75, 77].

3.4.2 Fuzzy logic

Despite of the fact that FL is a domain of classical binary logic with two possible values (*True* and *False*), fuzzy logic outcome conclusion could be any real values in [0,1] interval. This is because of usage so-called degree of fuzzy statement's membership, which is defined as a *membership function* [76]. So, each statement IF-THEN in FL can be not only TRUTH or FALSITY, yet also may hold concept of PARTIAL TRUTH that agrees with human understanding and reasoning.

Fuzzy sets inherited belongings principle from classical logic theory. Each set includes elements with the same sense or similar characteristics, which distinguish them from others. Yet, element of fuzzy set has degree of membership to this set [76]. General set is a collection of elements $A = \{(x_i)\}$, which belongs to a set with 100% degree. In FL, one element can be assigned to more than one fuzzy set. So we have:

$$A = \{(x_i, \mu(x_i))\} \tag{3.5}$$

Where $\mu(x_i)$ - is a membership degree of a particular element of this fuzzy set.

The membership degree is defined by membership function, which depends on statistical nature of the corresponding feature in a fuzzy set. There could be mentioned few possible Membership Function (MF) [75, 78]:

- Triangular

$$\mu(x_i) = max(min(\frac{x-a}{b-a}, \frac{c-x}{c-b}), 0) \tag{3.6}$$

- Trapezoidal

$$\mu(x_i) = max(min(\frac{x-a}{b-a}, \frac{d-x}{d-b}), 0) \tag{3.7}$$

- Gaussian

$$\mu(x_i) = \frac{1}{e^{\frac{(x-a)^2}{2b^2}}} \tag{3.8}$$

- Bell-shaped

$$\mu(x_i) = \frac{1}{1 + |\frac{(x-a)}{c}|^{2b}} \tag{3.9}$$

All the statistical parameters (a, b, c, d) in equations above are taken from the analysis of a given fuzzy set's elements. Example of parameters influences on membership functions are presented in the Figure 9. Most real data contain white noise and fault entries, which influences all elements in fuzzy set to be normally distributed (for more details see the Figure 26). It means that the most appropriate membership function for the malware detection based on real data is Gaussian[9] membership function.

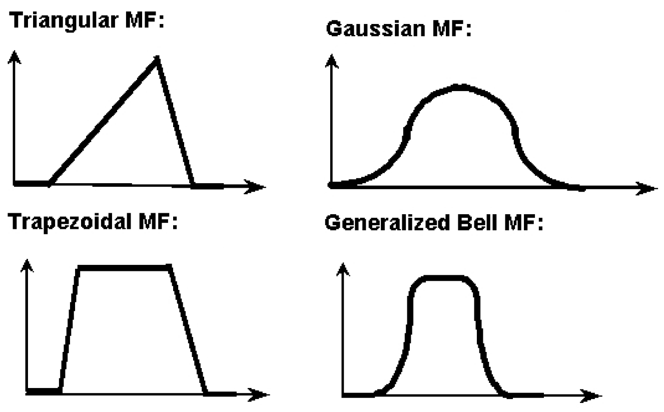


Figure 9: Membership functions examples [79]

The most important phase of FL statements construction is a transition from general raw input features or metrics to linguistic variables [79]. One can describe it as following:

1. For a particular new linguistic variable, there should be defined subjective linguistic terms: $T(linguisticVariable) = term_1, term_2, term_3$. These terms can be used as general measure applicable in most cases: $LOW, MEDIUM, HIGH$ or other specific values.

2. By utilizing statistical analysis over input feature $x_0 ... x_n$, we extract defined above (in the Figure 9) parameters (a, b, c, d) for membership functions for each term in a linguistic variable.

3. Calculation of Membership Function (MF) $\mu(x_i)$ for arbitrary test raw input feature x_i for all used terms in linguistic variable are performed. The minimal value

One can sketch the general transformation process that assigns a linguistic term with degree of membership to a new input raw feature. So this process provides a fuzzy association to many linguistic terms with truth value, which is much better to use in computational intelligence systems instead of discrete associations. It means that one can organize detection procedure in a way, which is appropriate for both detection and human natural reasoning. Decision rules are based on linguistic variables as values of logical statements.

3.4.3 Rules extraction using neuro-fuzzy

Because of we need to resolve the binary classification problem, our task is to construct decision trees for two classes utilizing fuzzy logic. This means that the general logic statement are extracted:

[9]The Gaussian or the Normal distribution is usually used in case if it is necessary to deal or to model the real life case data.

IF $x_0 \in \mathit{LinguisticVar}_0(\mathit{term})$ &...& $x_{n-1} \in \mathit{LinguisticVar}_{n-1}(\mathit{term})$ THEN $y \in C_k$, where $\mathit{LinguisticVar}_0$ and $\mathit{LinguisticVar}_{n-1}$ are corresponding linguistic variables. Additionally, the corresponding terms for each linguistic variable should be found based on minimal value of Membership Function (MF). Each part $x_0 \in \mathit{LinguisticVar}_0(\mathit{term}_q)$ of the rule is called *atom* and $y \in C_k$ - *consequent* or *conclusion* [75]. In our classification problem, each consequent has only two possible verbal values (benign or malicious). Meanwhile, malware detection problem is a multi-domain problem, so there are significant amount of atoms in each rule.

It can be highlighted that rule-construction process becomes a considerable procedure in malware detection process. Consider rules construction as a processing and transformation from raw input features to a useful and a significant set of relevant rules. It combines two steps:

1. **Rules extraction**

   During this step, all possible rules are extracted using extracted fuzzy sets from raw input features. Complexity of this step equals to $O(M^n)$ and depends from amount of input features. It means that amount of all possible extracted rules grows exponentially with increasing input data dimensionality n. Also amount of possible terms in each linguistic variable M affects complexity as well.

2. **Rules selection**

   Since constructed rules contain multiple rules with irrelevant or insignificant sense, additional filtering has to be applied. By means of statistical learning we select the most relevant and weighty rules for defined problem.

Considering rule-selection task as the main optimization problem we would like to denote next Section to this step. For performing multidimensional and non-linear optimization, ANN is better solution than any existing evaluation methods. ANN provides human-like reasoning of the problem and can be natively interpreted as results.

Adaptive learning systems such as ANN have became important statistical modeling tools in recent few decades. Due to flexibility in model construction ANN is widely used in pattern classification. The main disadvantage of the ANN is a complexity of a statistical model that was extracted from data. This is because all obtained weights for each layers are difficult to interpret and to represent in human-understandable manner.

General feed-forward[10] single-layer[11] perceptron has functional structure, which is shown in the Figure 10. It consists of input neurons, neurons strengths defined by corresponding weights and linear combiner.

Building of a statistical model utilizing ANN includes following steps as any of ML and PR approaches [76]:

- **Training**

  During this phase, collected classified patterns (processed malware samples) are going to be processed. This process means extracting statistical parameters for construction of the

[10]The feed-forward ANN implies that the signals inside the network are going strictly from input to output without cycles and reverse connections

[11]The single layer ANN consist of only single layer of the output nodes

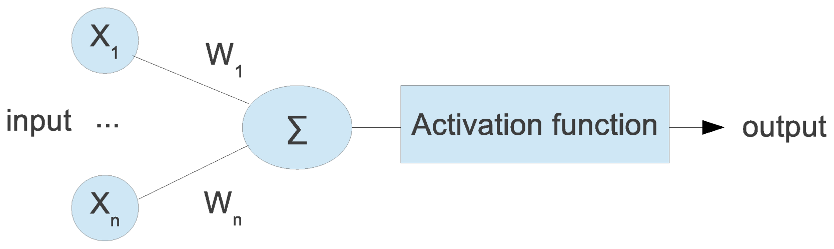


Figure 10: Scheme of a single layer perceptron

model. Training is an iterative mechanism that allows to extract the meaningful information for characterization of given data. Finally, this characterization provides possibility for new data classification without using given data set and expert knowledge.

- **Testing**

  Given new unclassified sample of application (pattern), ANN defines its class (malicious or benign) with some tolerance (error threshold) based on learned statistical model.

As the measure of learned model quality, *stopping criteria* can be utilized. It is necessary to use such criteria due to limit execution of Training phase when model does not show any improvement on it. That is why Training of ANN includes following stopping criteria:

- N_{epochs} - amount of epochs,
- $|d_i - y_i| < \epsilon$ - absolute difference between actual model output and desired class of the sample is less than defined error threshold,
- $|W_k - W_{k-1}| < \epsilon$ - absolute difference between ANN weights on current and previous step is less than defined error threshold,
- $|W_k' - W_{k-1}'| < \epsilon$ - absolute difference between derivatives of the ANN weights on current and previous step is less than defined error threshold.

Output signal from the combiner can be additionally processed by means of activation function. The weights define degree of influence with which corresponding input neuron affects output signal. By contrast with general ANN, in NF adaptive systems input neurons receives previously processed metrics or features while preserving general concept [76]. It means that there is a bigger amount of inputs due to considering of each term in each linguistic variable as a input neuron. Furthermore, that we treat each term's membership function as a neuron input instead of using raw input features.

In general simple NF system, additional layer of M^n linguistic terms is added as it is depicted in the Figure 11. Training phase is performed by means of adjusting each rule's weight. Finally, rules have the following format:

$$R_1: \text{ IF } x_1 \text{ is } A_1^1 \text{ AND } x_2 \text{ is } A_2^1 \text{ THEN } z \text{ is } B^1 \tag{3.10}$$

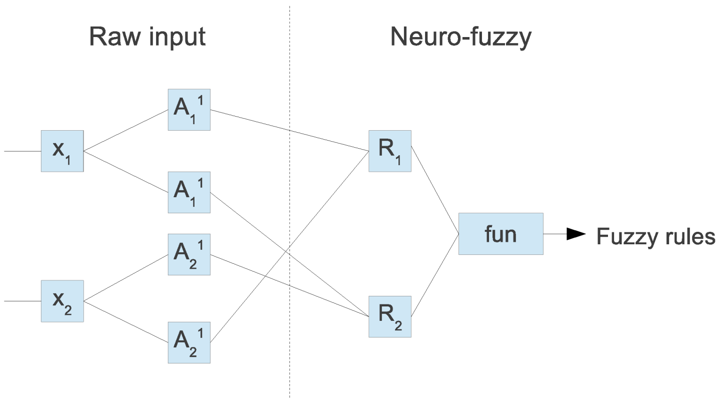


Figure 11: Simple NF system architecture

For our task we use following structure of the NF system:

- *Layer 1*

  Raw input data vector. This is security features.

- *Layer 2*

  Process of input data fuzzification according to defined fuzzy membership function and terms in each linguistic variable.

- *Layer 3*

  Rules extraction. Each rule represents one of M^n possible combinations of terms in linguistic variables.

- *Layer 4*

  ANN output signal to be compared with corresponding pattern (class identification number) of input data.

Mentioned previously NF procedure is an integral part of fuzzy inference system, which goal is to extract associative rules for decision making [56]. By applying fuzzy inference principle, training of NF system 11 can be treated as dual optimization problem. First, AND operand has to be applied in order to find the membership degree of each constructed rules with following format example [77]:

$$\text{IF } (x_0 \in LinguisticVar_0(term1)) \text{ AND } (x_1 \in LinguisticVar_1(term3)) \tag{3.11}$$

The membership degree according to T-norm [76] is calculated as next:

$$Class1 \wedge Class2 = \min(\mu_A(X), \mu_B X) = \mu_A(X) \cdot \mu_B(X) \tag{3.12}$$

We are using linguistic term membership function values multiplication as a way for fuzzy AND computing. Operation OR is based on maximization of membership values and is performed as following:

$$\mathrm{Class1} \wedge \mathrm{Class2} = \max(\mu_A(X), \mu_B X) = \mu_A(X) \cdot \ mu_B(X) \tag{3.13}$$

So for each extracted rule we compute rule's membership function using fuzzy inference principle as it is shown in the Figure 12.

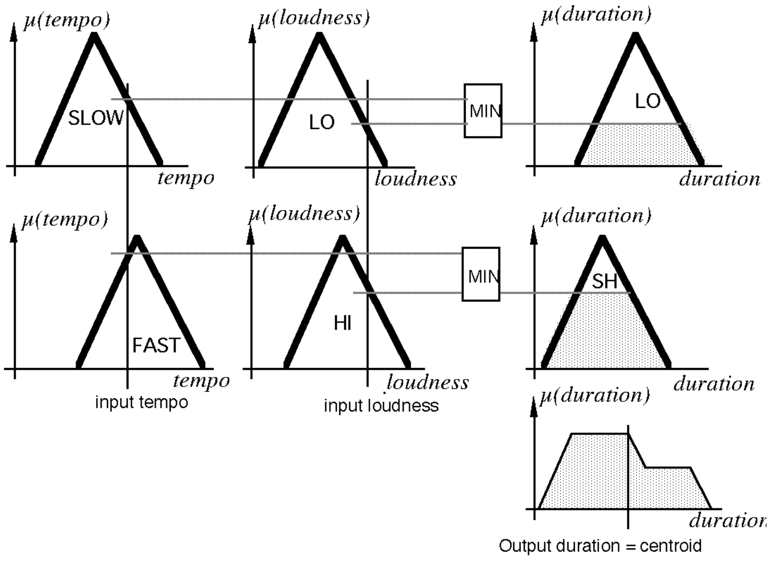


Figure 12: AND-OR rule-extraction principle[80]

Input features distribution defines distribution of output membership values for. Each rules in output has its own distribution over input linguistic terms. It means that depending on the unlabeled new data, the output rules membership function varies. Transition from input linguistic fuzzy terms in addition to transition security features to security metric provides more possibility to model precise non-linear dependency between input raw data and class label.

We realized that ANN is suitable adaptive procedure. Thus, it is used as rule-construction engine in our work. The rule-construction process is represented in the Figure 13. To cope with procedure understanding problem, we applying bunch of ANN and FL. So, NF ensembles both non-linear statistical modeling and rules extraction for malware detection.

Innovative rules tuning and selection

In this subsection, the authors are solving the critical question: whether all of the contracted rules are going to be used or amount of necessary rules can be decreased? We introduce our approach for rules selection based on consequent decision weights.

The extended structure of fuzzy inference system is presented on the Figure 14. Each layer should have corresponding array of weight for each neuron in this layer. This increases robustness and non-linearity of the NF. However, we are considering NF as a rules extraction instrument.

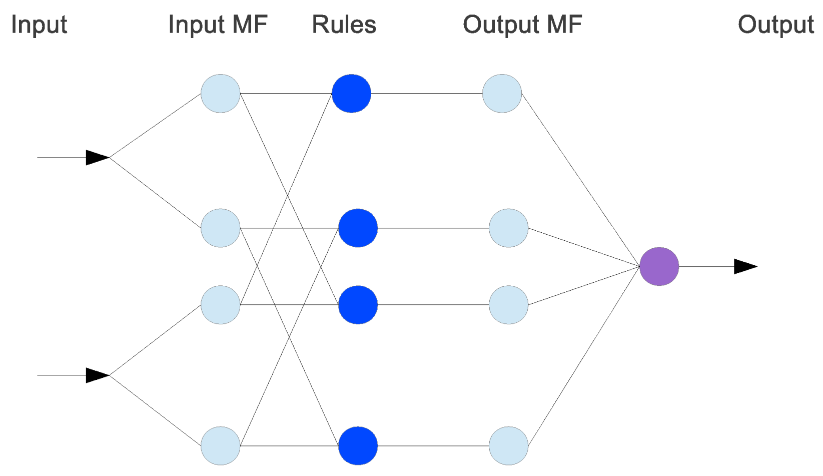


Figure 13: Scheme of rule-extraction process

That is why for our problem, the most important role play each rules layer weights. As basic approximation, we assign to input layer and to consequent layer equal weights and do not use them in calculations.

Extracted metrics tend to have normal distribution, which is characterized by central tendency and dispersion. Therefore, Gaussian functions is used for membership degree values calculation [76]:

$$\mu(x_i) = \frac{1}{e^{\frac{(x-a)^2}{2b^2}}} \tag{3.14}$$

After the rules extraction phase, we are faced with two problems:

1. *Rules redundancy*

   So, we need to prune M^n rules in order to obtain simpler and robust model. This model is described by selected rules and can classify application with some degree of tolerance.

2. *Covering the range of fuzzy set values*

   It might happen that due to complexity in metric's relationship, obtained statistical model (rules) can have overfitting [10]. It means that less significant rules (based on negligible amount of training samples) produces error on testing phase.

Problem 1 can be solved by utilizing Delta learning rule, which is variation of a Hebbian learning rule [82]. It justifies that the neurons, which have stronger degree of influence, have bigger value of connection's weight. As we need to select most relevant rules, the selection decision should include rules weights as a decisive factor [formula]. In other words, rules that have the smallest weights is excluded from statistical model. Finally, this approach provides most important and significant rules for classes separation.

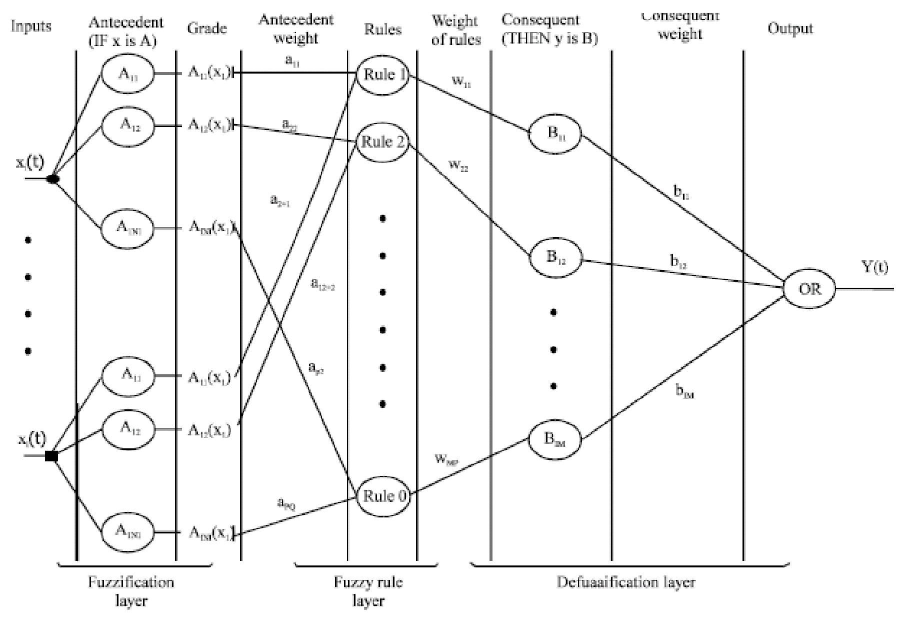


Figure 14: Sample of extended NF scheme [81]

Problem 2 is solved by transferring defined two classes problem in a single class domain: part of the samples are labeled "class A" and all others are "non class A". Therefore, according to logic theory, one can extract rules for only single class whose rules have bigger significance.

It should be mentioned that NF can not be trained for each class separately, because information from other class is lost. That is why input membership function's statistical parameters are extracted from all data set. Finally, NF is trained based on two-classes labeled data.

The main idea of our new proposed method is to utilize unsupervised and supervised ML approaches together in fuzzy inference system for rules construction:

- *Supervised learning* - approaches for training with labeled samples.

  Neuro-fuzzy takes as labeled input samples from both classes, which allows to build error cost function for weight optimization as following:

$$C = \sum (y_i - d_i)^2 \tag{3.15}$$

  This function is related to posterior probability of appearance of each input pattern after processing given data. The example of error cost function that is described by the Equation 3.15 is presented in the Figure 15. Our task is to minimize this function that provides us optimal weights values for each rule. By utilizing Gradient Descent [10] the cost function C can be optimized in order to get optimal value of weight. The optimization problem is related to rules weight. Therefore, the best direction of adjusting will be along the partial derivative of the cost function. After differentiating using chain rule, the partial derivative of the function will have following form:

$$\frac{\partial C}{\partial \omega_i} = \frac{\partial \left((y_i - d_i)^2 \right)}{\partial_i} = \frac{\partial \left((y_i - d_i)^2 \right)}{\partial d_i} \cdot \frac{\partial d_i}{\partial \omega_i} = -2 \cdot (y_i - d_i) \cdot \frac{\partial d_i}{\partial \omega_i} \tag{3.16}$$

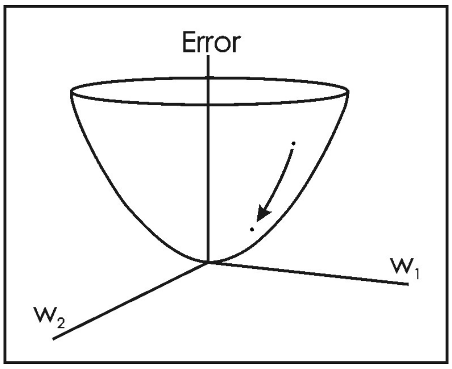


Figure 15: Example of weights error function with obvious local minimum

- *Unsupervised learning* - approaches for revealing concealed dependencies and structures in data.

  Processing extraction of constructed rules according to some user-defined criteria such that neuron weight or membership function values.

Error cost function represents multidimensional function over rule's weights as input variables. In order to find local minimum of this function, we use first-order optimization method - Gradient Descent (GD). Using the partial derivative of the cost function 3.16 we can find the optimal change of the weight:

$$\Delta\omega_i = -\alpha \cdot (y_i - d_i) \cdot x_i \tag{3.17}$$

where actual output of the Network is $y_i = \frac{1}{1+e^z}$, $z = \sum_i w_i x_i$ and x_i is an input pattern.

GD optimizes cost function in direction that is opposite to gradient of the error cost function in a particular input value. It means that each weight is adjusted separately from other and proportionally to projected error on current iteration. Despite solving optimization problem, there exists other drawback of GD usage. The rate α should be chosen optimally on in order to get local minimum for each of the weights on each iteration. The Figure 16 is showing difference in using different rates [83]. As the enhancement of the GD, one-dimensional optimization can be used in order to find optimal learning rate.

Finally, the constructed fuzzy rules should follow the next concept [84]:

- **Consistency**

  Set of rules has to be the same over different runs with the same input parameters.

- **Steadiness**

  Classification using extracted rules set should provide the same accuracy of classification on the same testing data set.

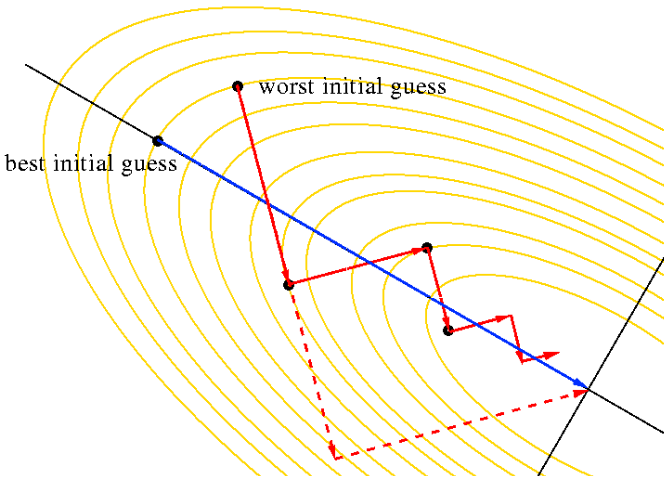


Figure 16: Influence of the learning rate on Gradient Descent performance: blue rate is optimal, red is adjustable [83]

Proof-of-concept demonstration

As a basic example we have chosen to utilize Iris Data Set for training and testing of proposed previously method [85]. The dataset consists of four numerical features: 'sepal length', 'sepal width', 'petal length', 'petal width'. In total there represented 150 instances. The fuzzy rules were used successfully for Iris Data set classification before [86, 87, 88], so we decided to execute proof-of-concept demonstration on this dataset. Despite the fact that this data set has originally three classes, we modify it for our purposes. The following binary classification problems were obtained with 100 data samples in each:

1. Iris Setosa vs Iris Versicolour
2. Iris Setosa vs Iris Virginica
3. Iris Versicolour vs Iris Virginica

Each of four features represented in Iris Data set has normal distribution as it is shown in the Figure 17. Using this assumption, we provide fuzzification using Gaussian membership function mentioned previously.

Among mentioned 100 samples, 90 are used as a training dataset and 10 as a testing dataset. Further experiments including training/testing phases were performed with equal amount of terms in each used linguistic variable. There were also utilized three (LOW, MEDIUM, HIGH) and 5 terms (VERY LOW, LOW, MEDIUM, HIGH, VERY HIGH). For linguistic variables with three terms, we use following membership functions centroids: $\mu - \sigma,\ \mu,\ \mu + \sigma$. In case of five terms, the set includes following centroids: $\mu - 2 \cdot \sigma,\ \mu - \sigma,\ \mu,\ \mu + \sigma,\ \mu + 2 \cdot \sigma$. The procedure can be sketched in pseudo-code as following algorithm depicted in a Pseudocode 1.

The results of proposed procedure execution are presented in the Tables 4 and 5. There was use ANN with 1000 epochs. More information and examples of extracted rules can be found in

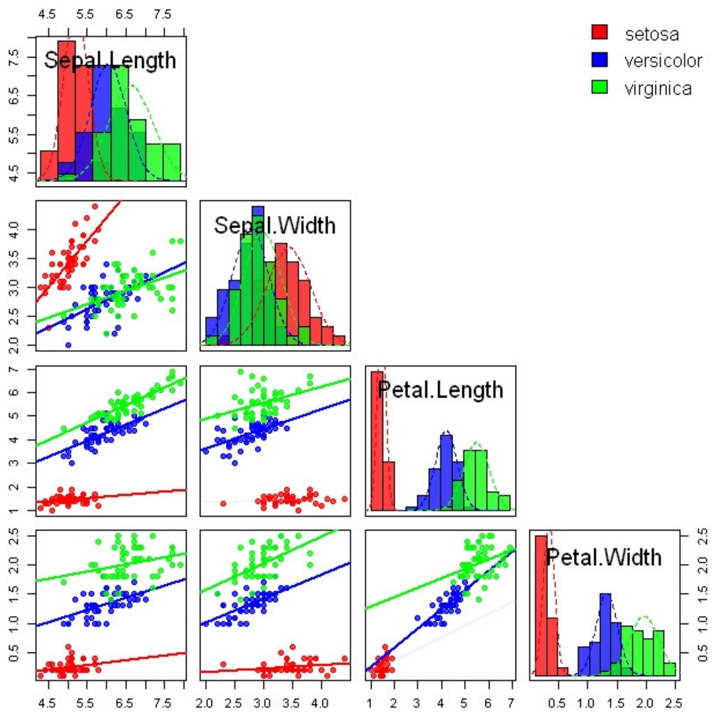


Figure 17: Distribution of Iris Data set features for different classes [89]

the Appendix C.

The Figures 18 and 19 show results of rules extraction process for Setosa-Versicolor classification problem with three and five terms in linguistic variable respectively.

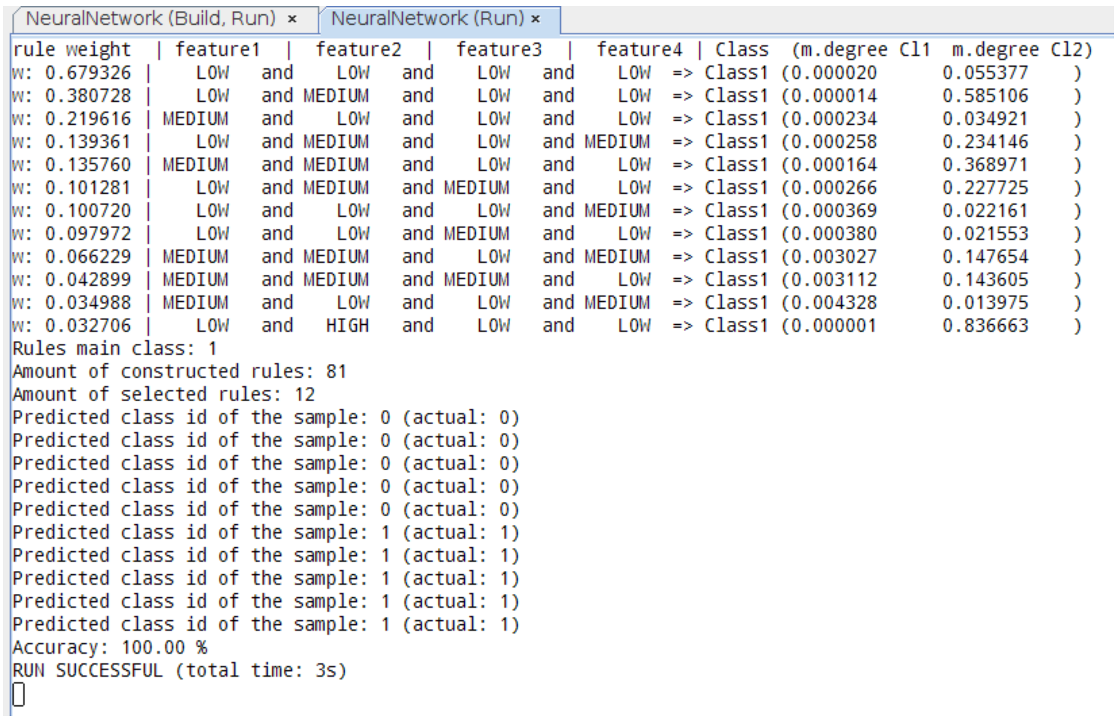
In compare to other studies targeted on fuzzy rules extraction our method provide more naive rules. Example of the corresponding fuzzy rules are shown in the Figure 20. It can be noticed that the rules represented in the Figure 18 has less complex expression. However, the detection accuracy of proposed method is not vary significantly from the method that was used in [86].

Anomaly-detection perspective

The malware detection problem with utilizing fuzzy rules can be solved from anomaly-detection perspective as well. The main difference between anomaly (behavioral-based) and signature (knowledge-based) detection mechanisms is using baseline of normal behavior [1, 2]. Particularly, one can distinguish two behavioral domains:

- *User* - specific interaction between a user and a device.
- *Program* - interaction a program and OS under specific conditions and triggered events, which mainly depends on user activity and stored information.

The baseline can be learned gradually or defined initially as a prior information. However, most of the available anomaly-detection solutions are experimental and depend on area of uti-



```
rule weight   | feature1  |  feature2  |  feature3  |  feature4 | Class  (m.degree Cl1  m.degree Cl2)
w: 0.679326 |    LOW  and    LOW  and    LOW  and    LOW  => Class1 (0.000020     0.055377    )
w: 0.380728 |    LOW  and MEDIUM  and    LOW  and    LOW  => Class1 (0.000014     0.585106    )
w: 0.219616 | MEDIUM  and    LOW  and    LOW  and    LOW  => Class1 (0.000234     0.034921    )
w: 0.139361 |    LOW  and MEDIUM  and    LOW  and MEDIUM  => Class1 (0.000258     0.234146    )
w: 0.135760 | MEDIUM  and MEDIUM  and    LOW  and    LOW  => Class1 (0.000164     0.368971    )
w: 0.101281 |    LOW  and MEDIUM  and MEDIUM  and    LOW  => Class1 (0.000266     0.227725    )
w: 0.100720 |    LOW  and    LOW  and    LOW  and MEDIUM  => Class1 (0.000369     0.022161    )
w: 0.097972 |    LOW  and    LOW  and MEDIUM  and    LOW  => Class1 (0.000380     0.021553    )
w: 0.066229 | MEDIUM  and MEDIUM  and    LOW  and MEDIUM  => Class1 (0.003027     0.147654    )
w: 0.042899 | MEDIUM  and MEDIUM  and MEDIUM  and    LOW  => Class1 (0.003112     0.143605    )
w: 0.034988 | MEDIUM  and    LOW  and    LOW  and MEDIUM  => Class1 (0.004328     0.013975    )
w: 0.032706 |    LOW  and   HIGH  and    LOW  and    LOW  => Class1 (0.000001     0.836663    )
Rules main class: 1
Amount of constructed rules: 81
Amount of selected rules: 12
Predicted class id of the sample: 0 (actual: 0)
Predicted class id of the sample: 0 (actual: 0)
Predicted class id of the sample: 0 (actual: 0)
Predicted class id of the sample: 0 (actual: 0)
Predicted class id of the sample: 0 (actual: 0)
Predicted class id of the sample: 1 (actual: 1)
Predicted class id of the sample: 1 (actual: 1)
Predicted class id of the sample: 1 (actual: 1)
Predicted class id of the sample: 1 (actual: 1)
Predicted class id of the sample: 1 (actual: 1)
Accuracy: 100.00 %
RUN SUCCESSFUL (total time: 3s)
```

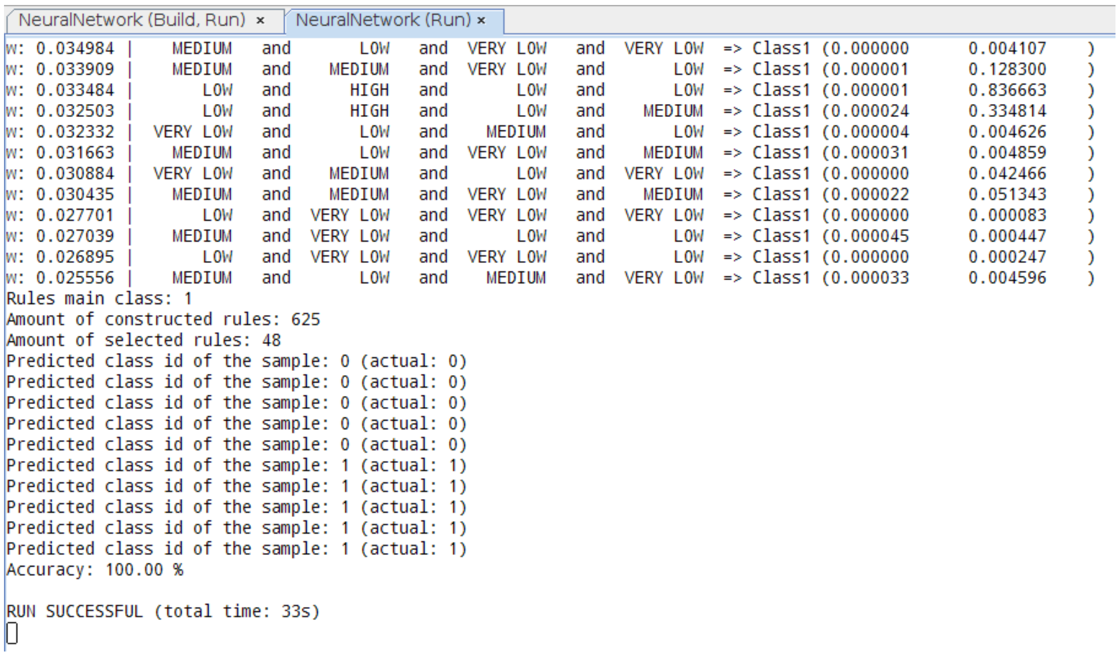
Figure 18: Rules for Setosa-Versicolor classification, three terms

lization. This is because user influences program execution and it varies a lot from host to host. As a result, there are no universal datasets of normal behavior such that in case of signature-based solutions.

One can state that for anomaly detection, proposed security metrics can be used. NF is able to learn continuously from new data and partially "forget" old one. That is why there is a perspective of tuning proposed method with special focus on behavioral analysis. Supporting metrics should be developed in order to characterize efficiently behavioral patterns. For example, Hidden Markov Model can be utilized for probabilistic modeling of program execution with dependency on user location and time [10, 90]. Our future work can be sketched as extension of developed method in order to analyze behavioral data. As it was studied earlier [91], a lot of behavioral data are accessible on mobile devices. Therefore, elaboration on behavioral analysis using NF can provide appropriate detection rate. Moreover, extensive study of user profiles is needed.

3.4.4 Application in big data analysis

Recently amount of applications in applications markets is growing exponentially. This leads to impossibility to process and analyze all the applications using available security solutions in reasonable time. Big amount of applications are submitting and new vulnerabilities are discovering every day. One can characterize such markets as a huge dataset, widely distributed, proactive, dynamic and rapidly changing environment. As results, the problems related to huge data are appearing [74].


NeuralNetwork (Build, Run) ×   NeuralNetwork (Run) ×

```
w: 0.034984 |   MEDIUM  and      LOW  and VERY LOW  and VERY LOW => Class1 (0.000000     0.004107    )
w: 0.033909 |   MEDIUM  and   MEDIUM  and VERY LOW  and      LOW => Class1 (0.000001     0.128300    )
w: 0.033484 |      LOW  and     HIGH  and      LOW  and      LOW => Class1 (0.000001     0.836663    )
w: 0.032503 |      LOW  and     HIGH  and      LOW  and   MEDIUM => Class1 (0.000024     0.334814    )
w: 0.032332 | VERY LOW  and      LOW  and   MEDIUM  and      LOW => Class1 (0.000004     0.004626    )
w: 0.031663 |   MEDIUM  and      LOW  and VERY LOW  and   MEDIUM => Class1 (0.000031     0.004859    )
w: 0.030884 | VERY LOW  and   MEDIUM  and      LOW  and VERY LOW => Class1 (0.000000     0.042466    )
w: 0.030435 |   MEDIUM  and   MEDIUM  and VERY LOW  and   MEDIUM => Class1 (0.000022     0.051343    )
w: 0.027701 |      LOW  and VERY LOW  and VERY LOW  and VERY LOW => Class1 (0.000000     0.000083    )
w: 0.027039 |   MEDIUM  and VERY LOW  and      LOW  and      LOW => Class1 (0.000045     0.000447    )
w: 0.026895 |      LOW  and VERY LOW  and VERY LOW  and      LOW => Class1 (0.000000     0.000247    )
w: 0.025556 |   MEDIUM  and      LOW  and   MEDIUM  and VERY LOW => Class1 (0.000033     0.004596    )
Rules main class: 1
Amount of constructed rules: 625
Amount of selected rules: 48
Predicted class id of the sample: 0 (actual: 0)
Predicted class id of the sample: 0 (actual: 0)
Predicted class id of the sample: 0 (actual: 0)
Predicted class id of the sample: 0 (actual: 0)
Predicted class id of the sample: 0 (actual: 0)
Predicted class id of the sample: 1 (actual: 1)
Predicted class id of the sample: 1 (actual: 1)
Predicted class id of the sample: 1 (actual: 1)
Predicted class id of the sample: 1 (actual: 1)
Predicted class id of the sample: 1 (actual: 1)
Accuracy: 100.00 %

RUN SUCCESSFUL (total time: 33s)
```

Figure 19: Rules for Setosa-Versicolor classification, five terms

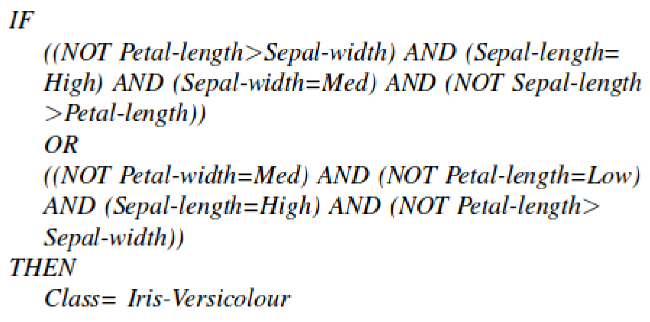
Several advantages of proposed method for a huge data analysis can be highlighted. Firstly, security metrics help to eliminate curse of dimensionality [92]. This is due to the fact that dimensionality of security metrics vector is much less than dimensionality of raw features vector. Consequently, complexity of the statistical model is less and learning time is shorter. Secondly, NF is fast, flexible and extremely adoptable method. Additionally, it has an ability to learn new data continuously without full re-training of the fuzzy model. Therefore, there is no need to update whole data base of signatures as it is in pure signature-based solutions.

Known obstacles

In this section we provide overview of possible obstacles that one can face during feature-extraction process.

From one side, one can consider input data as a source of errors and inaccuracies in rules extraction. It was mentioned previously that data preprocessing is a vital step in each ML process. Due to diversity of data types (like nominal and ordinal features) and noise in input data, rules can be extracted with considerable level of error. Therefore, utilizing of normalization and filtering helps to decrease influence of such negative factors.

From the other side, specific details of proposed method have to be taking into account. ANN is utilized as a complex and powerful pattern matching procedure that allows to solve us defined problem. However, due to significant theory beside it, there are some unavoidable drawbacks. This is in accordance to "no free cheese rule". Initially, we should consider overfitting of the statistical model in training phase. Due to this specificity of ANN, after some number of epochs, model is becoming over-trained. This leads to small error on training set validation and incremental error on testing set as it is presented in the Figure 21.



IF
((NOT Petal-length>Sepal-width) AND (Sepal-length=High) AND (Sepal-width=Med) AND (NOT Sepal-length >Petal-length))
OR
((NOT Petal-width=Med) AND (NOT Petal-length=Low) AND (Sepal-length=High) AND (NOT Petal-length> Sepal-width))
THEN
Class= Iris-Versicolour

Figure 20: Sample of fuzzy rules extracted for Iris dataset classification [86]

As a slack solution to this problem, one can propose to use more advanced stopping criteria and use them in ensemble. It ensures that the model does not overfit train data. That is why early stopping is needed for getting more generalized model.

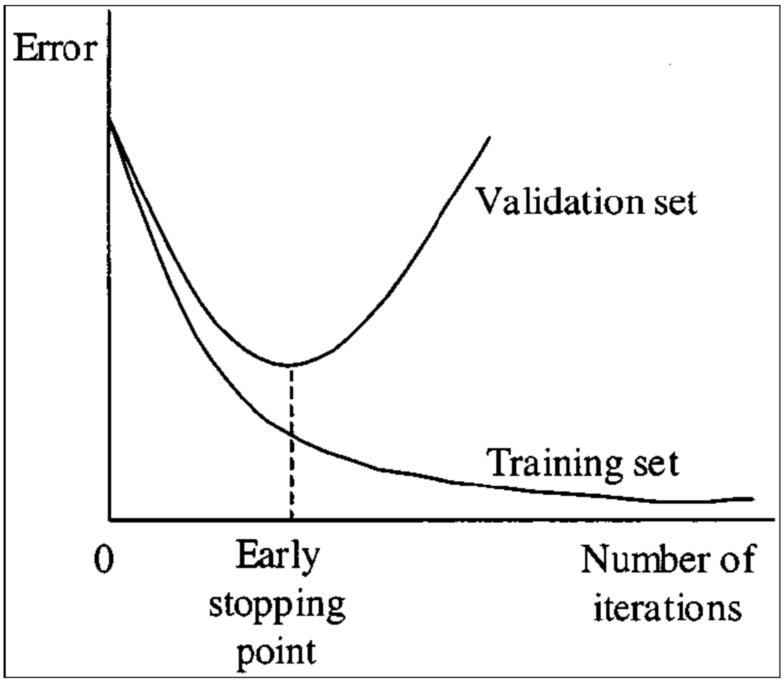


Figure 21: Effect of overfitting in ANN [93]

```
Data: Labeled dataset
neural network weights initialization;
read Training and Testing data set;
calculation of $\mu$ and $\sigma$ for each of classes and dataset in total;
neural network training;
while not successful Stopping Criteria do
    while not all of input patterns are processed do
        while not all of constructed rules do
            assign degree of membership for current input[i] $\forall$ terms;
        end
        linear combiner takes as input all membership degrees;
        while not all of constructed rules do
            calculate error rate for current rule;
            using Gradient Descent adjust corresponding weight;
        end
        calculation of activation function over all rules;
    end
end
sorting constructed rules according to weight value in descendant order;
defining class ID for each constructed rule;
while not all of constructed rules do
    calculate cumulative membership degree (CMD) for current rule;
    if CMD for C1 bigger than for C2 then
        assigning C1 to the rule;
    else
        assigning C2 to the rule;
    end
end
performing selection process of rules;
define main class ID of rule with the most significant weight value;
while not all of sorted constructed rules do
    if class ID of current rule is the same as main class then
        assign current constructed rule as a selected rule;
    else
        exit ;
    end
end
Result: Extracted malware detection rules
```

Algorithm 1: Proposed rule-construction algorithm

Data SubSet	Rules extracted	Rules selected	Accuracy	Main class
Setosa-Versicolor	81	12	100%	Versicolor
Setosa-Virginica	81	11	100%	Virginica
Versicolor-Virginica	81	11	70%	Virginica

Table 4: Results of rules extraction using proposed method, three terms in each linguistic variable

Data SubSet	**Rules extracted**	**Rules selected**	**Accuracy**	**Main class**
Setosa-Versicolor	625	48	100%	Versicolor
Setosa-Virginica	625	64	100%	Virginica
Versicolor-Virginica	625	58	70%	Virginica

Table 5: Results of rules extraction using proposed method, five terms in each linguistic variable

4 Experimental setup & Results

This chapter provides practical aspects and discussions for each defined previously research question. As a material of the experiment, data collection process is described. Additionally, statistical properties of collected data are going to be presented.

By utilizing our expert knowledge, we developed test framework for experiments justification of found theoretical answers. Initially, security metrics construction from collected applications are performed. Then importance of static and dynamic tests is shown. After that we concentrate our attention on malware detection suing linguistic rules and related topics. Finally, results of user profile influence on malware detection are overviewed.

Moreover, important implementation details and corresponding discoveries are examined. Our goal is not only to present practical proof for theoretical answer, yet also to reveal unusual and previously unknown pros and cons of the theory.

4.1 Overview of the collected dataset

This section contains description of the material that are going to be used in our experiments. Our experiments are conducted on manually constructed dataset, which was generated using malicious and benign applications. In point of the fact, we collected separately 521 samples of known to be malicious applications. It should be mentioned that some applications have the same functionality. However, most of the were slightly modified by attackers in order to avoid signature-based AV software. It means that MD5 hash sums[1] are different for each of used applications. Due to mentioned aspects, we finally got 388 unique malware with only 353 suitable for dynamic and static tests. Furthermore, 460 applications (251 suitable for testing) known to be benign from official Android Market were gathered. In this dataset, we assigned *Class 1* to a malicious application's features vector and *Class 0* to corresponding vector of benign application. For experimental purposes, it was used following collections and datasets:

- *benignApplications* - folder with collection of APK files of the applications defined as benign.
- *maliciousApplications* - folder with collection of APK files of the applications defined as malicious.
- *featuresDatasets/appsFeatures_all_filtered.arff* - dataset ARFF[2] file with extracted features after static and dynamic testing processes
- *securityMetricsDataset/constructedMetrics_all_filtered.arff* - dataset ARFF file with constructed security metrics from the extracted previously features.

[1]MD5 is an obsolete 128bit cryptographic hash algorithm, suitable for

[2]Attribute-Relation File Format, an ASCII file format used for storing data instances with common attributes. Applicable for using in Weka software package

The examples of used datasets are presented in the Appendix A.

The extraction process of feature vector for each of application to be analyzed is presented in the Figure 22. First, an application is sent in a specific way to the testing laboratory. Then, static and dynamic tests are performed with previously defined sequence of actions. This sequence is consistent over all tested applications. It means that tests conditions are equal and can easily be repeated later.

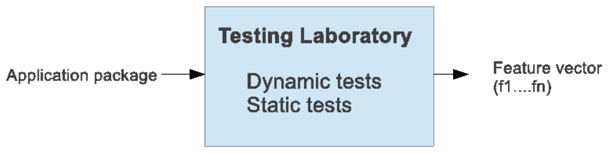


Figure 22: Scheme of the features extraction for each application during testing

For more precise understanding of the problem, descriptive statistics should be applied [10]. As it could be seen from the Figures 23, 24 and 25, there are noticeable dependencies between difference features. The last statements means that Pearson correlation[3] is not zero for these pairs of features and dependent [94].

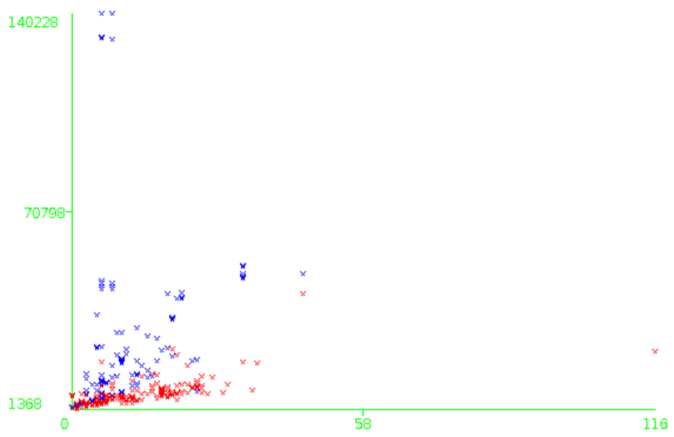


Figure 23: Correlation between 'permissions_number' and 'manifest_size' features for both classes

From the performed descriptive statistics analysis, we see that a features values have Gaussian (Normal) distribution (in the Figure 26. Therefore, as membership function in the NF approach it is logical to use Gaussian membership function, which fits to the dataset.

So, one can summarize that constructed data set has appropriate statistical properties and can be used with previously defined theoretical methodology.

[3]Pearson correlation is a coefficient of the linear correlation between two variables in data samples

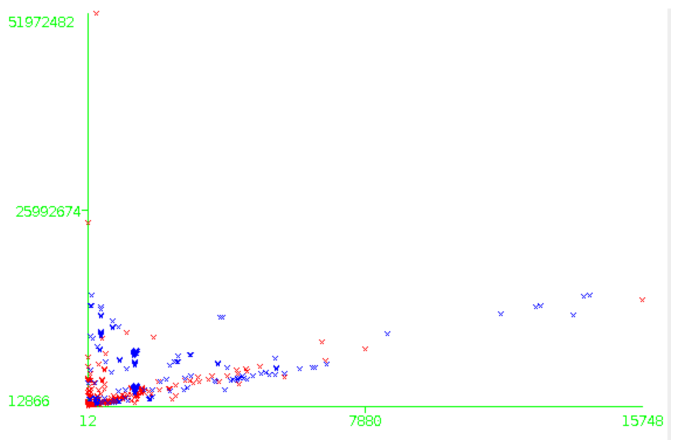


Figure 24: Correlation between ’res_folder_size’ and ’filesize’ features for both classes

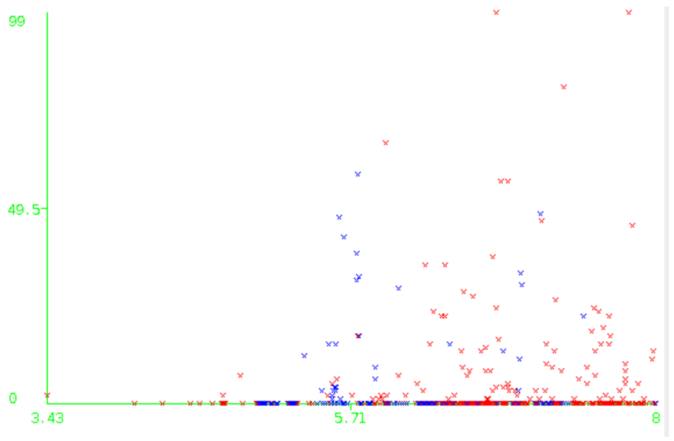


Figure 25: Correlation between ’package_entropy’ and ’cpu_usage_peak’ features for both classes

4.2 Extracted security metrics

The purpose of this section is to provide results of proposed security metrics usage in malware detection. As it was stated previously, security metrics represent additional transformation between features from raw data and ML approach. More specifically it means extracting a new layer of knowledge from already available.

For example, one can consider application’s CPU utilization time series over period of time. Since one can not use time series in a raw format as input data to the ML approach, we need to involve mapping to a single value. This transformation also should preserve information from time series and reflect its property.

Metrics construction process is depicted on the Figure 27. Each of the metrics is an abstraction of specific knowledge domain, which uses function over raw features related to this domain. Resources usage and functions structure can be considered as two different domains, while they might be characterized by more raw features. As a basic abstraction level, we concentrate our

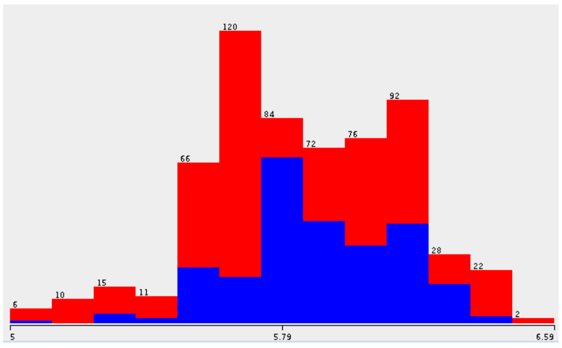


Figure 26: Gaussian distribution in extracted features

attention on linear combiner, which takes weighted values from input features. Each of the feature has own weights a_i that we defined empirically from background knowledge and available statistics information.

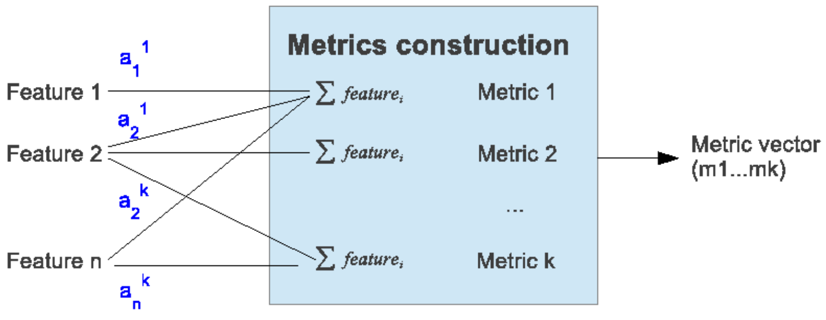


Figure 27: Scheme of the security metrics construction from the features

There is an obstacle, which have been faced during the method development. Features weights a_i need to be adjusted precisely through optimization methods instead of using prior available information. The last procedure was performed by using following classes in Weka package [94]:

- *weka.attributeSelection.ReliefFAttributeEval* - RELIEF method with amount of nearest neighbors for attribute estimation equal to 10.
- *weka.attributeSelection.Ranker* - search methods for ranking attributes in RELIEF.

It is important to utilize metrics rather than features, because such metrics add one more layer of non-linear abstraction to fuzzy inference systems. Finally, this fact provides more broader flexibility in building expert system. Therefore, posterior information from such methods is used in metrics construction based on collected statistical information. A positive property of proposed methods is an input vector data dimensionality reduction. Security metrics allow to use smaller amount of input data for building statistical model, which leads to mitigation of a curse of dimensionality [92]. Apparently, utilization of security metrics will influence the detection process, yet does not have significant impact on the detection rate.

4.2.1 Detection reliability

As was mentioned in the Chapter 3, expert knowledge and statistical analysis are using to build the metrics. Execution of RELIEF method together with 10-fold cross-validation gives us the following features merit (weights) for the dynamic metric 6 and sdk-related metric 7. Other metrics data are presented in the Appendix E (Tables 14, 15 and 16). After evaluation, weighted features are linearly combined into a metric.

feature	**average merit**	**average rank**
databases	0.015 ± 0.001	1.3 ± 0.64
log_launch_size	0.014 ± 0.002	1.9 ± 0.3
databases_size	0.008 ± 0.005	2.9 ± 0.7
shared_prefs	0.005 ± 0.001	4.3 ± 0.78
shared_prefs_size	0.005 ± 0.001	4.9 ± 0.54
execution_time	0.005 ± 0	5.7 ± 0.64
files	0.004 ± 0	7.2 ± 0.4
pull_data_size	0.003 ± 0.001	7.9 ± 0.54
files_size	0.001 ± 0	8.9 ± 0.3

Table 6: Calculated feature merits (weights) using RELIEF for 'METRICdynamics' security metric

feature	**average merit**	**average rank**
targetSdkVersion	0.064 ± 0.001	1 ± 0
sdkVersion	0.044 ± 0.003	2 ± 0 1

Table 7: Calculated feature merits (weights) using RELIEF for 'METRICsdk' security metric

In order to prove reliability of extracted security metrics, classification accuracy was tested based on metrics. Weka was used to perform these tests and the results are presented in the Figure 8 [94].

Approach	**EM**	**SVM**	**MP**	**BN**	**J48**
Accuracy on 36 features	51.49%	90.23%	92.88%	91.22%	94.03%
Accuracy on 5 metrics	52.98%	82.61%	86.92%	86.09%	90.89%

Table 8: Classification accuracy of EM (clustering), SVM, MP (Multilayer Perceptron), BN (Baessian Belief Network), J48 (C4.5 decision tree) from Weka package [94]

It can be seen that even though classification accuracy does not affected mostly, the complexity of the input data and statistical model is decreased. So it could be concluded that assumption about security metrics usage is proper. Finally, we extensively proved that selected security metrics can be used in malware detection without significant degradation of the result.

4.2.2 Digital evidence perspective

Beside developing reliable and interpretable malware detection and analysis methods, we are looking toward forensic soundness of the security metrics. In order to construct proofs, police use so-called jurisdiction data, which is going to be used further in court of law [58]. Such data can be connected to a person or some activities, ones it is proved to be derived from scientific method.

According to Daubert standard, there have to be satisfied following conditions [95]:

- *Enough data are used for testimony construction.* Data (pure technical features) are transferred to knowledge (explainable security metrics). Process is simple and repeatable.
- *Used scientific methodology is reliable, interpretable and relevant.* NF is repeatable ML approach with known small rate of potential error. Extracted rules present human-like reasoning of the knowledge.
- *Methodology was applied properly.* Process of metrics and rules extraction is automated and can be cross-validated. Therefore, no human error present in method application to given data.

It can be concluded that used security metrics fits Daubert guidelines and can be used as evidences and proof in court of law.

4.3 Malware detection process and influence of stored information

The goal of this Section is to measure and to estimate influence of stored information on malware detection. The experiments for this Section are like previous based on developed testing laboratory. To investigate such influence on malware detection, we created artificial user profile, which contains "sensitive information". Finally, automated search operations are performed over an application's data pulled from tests.

New mobile device usually contains only pre-installed applications and samples of the content like music, pictures and audio files. In order to get more clear picture of "black box" behavior in such problem, user profile is created. When emulator is starting from the scratch, both *userdata.img* and *userdata-qemu.img* images are used and filled by the information.

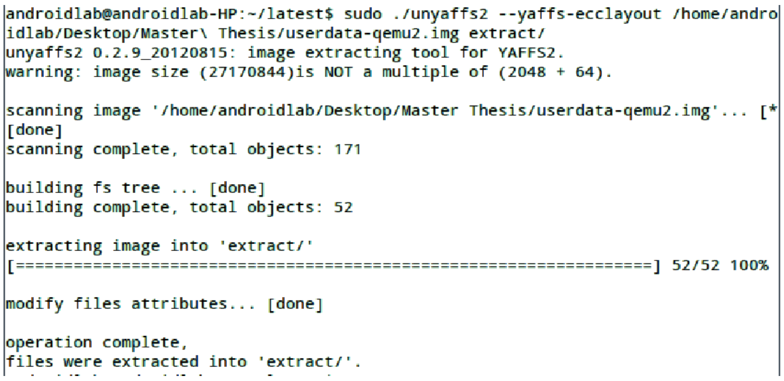
In testing laboratory, all user data are wiped and emulator is relaunched after successful application test is performed. So for created image utilization it should be connected on every startup. According to Android SDK developer documentation this operation has following format: *-initdata userdata-qemu.img* [30]. The information about the created YAFFS2[4] image is presented in the Figure 28. Also, stored available information on this image can be extracted by yaffs2 utility [96] as it is shown in the Figure 52 [97].

Automated search over all extracted information reveals only single application that was transmission user sensitive information (the function calls are presented in the Figure 29). Among all function calls, we found traces that contain information to be sent about accounts, sent SMS details and IMEI code.

Prior to *sendto()*[5] operations there also can be seen attempts to connect to remote host

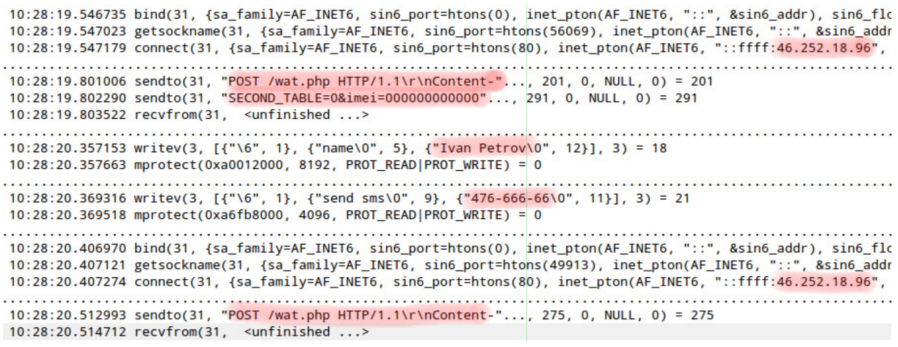
[4]A file system designed for flash memory with negated AND logic architecture

[5]A command that sends a message on a socket



```
androidlab@androidlab-HP:~/latest$ sudo ./unyaffs2 --yaffs-ecclayout /home/andro
idlab/Desktop/Master\ Thesis/userdata-qemu2.img extract/
unyaffs2 0.2.9_20120815: image extracting tool for YAFFS2.
warning: image size (27170844)is NOT a multiple of (2048 + 64).

scanning image '/home/androidlab/Desktop/Master Thesis/userdata-qemu2.img'... [*
[done]
scanning complete, total objects: 171

building fs tree ... [done]
building complete, total objects: 52

extracting image into 'extract/'
[==================================================================] 52/52 100%

modify files attributes... [done]

operation complete,
files were extracted into 'extract/'.
```

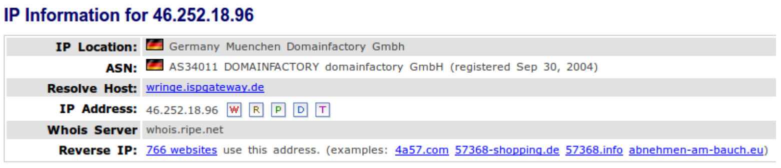
Figure 28: Information about created YAFFS2 image with user sensitive information



```
10:28:19.546735 bind(31, {sa_family=AF_INET6, sin6_port=htons(0), inet_pton(AF_INET6, "::", &sin6_addr), sin6_flc
10:28:19.547023 getsockname(31, {sa_family=AF_INET6, sin6_port=htons(56069), inet_pton(AF_INET6, "::", &sin6_addr
10:28:19.547179 connect(31, {sa_family=AF_INET6, sin6_port=htons(80), inet_pton(AF_INET6, "::ffff:46.252.18.96",
.................................................................................................................
10:28:19.801006 sendto(31, "POST /wat.php HTTP/1.1\r\nContent-"..., 201, 0, NULL, 0) = 201
10:28:19.802290 sendto(31, "SECOND_TABLE=0&imei=000000000000"..., 291, 0, NULL, 0) = 291
10:28:19.803522 recvfrom(31,  <unfinished ...>
.................................................................................................................
10:28:20.357153 writev(3, [{"\6", 1}, {"name\0", 5}, {"Ivan Petrov\0", 12}], 3) = 18
10:28:20.357663 mprotect(0xa0012000, 8192, PROT_READ|PROT_WRITE) = 0
.................................................................................................................
10:28:20.369316 writev(3, [{"\6", 1}, {"send sms\0", 9}, {"476-666-66\0", 11}], 3) = 21
10:28:20.369518 mprotect(0xa6fb8000, 4096, PROT_READ|PROT_WRITE) = 0
.................................................................................................................
10:28:20.406970 bind(31, {sa_family=AF_INET6, sin6_port=htons(0), inet_pton(AF_INET6, "::", &sin6_addr), sin6_flc
10:28:20.407121 getsockname(31, {sa_family=AF_INET6, sin6_port=htons(49913), inet_pton(AF_INET6, "::", &sin6_addr
10:28:20.407274 connect(31, {sa_family=AF_INET6, sin6_port=htons(80), inet_pton(AF_INET6, "::ffff:46.252.18.96",
.................................................................................................................
10:28:20.512993 sendto(31, "POST /wat.php HTTP/1.1\r\nContent-"..., 275, 0, NULL, 0) = 275
10:28:20.514712 recvfrom(31,  <unfinished ...>
```

Figure 29: Extracted list of function calls that contains user sensitive information

46.252.18.96 (see the Figure 30 for whois details). This IP address is located in Germany and is resolved to few hundred websites like "real estate bests services". Such sites likely can be fraudulent resources. Also we notice that this website on this IP address should contain *wat.php* script, which receives POST request with sensitive information from the devices.



IP Information for 46.252.18.96

IP Location:	Germany Muenchen Domainfactory Gmbh
ASN:	AS34011 DOMAINFACTORY domainfactory GmbH (registered Sep 30, 2004)
Resolve Host:	wringe.ispgateway.de
IP Address:	46.252.18.96 W R P D T
Whois Server	whois.ripe.net
Reverse IP:	766 websites use this address. (examples: 4a57.com 57368-shopping.de 57368.info abnehmen-am-bauch.eu)

Figure 30: Whois response for the IP address 46.252.18.96

As it can be concluded, user profiles usage in automates analysis is perspective and powerful area of malware analysis. This is because manual reverse engineering requires much more time and human efforts to extract the same data. After searching the Internet we found examination of the same malware sample with help of not automated testing box solution DroidBox [98, 99]. Results of our test and test in found report are almost identical: sensitive information leakage. However, our test approach allows to extract more specific details from pulled data about information leakage based on user profile.

Summary

To summarize, user sensitive information experiments, we state that it has influence on the malware detection. It was shown earlier in this Section that we were able to intercept sensitive information that applications try to sent to external sources. This can be crucial decisive factor in malware detection. However, very few evidences that sent in open text format were found. Hence, more deep tests and examination is required for revealing such information sent in encrypted or modified format. In conclusion, the achieved during experiments results are impressive and were not considered before in such perspective in literature. Furthermore, user profiles and user behavioral information can be used as one of the malware detection key factors.

4.4 Significance and reliability of malware detection

Due to growing complexity and range of possible attacks on mobile platform, pure statical analysis is no longer reliable. Recent malware tend to hide malicious actions by means of obfuscation, encryption and downloaded payload execution [12, 100]. It could happen that application just carry an address of a malware that it loads on the devices while executing. Therefore, static and dynamic tests of a "black box" application have to be performed. This gives not only comprehensive picture of the application, yet allows to detect privacy threats more reliably.

One can consider static test as an analysis of the application without using it according to its main purpose. At the same time dynamic test has aim to execute application as intended in protected environment. For the experiment setup we are using re-developed testing laboratory for mobile malware designed by author previously [16]. It has following predefined by us testing routine for each application:

1. *Static phase*
   - 1.1. Installation package structure analysis;
   - 1.2. Java code extraction;
   - 1.3. Resources and assets processing.
2. *Dynamic phase*
   - 2.1. Simulation of 500 random user actions with help of UI[6]/Application Exerciser Monkey, which is provided by Android SDK [30];
   - 2.2. Stored information processing and extracting from an emulated virtual mobile device;
   - 2.3. Tracing of all function calls during application execution by means of strace utility [72];
   - 2.4. Applications traffic intercepting with tcpdump [101]. Further extraction from raw cap dump into XML[7] structure is performed by Tshark program from Wireshark package [102];

[6]User Interface

[7]eXtensible Markup Language - common markup language, which is often used for storing different data structures or parameters

2.5. User profile simulation by means of prepared userqemu-data.img image file;

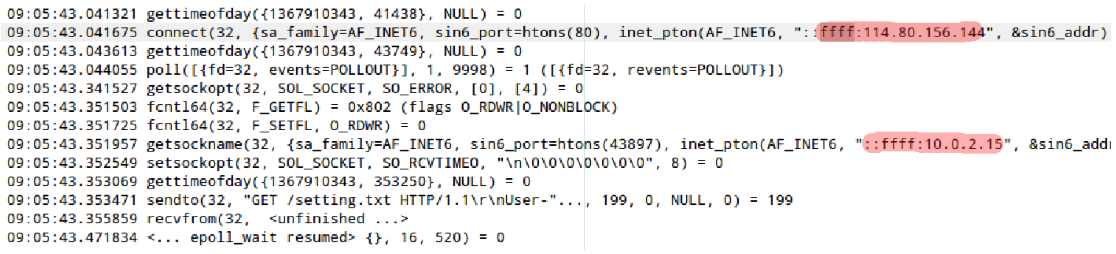
2.6. Resource usage tracking (CPU utilization, Virtual memory set size, Resident memory set size and number of launched threads [61]) from available information in top and ps Linux programs.

4.4.1 Results of automated analysis

Manual reverse engineering of application based on expert knowledge can be considered as the most informative analysis [103, 104, 105]. This is because of specific designing issues of each particular application and comprehensive awareness of an expert. Despite this fact that automated analysis with well designed testing routing could provide efficient results.

The author would like to emphasize on capabilities of the automated applications processing. By execution designed previously testing routine, there were collected feature vectors for each application. Upon manual reviewing of the extracted data, there should be mentioned few valuable artifacts.

Traced function calls obtained by Strace utility for a malware is presented in the Figure 31 [72]. As it could be seen, there are attempts to connect to external IP address 114.80.156.144 and get name of the virtual device's local address IP 10.0.2.15. Both addresses are IPv4 and are expressed in IPv6 format.



```
09:05:43.041321 gettimeofday({1367910343, 41438}, NULL) = 0
09:05:43.041675 connect(32, {sa_family=AF_INET6, sin6_port=htons(80), inet_pton(AF_INET6, "::ffff:114.80.156.144", &sin6_addr)
09:05:43.043613 gettimeofday({1367910343, 43749}, NULL) = 0
09:05:43.044055 poll([{fd=32, events=POLLOUT}], 1, 9998) = 1 ([{fd=32, revents=POLLOUT}])
09:05:43.341527 getsockopt(32, SOL_SOCKET, SO_ERROR, [0], [4]) = 0
09:05:43.351503 fcntl64(32, F_GETFL) = 0x802 (flags O_RDWR|O_NONBLOCK)
09:05:43.351725 fcntl64(32, F_SETFL, O_RDWR) = 0
09:05:43.351957 getsockname(32, {sa_family=AF_INET6, sin6_port=htons(43897), inet_pton(AF_INET6, "::ffff:10.0.2.15", &sin6_add
09:05:43.352549 setsockopt(32, SOL_SOCKET, SO_RCVTIMEO, "\n\0\0\0\0\0\0\0", 8) = 0
09:05:43.353069 gettimeofday({1367910343, 353250}, NULL) = 0
09:05:43.353471 sendto(32, "GET /setting.txt HTTP/1.1\r\nUser-"..., 199, 0, NULL, 0) = 199
09:05:43.355859 recvfrom(32,  <unfinished ...>
09:05:43.471834 <... epoll_wait resumed> {}, 16, 520) = 0
```

Figure 31: Function calls traces that includes attempts to connect to the external IP address 114.80.156.144

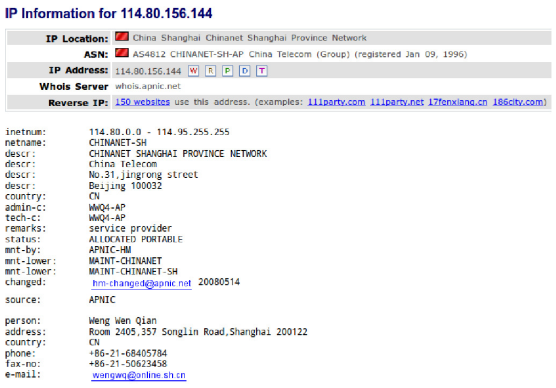
For examination purposes the author checked that external IP address by publicly available whois service [106] and the output is presented in the Figure 32. One found that this address belongs to China and there hundreds of websites domains connected to this IP.

However, most of the websites are either blocked or protected from direct browser access (see Figure 33). The error 404[8] appeared often, which makes think that there could be located some concealed scripts that accept only specific GET/POST requests[9].

Now we look more deeply into another domain of dynamic tests, network traffic analysis. Screenshot of the Wireshark GUI depicting obtained traffic dump is shown in the Figure 34 [102]. There are several 3-way handshake TCP requests to external IP addresses. Those requests are integral part of HTTP protocol[10] request. As no human interaction were taken during tests execution, these requests were generated by application itself.

[8]HTTP 404 or not found error implies that server is able to receive requests, but requested resources are unavailable or absent

[9]Two commonly used methods for client-server communication. GET sends data using address url of the page (or

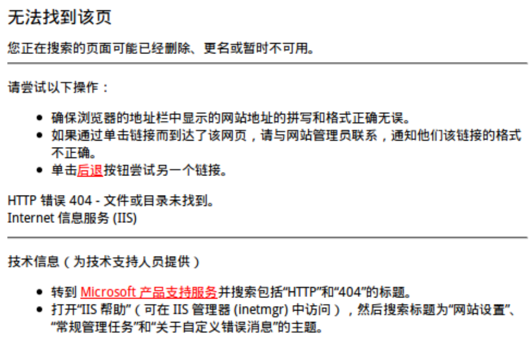


IP Information for 114.80.156.144

IP Location:	China Shanghai Chinanet Shanghai Province Network
ASN:	AS4812 CHINANET-SH-AP China Telecom (Group) (registered Jan 09, 1996)
IP Address:	114.80.156.144 W R P D T
Whois Server	whois.apnic.net
Reverse IP:	150 websites use this address. (examples: 111party.com 111party.net 17fenxiang.cn 186city.com)

```
inetnum:        114.80.0.0 - 114.95.255.255
netname:        CHINANET-SH
descr:          CHINANET SHANGHAI PROVINCE NETWORK
descr:          China Telecom
descr:          No.31,jingrong street
descr:          Beijing 100032
country:        CN
admin-c:        WWQ4-AP
tech-c:         WWQ4-AP
remarks:        service provider
status:         ALLOCATED PORTABLE
mnt-by:         APNIC-HM
mnt-lower:      MAINT-CHINANET
mnt-lower:      MAINT-CHINANET-SH
changed:        hm-changed@apnic.net 20080514

source:         APNIC

person:         Weng Wen Qian
address:        Room 2405,357 Songlin Road,Shanghai 200122
country:        CN
phone:          +86-21-68405784
fax-no:         +86-21-50623458
e-mail:         wengwq@online.sh.cn
```

Figure 32: Whois response for the IP address 114.80.156.144



无法找到该页

您正在搜索的页面可能已经删除、更名或暂时不可用。

请尝试以下操作：

- 确保浏览器的地址栏中显示的网站地址的拼写和格式正确无误。
- 如果通过单击链接而到达了该网页，请与网站管理员联系，通知他们该链接的格式不正确。
- 单击后退按钮尝试另一个链接。

HTTP 错误 404 - 文件或目录未找到。
Internet 信息服务 (IIS)

技术信息（为技术支持人员提供）

- 转到 Microsoft 产品支持服务并搜索包括“HTTP”和“404”的标题。
- 打开“IIS 帮助”（可在 IIS 管理器 (inetmgr) 中访问），然后搜索标题为“网站设置”、“常规管理任务”和“关于自定义错误消息”的主题。

Figure 33: The information returned to the client when visited 114.80.156.144

Whois[11] is of one of the IP addresses states that it is located in USA and there are few thousands of reverse connected domain.

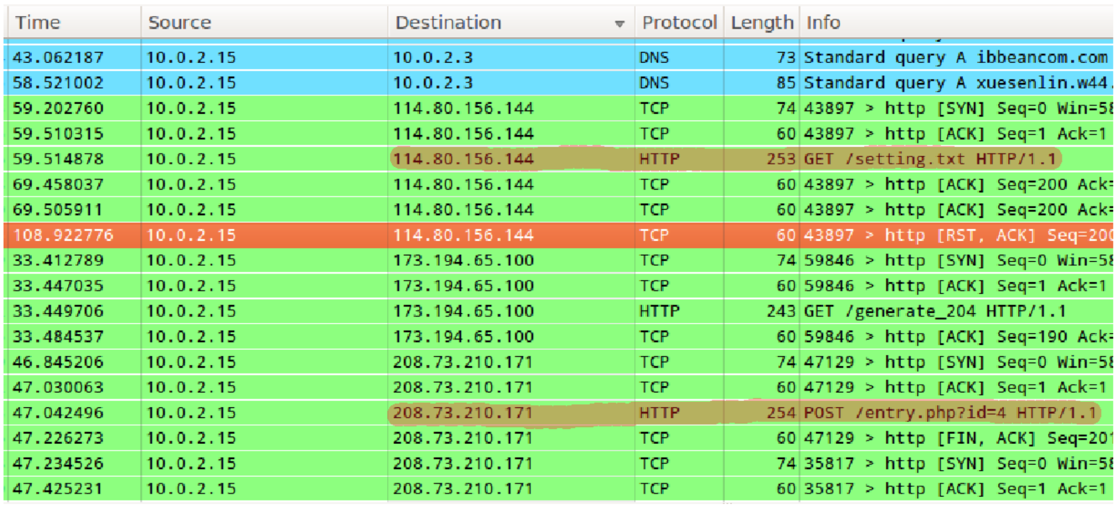
By following couple of registered websites domains, it brings us to the same page presented in the Figure 36. Information on the page recommends us to participate in an arbitrary survey. Even if some answers were given falsely on purpose, we got a "unique chance" to buy a possibility to win IPhone 5 after several similar surveys during one month. Moreover, it seems that we need to sent SMS to a top-rate chargeable services that finally cost us around 100$. Finally, one can also notice that such web site has Geo targeting that provides translated content based on location of a client. Basically, from our experience in information security area, one can say that this is typical fraud page.

One can conclude that it was shown how information from both static and dynamic tests are crucial for malware detection. All the presented data were extracted without any human participation. So, static and dynamic test can be treated as reliable source of information for automated

header of the request), while POST sends data through concealed variables (body of the request)

[10]Hyper Text Transfer Protocol - server-client interaction foundation for WWW

[11]Request for getting human-readable information about domain name, particularly dates, name servers and owner



Time	Source	Destination	Protocol	Length	Info
43.062187	10.0.2.15	10.0.2.3	DNS	73	Standard query A ibbeancom.com
58.521002	10.0.2.15	10.0.2.3	DNS	85	Standard query A xuesenlin.w44.
59.202760	10.0.2.15	114.80.156.144	TCP	74	43897 > http [SYN] Seq=0 Win=5
59.510315	10.0.2.15	114.80.156.144	TCP	60	43897 > http [ACK] Seq=1 Ack=1
59.514878	10.0.2.15	114.80.156.144	HTTP	253	GET /setting.txt HTTP/1.1
69.458037	10.0.2.15	114.80.156.144	TCP	60	43897 > http [ACK] Seq=200 Ack=
69.505911	10.0.2.15	114.80.156.144	TCP	60	43897 > http [ACK] Seq=200 Ack=
108.922776	10.0.2.15	114.80.156.144	TCP	60	43897 > http [RST, ACK] Seq=20
33.412789	10.0.2.15	173.194.65.100	TCP	74	59846 > http [SYN] Seq=0 Win=5
33.447035	10.0.2.15	173.194.65.100	TCP	60	59846 > http [ACK] Seq=1 Ack=1
33.449706	10.0.2.15	173.194.65.100	HTTP	243	GET /generate_204 HTTP/1.1
33.484537	10.0.2.15	173.194.65.100	TCP	60	59846 > http [ACK] Seq=190 Ack=
46.845206	10.0.2.15	208.73.210.171	TCP	74	47129 > http [SYN] Seq=0 Win=5
47.030063	10.0.2.15	208.73.210.171	TCP	60	47129 > http [ACK] Seq=1 Ack=1
47.042496	10.0.2.15	208.73.210.171	HTTP	254	POST /entry.php?id=4 HTTP/1.1
47.226273	10.0.2.15	208.73.210.171	TCP	60	47129 > http [FIN, ACK] Seq=20
47.234526	10.0.2.15	208.73.210.171	TCP	74	35817 > http [SYN] Seq=0 Win=5
47.425231	10.0.2.15	208.73.210.171	TCP	60	35817 > http [ACK] Seq=1 Ack=1

Figure 34: Example of HTTP requests to an external IP address using POST and GET

malware detection. Moreover, functionality of a Android SDK emulator grants automated analysis with good trade-off between complexity of performed tests and spent time. Finally, it has to be mentioned that fraud intentions spreads not only on general PC, yet already we found evidences on mobile devices.

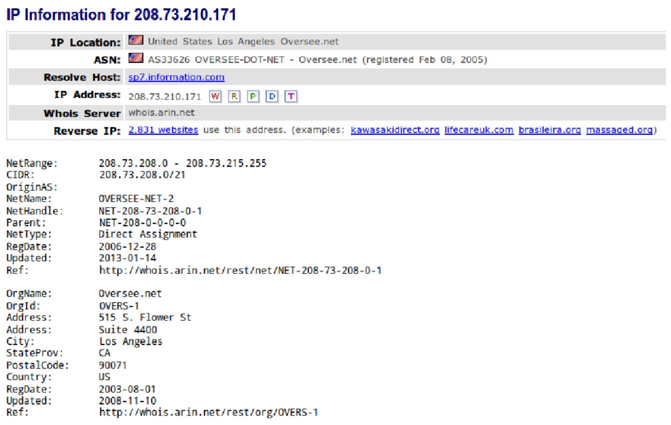
4.4.2 On-line learning perspective

New malware samples and variations of already existing appear everyday on Mobile Markets. This results in growing amount of applications from which statistical features are extracted. Typical off-line ML based malware detection solutions require much computational resources and time to get all applications processed. Therefore, we are looking towards adaptable malware detection. Statistical properties of the model can be easily adjusted during adding new applications. This fact makes it possible to build on-line learning system that will partially retrain a statistical model after changing applications dataset. Most off-line learning approaches require a lot of time and are inefficient in dealing with a huge data storage (application's markets).

Proposed detection mechanism is based on ANN that has an ability to be trained continuously. Thus, features from a new application can be extracted on-fly and as rules are adjusted immediately with a smallest delay. By utilizing this concept, we are projecting application classification challenge into data streams mining, treating new coming applications as a stream. This is a new and developing area of computational intelligence that was extensively studied in the research [107]. In conclusion, one can say that such malware detection adaptive system will be more flexible that existing regularly updated signature-based AV software.

4.5 Fuzzy rules for malware detection

The aim of the Section is to present and discuss results of linguistic rules application in malware detection. To perform comprehensive testing we created following procedure (see the Figure 37) based on developed earlier theoretical justification. Initially, all possible rules are constructed



IP Information for 208.73.210.171

IP Location:	United States Los Angeles Oversee.net
ASN:	AS33626 OVERSEE-DOT-NET - Oversee.net (registered Feb 08, 2005)
Resolve Host:	sp7.information.com
IP Address:	208.73.210.171 W R P D T
Whois Server	whois.arin.net
Reverse IP:	2,831 websites use this address. (examples: kawasakidirect.org lifecareuk.com brasileira.org massaged.org)

```
NetRange:       208.73.208.0 - 208.73.215.255
CIDR:           208.73.208.0/21
OriginAS:
NetName:        OVERSEE-NET-2
NetHandle:      NET-208-73-208-0-1
Parent:         NET-208-0-0-0-0
NetType:        Direct Assignment
RegDate:        2006-12-28
Updated:        2013-01-14
Ref:            http://whois.arin.net/rest/net/NET-208-73-208-0-1

OrgName:        Oversee.net
OrgId:          OVERS-1
Address:        515 S. Flower St
Address:        Suite 4400
City:           Los Angeles
StateProv:      CA
PostalCode:     90071
Country:        US
RegDate:        2003-08-01
Updated:        2008-11-10
Ref:            http://whois.arin.net/rest/org/OVERS-1
```

Figure 35: Whois response for the IP address 208.73.210.171

based on extracted metrics on preprocessing step. Then, ANN is used for rules construction and tuning based on class with highest weight in learned model.

Extracted in this way rules can be treated as malware patterns with fuzzy nature. What is interesting about such rules is flexibility in training process. One can say that important downside of the classic signature-based solutions is the need for regeneration of signature when one of the characteristics of malware is changed. Linguistic terms in fuzzy rules compose an abstraction layer, which use statistical parameters of previously extracted metrics. Therefore, there is no need to change rules significantly, yet only statistical parameters behind them. It can be seen that the parameters are purely based on constructed metrics.

4.5.1 Evaluation of classification process

Proposed method is not just provide a simple signature matching, yet also linguistic rules extraction. Compare to *NNge*[12] and *Decision Tree* classification methods in Weka [94], our method does not produce hardly-understandable rules based only on discrete numerical values. Instead, fuzzy IF-THEN rules are extracted. So, the prerequisites for mobile malware success one can define as following extracted rules in the Figure 38.

Whole process leads to determination of rules with bigger membership value of input metric vector for an application to be classified. Used security metrics are accessible from emulated and native mobile device environment. Furthermore, such method has moderate computational complexity that does not consume much resources.

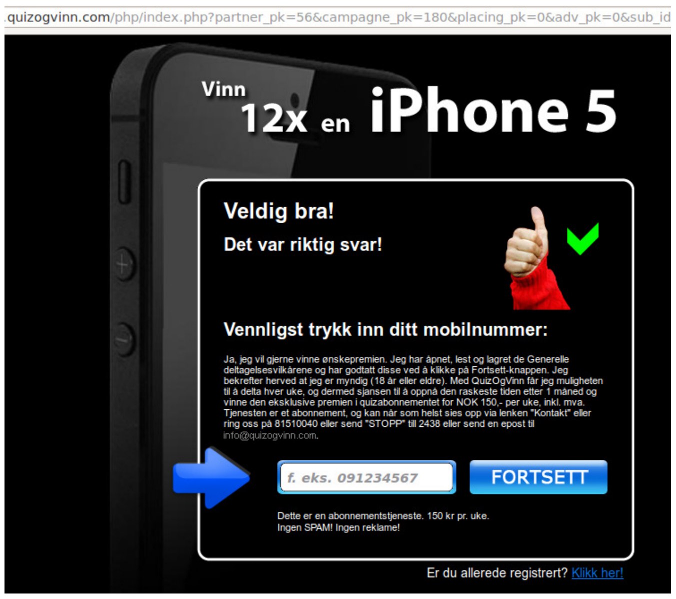
The developed system for fuzzy rules extraction has following Data Flow Diagram[13]

4.5.2 Accuracy of classification

Malware detection is a binary classification problem. That is why confusion matrix 9 is used in performance evaluation of classification using extracted rules [10]. This matrix shows information about amount of actual and predicted classes (benign and malicious) So, classification

[12]Algorithm based on Nearest-Neighbor principle, which is using hyperrectangles that can be viewed as if-then rules

[13]Represents flow of the different data through the developed system, particularly revealing processing steps


quizogvinn.com/php/index.php?partner_pk=56&campagne_pk=180&placing_pk=0&adv_pk=0&sub_id

Vinn 12x en iPhone 5

Veldig bra!

Det var riktig svar!

Vennligst trykk inn ditt mobilnummer:

Ja, jeg vil gjerne vinne ønskepremien. Jeg har åpnet, lest og lagret de Generelle deltagelsesvilkårene og har godtatt disse ved å klikke på Fortsett-knappen. Jeg bekrefter herved at jeg er myndig (18 år eller eldre). Med QuizOgVinn får jeg muligheten til å delta hver uke, og dermed sjansen til å oppnå den raskeste tiden etter 1 måned og vinne den eksklusive premien i quizabonnementet for NOK 150,- per uke, inkl. mva. Tjenesten er et abonnement, og kan når som helst sies opp via lenken "Kontakt" eller ring oss på 81510040 eller send "STOPP" till 2438 eller send en epost til info@quizogvinn.com.

Dette er en abonnementstjeneste. 150 kr pr. uke.
Ingen SPAM! Ingen reklame!

Er du allerede registrert? Klikk her!

Figure 36: Advertising website that is located on the found IP address

accuracy [10] can be estimated as following

$$Accuracy = \frac{Nc}{N} \cdot 100\% \tag{4.1}$$

Where Nc - is an amount of correctly classified applications and N - is a total amount.

	benign (predicted)	**malicious** (predicted)
benign (actual)	200	51
malicious (actual)	92	261

Table 9: Confusion Matrix for malware classification problem

Proposed method shows unified accuracy 76.32% (using the Equation 4.1) on security metrics dataset for both malicious and benign applications. If we compare the Table 8, it can be seen that the accuracy of proposed method is less than the SVM classifier. However, this difference is 6-7%, while SVM has much more complex nature and uninterpreted statistical model. Further, accuracy can be increased by applying more advanced filtering methods in order to exclude noise and mistaken data from the features dataset.

Summary

In overall mobile malware detection process using fuzzy rules can be applied in this process. The goal was to find how reliable such rules are and what is the accuracy on real-world data. Results considerably indicates that theory behind security metrics and their construction were proper. From this point we can see that our method is able do detect human-understandable and computational efficient malware detection. During experiments, we notice that benign applications are more selective in mobile devices. It means that they are not launched if a device is not

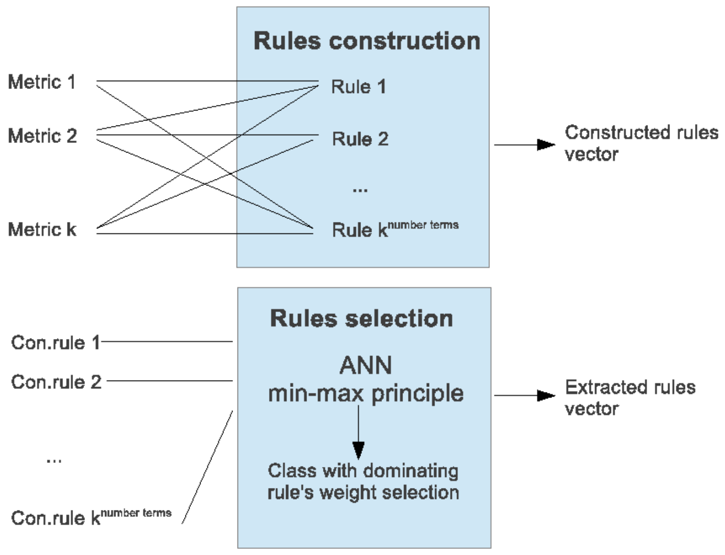
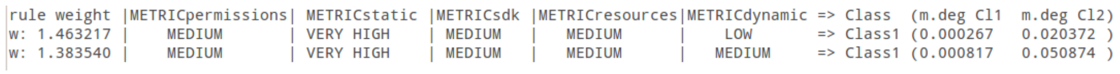


Figure 37: Scheme of rules extraction for malware classification



```
rule weight |METRICpermissions| METRICstatic |METRICsdk |METRICresources|METRICdynamic => Class  (m.deg Cl1  m.deg Cl2)
w: 1.463217 |    MEDIUM       | VERY HIGH    | MEDIUM   |    MEDIUM     |     LOW      => Class1 (0.000267   0.020372 )
w: 1.383540 |    MEDIUM       | VERY HIGH    | MEDIUM   |    MEDIUM     |   MEDIUM     => Class1 (0.000817   0.050874 )
```

Figure 38: Example of extracted fuzzy rules for malware detection after processing malicious and benign application

suitable or has different configuration. From another perspective, malicious applications have simpler requirements and can be run on almost all devices. Furthermore, it was found that both classes of application has significant statistical differences in collected raw features.

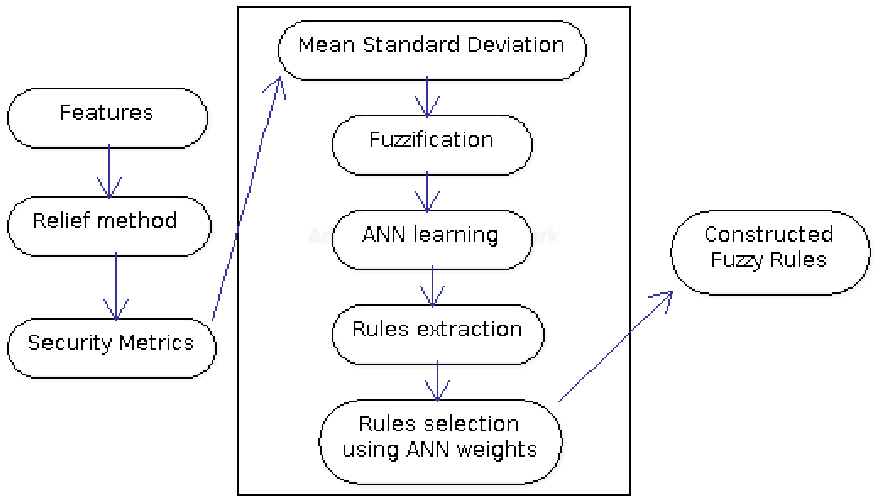


Figure 39: Data Flow Diagram of the rule-construction system using, drawn in ArgoUML

5 Discussions

This chapter provides comprehensive discussions of theoretical and practical aspects of this master thesis.

5.1 Data and Experiments

In this Section the used data, developed method and experimental considerations will be discussed.

5.1.1 Methodology

The methodology was formulated after extensive study of related research literature, presented in the Chapter 2. The work is targeted on creating pro-active malware detection and analysis system using machine learning. The advantage of built up methodology is an application in automated analysis of large datasets of applications like software markets or commercial collections. As it was found, automated analysis reveals sometimes more useful information for malware detection in artifacts than manual reverse engineering can do. As continuation of the research, on-site defense can be developed for usage on mobile devices.

The drawback of proposed methodology is that it requires expert knowledge in proper construction of security metrics from statistical analysis. As an alternative to this, multidimensional optimization can be utilized in order to detect proper weights for each of the features related to security metric's domain.

In the thesis, our goal was to concentrate on developing forensics soundness detection method. The NF procedure was chosen in order to obtain human-understandable detection rules with negligible detection error. Completed further tests on other ML approaches like SVM shown that the methodology was chosen properly.

5.1.2 Dataset

One can state that there were no previously collected datasets for applications on mobile platforms. Construction of generalized dataset was a significant challenge not just because of difficulties in gathering malware for mobile devices, yet also because of making sure that they are malicious. Eventually, we gathered around thousand of both types of applications. Due to ethical and legal considerations, it was decided to use publicly available sources like this [108] for collecting the applications. All the malicious applications were cross-validated with help of free and open-source ClamAV[1] AV software. The dataset consists of all popular applications with most recent versions. It is suitable for future researches in the field and malware detection benchmarks because the amounts of benign and malicious applications are balanced.

In order to answer the research questions, we were using two more datasets. Execution of dynamic and static automated tests brought a first dataset consisted from an extracted features

[1]http://www.clamav.net/lang/en/

for each application. Gradually, using RELIEF method was extracted the second dataset with security metrics for each applications. These two datasets are suitable not for testing the reliability of malware detection process, yet also for extracting decision rules.

5.1.3 Complexity

It was shown in the Chapter 4 that implemented fuzzy-rule construction module achieves impressive time performance while learning from metrics dataset. With help of parallel execution the ANN learning process and the rule-construction processes cause delay around 5-6 seconds. This value is much less than collecting and updating signatures datasets for classical signature-based AV software. It can be considered as one of the achievements of the research.

For obtaining more scientifically sound measures of complexity, following complexity metrics are evaluated [109]:

- *Algorithmic complexity*

  It is analyzed how amount of features and linguistic terms affects complexity of the rule-construction algorithm. While executing, each atomic operation on individual input data sample requires several processor instructions, which amount depends on the operation. While learning, ANN executes several such operations on each of the dimension (attribute) in the input data sample. ANN learning represents a NP-complete problem that can be solved in a polynomial time. The dependency on input data is linear, while dependency on input data dimensionality is polynomial.

  The amount of necessary operational step for ANN learning and rules extraction can be defined as following function:

  $$f_1(n) = L \cdot N \cdot (n+1) \cdot M^n \tag{5.1}$$

  N - amount of the input data samples, n - dimensionality of the input data, M - number of terms in each linguistic variable (security metric), L - amount of epochs in ANN learning phase.

  The amount of necessary operations for rules selection is defined by function:

  $$f_2(n) = 2 \cdot M^n \cdot \log(N) \tag{5.2}$$

  The total amount of steps in ANN learning and rules extraction can be estimated as following:

  $$f(n) = f_1(n) + f_2(n) = L \cdot N \cdot (n+1) \cdot M^n + 2 \cdot M^n \cdot \log(N) \tag{5.3}$$

  The result roughly best-case algorithmic complexity of the rules construction process is:

  $$A(n) = O(n \cdot M^n) \tag{5.4}$$

  Apparently, the computational complexity is growing when the dimensionality of the input data is increasing.

- *Time complexity*

  Considering implementation and practical usage of the method, the problem of runtime complexity evaluation arises. The rules extraction algorithm is treated as sequential algorithm. While applying multiprocessor optimization, the time complexity can be decreased because of execution multiple tasks in parallel. In our case this is done by learning and evaluating parameters for each rule in parallel. Other operations have to be executed only serially. If we denote p as an amount of execution threads and modify the expression 5.4, then the algorithmic complexity per each thread are as following:

  $$f(n) = \frac{1}{p}\left(L \cdot N \cdot (n+1) \cdot M^{n}\right) + 2 \cdot M^{n} \cdot \log(N) \tag{5.5}$$

  Final best-case runtime complexity of rules construction depends on the amount of thread and have following equation:

  $$T(n) = O\left(n \cdot \frac{M^{n}}{p}\right) \tag{5.6}$$

  The best time complexity can be obtained if number of execution threads is equal to total number of constructed fuzzy rules. This is because rules parameters evaluation is the most inner task in ANN learning. Theoretically, the speed up while using parallel computing can be equal to number of execution threads. However, in real case this number depends on data transferring overheads and therefore it is less or almost equal to amount of threads in best case scenario. We denote faction of total operations that should be executed only serially as α:

  $$\alpha = \frac{2 \cdot M^{n} \cdot \log(N)}{L \cdot N \cdot (n+1) \cdot M^{n} + 2 \cdot M^{n} \cdot \log(N)} \tag{5.7}$$

  Then, according to Amdahl's law [110], the maximal possible speedup of the parallel algorithm will be following:

  $$S_p = \frac{1}{\alpha + \frac{1-\alpha}{p}} \tag{5.8}$$

  The delays are caused by transferring data from CPU to GPU and back. However, GPU can provide more threads to execute threads than CPU, which is a big advantage of its utilization.

- *Space complexity*

  Space complexity of the method is an important factor in evaluation of performance. We can defined that amount of memory blocks necessary for storing the rules and corresponding weights:

  $$f(n) = 2 \cdot M^{n} \tag{5.9}$$

The total space complexity has exponential dependency on input data dimensionality as it is depicted in the Equation 5.10.

$$S(n) = O(M^n) \tag{5.10}$$

One can summarize that the algorithm has optimal time-space-tradeoff that does not require additional storage space except RAM memory. Apparently, it causes less possible execution delays.

5.1.4 Robustness & Reliability

Under the robustness term we understand the ability of the malware detection method to withstand different obfuscation and protection techniques inside malware. While testing other ML methods on the acquired dataset, it was shown that the results are not so different from the proposed methods. It means that the method is able to detect malware by extracted metrics even if its payload is hidden.

Reliability of the method implies not only proper detection of malware sample, yet also understanding by human brain. In this case extracted rules contain readable linguistic terms, which conceals complex numerical parameters. By reading fuzzy rules used for detection, common user can estimate the prerequisites for successful malware execution. Moreover, it increases user awareness about information security more than average AV software.

5.2 Implementation Architecture

Testing laboratory for mobile malware developed by author previously is re-build and re-coded significantly for using in this Master Thesis [16]. Basically, there exist a problem in testing mobile applications and extracting corresponding security-related features. This is because of performance limitations and diversity of modern mobile platforms configuration. Although, mobile devices are cheap and widespread, it is infeasible task to test dataset of collected application on one or several devices in a short time range. Devices from different producers has different Android API level, installed applications and hardware specification

For undertaking all the mentioned challenges, it was completely rebuild *Testing* and *Analysis* functionality of previously implemented by author testing laboratory. This leads to fulfilling following testing laboratory requirements: feature extraction and selection, security metrics extraction, rules construction.

Experimental setup was performed on three computers with following hardware details:

1. *PC 1*

   - CPU: AMD Athlon 4400+ X2, 2.3 Ghz (1M L2 cache 2000MHz HT)
   - RAM: 4GB, DDR2 667Mhz
   - Hard Disk Drive (HDD): Seagate 250GB, 16MB cache, SATA2
   - GPU: MSI GeForce N210, Core 589MHz, 1GB GDR3 64bit, CUDA 1.2

2. *PC 2*

  - CPU: Intel Core2 6300 1.86GHz x2 (2M L2 cache, 1066 MHz FSB)
  - RAM: 2GB, DDR2 533Mhz
  - HDD: Seagate 80GB, 8MB cache, SATA2
  - GPU: Intel GMA x3000, 256MB, 64bit

3. *PC 3*
   - CPU: Intel Core2 Duo T8100 2.10GHz x2 (3M L2 cache, 800MHz FSB)
   - RAM: 3GB, DDR2 667Mhz
   - HDD: Hitachi 120GB, 8MB cache, SATA1
   - GPU: Intel GMA x3100, 384MB, 64bit

Additionally, there were installed next *software:*

- OS: Ubuntu 12.04 64bit
- Android SDK v22
- Apache Web Server v2.2.22-1ubuntu1.3
- PHP v5.3.10-1ubuntu3.6
- MySQL v5.5.31-0ubuntu0.12.04.1
- WireShark v1.6.7-1
- tcpdump v4.2.1-1ubuntu2
- g++ v4:4.6.3-1ubuntu5
- OpenMP v4.6.3-1ubuntu5
- Thrust v1.6.0 (only for PC1)
- CUDA v5.0 (only for PC1)

The source code was developed (more detailed description and source code listings can be found in the Appendix F) during work on master thesis and has following structure:

- *androidlab/test_cycle.php* - Implementation of testing cycle as a part of the testing laboratory [16] in PHP using MVC[2]. This part is able to setup configuration of the static and dynamic tests, launch automated testing of multiple applications using various emulator parameters.
- *androidlab/analysis.php* - Test data analysis and feature extraction functionality was implemented in PHP using MVC.

[2]Model-View-Controller, a template for application development that helps to separate data, view and functionality. MVC provides flexibility and more-understandable implementation in building a software carcase

- *NeuralNetwork/main.cpp* - Perform ANN learning based on the train data, fuzzy-rule extraction for binary classification problem using NF approach.
- *NeuralNetwork/main.cu* - A variant of main.cpp, which is optimized for multiprocessing and implemented with help of CUDA native code.

Limitations

Within given working time constraints there were defined implementation and usage limitations. Under these limitations it was possible to implement, test and analyze data for successful answering on defined research questions. The quality and quantity boundaries are the following:

- Only Android with version at least 4.0 was used for the experiments. However, there is a support for older platform versions (like Android 2.2 and Android 2.3.3), yet without possibility of taking the screenshots.
- Emulated Android device was not rooted in order to achieve real case scenario. According to our observations, some of the malware requires rooting operation while executed.
- It was created and developed only single user profile because of time limitations and negligible effect on malware detection process.
- The testing laboratory is able to extract logs of installation, UI testing, launch and uninstallation processes. In feature extraction was used only application launch log (An example can be seen in the Appendix D).
- The maximal amount of APK files processed at once in App Management sub-system is limited to 500.
- Testing of large amount (thousands) of applications can reach standard 30 seconds timeout of PHP script execution on Apache server.
- Despite the fact that all checks of necessary system components and software are implemented in the testing laboratory, some parts of the LAMP server[3] can be missing.
- C++ code was compiled with 64bit Linux architecture support, which can cause problems while executing on old 32bit machines. It is possible to filter out this problem by means X86_64 emulation such that QEMU [111].
- Amount of ANN learning epochs was chosen empirically as 1000. It means that the performance of the method does not improve significantly with bigger amount of epochs.
- Implemented module has support only 3 and 5 terms linguistic variables, which is enough for the experiments
- Input labeled data for rule-construction module can have only single-precision real-values (type 'float') of features/metrics with that allows to store no more than 7 digits of precision.

[3] `http://help.ubuntu.ru/wiki/lamp`

- OpenMP support has effect only in case if the CPU has support of the multi-threading features (by means of several processors, cores or hardware threads. Otherwise, effect of the multi-threading support will be negligible or even negative because of sequential execution of the sub-task
- CUDA experimental implementation is targeted only on modern Nvidia cards with support of 32bit single-precision floating-point numbers and operations support (CUDA version has to be at least 1.2).

In order to achieve consistency, elegance and readability, all source code was implemented with help of the next guidance: PHP Zend Framework[4] and C++[5] coding conventions. By using common programming style, the program part is readable and understandable for everybody from the area. Moreover, the maintenance and further development will consume less time on code reading and functionality understanding.

5.2.1 Application testing and feature extraction

Because of rebuilding of the testing laboratory, it becomes possible to launch testing from the scratch and perform wider dynamic tests. User profile was connected to study influence of stored information. Network traffic is captured and parsed into XML format. User and system function calls are also collected. Experiment that consists of static and dynamic testing of applications was conducted as it is described in the Chapter 3. All developed PHP code was optimized and profiled in order to achieve the best possible performance.

Android SDK [30] offers system images with Advanced RISC Machine (ARM). However, ARM instructions are quite different from personal computer architectures like x86 and extended x86-64 [112]. That is why emulator execution and round test takes much time. In order to deal with this, it was decided to increase performance of the testing laboratory by means of three main steps:

1. *x86-64 system architecture*

   The personal computer, which is used for testing has x86-64 architecture. It is logical to use native x86 instructions instead of ARM instructions emulation. Android Software Development Kit (SDK) provides x86 system image for last version of Android (4.2.2). Usage of 64 bit extension gives more allocatable address space and performance improvements [113].

2. *VM acceleration*

   Basically, running Android emulator [30] on x86-64 architecture require virtualization of emulated host. In order to improve total performance, kernel-based Virtual Machine (VM) (KVM) was installed on Linux. KVM is a program solution, which supports hardware virtualization (Intel VT or AMD-V technologies) [111]. According to Android Developers [30], it is possible to turn on support of KVM in Android virtual machine by using parameters: -*qemu -m 512 -enable-kvm*. Virtualization provides more protected and isolated environment on host machine, which allows to execute several VM at the same time.

[4]`http://framework.zend.com/manual/1.12/en/coding-standard.html`
[5]`http://www.c-xx.com/ccc/ccc.php`

3. *GPU acceleration*

   To compensate complex calculations for drawing visual elements in emulator screen, GPU acceleration should be used. In 2012, Google announced that latest Android emulator [30] versions support graphical acceleration by means of host graphical card utilization. This leads to faster redrawing and execution of visual effects on emulator. In order to use such acceleration, launch parameter *-gpu on* is needed [30].

Taking into consideration all mentioned adjustments in configuration, one can state that the speed of emulator execution has increased considerably. Specific performance details of obtained configuration is discussed further.

5.2.2 Advantage of virtual environment usage

During work on the testing laboratory, we obtained valuable experience while deployed virtualized environment. It was noticed considerable time reduction on launch in comparing not only to emulated ARM, yet also to real mobile devices. In contrast to mobile device, emulator has shorter launch period and dynamic testing phase. As results of using virtualization, multiple instances of emulator can be launched on multi-core PC at the same time without noticeable performance drops. From our point of view, such testing laboratory can find application in large-scale testing and automated forensics investigation of mobile applications

5.2.3 Rule-construction module

NF functionality for rules construction based on security metrics was implemented with help of C++ language and Standard Template Library (STL). The reason behind choice C++ is that it has low-level memory interaction. Compilable nature of C++ allows to achieve smaller execution time than interpretable nature of PHP or Python.

Developed module has following working algorithm:

1. Reading of training / validation labeled datasets (2 classes)
2. Unsupervised centroids detection for each of input metric using Radial Basis Function based on a standard deviation and a mean
3. Rules extracted for all possible combinations of metrics
4. ANN learning. It was used following NF parameters:
   - amount of epochs: 1000
   - weights learning rule: Delta

     $$\triangle w = \mu \cdot (y_i - d_i) \cdot f(network) \cdot x_i \tag{5.11}$$

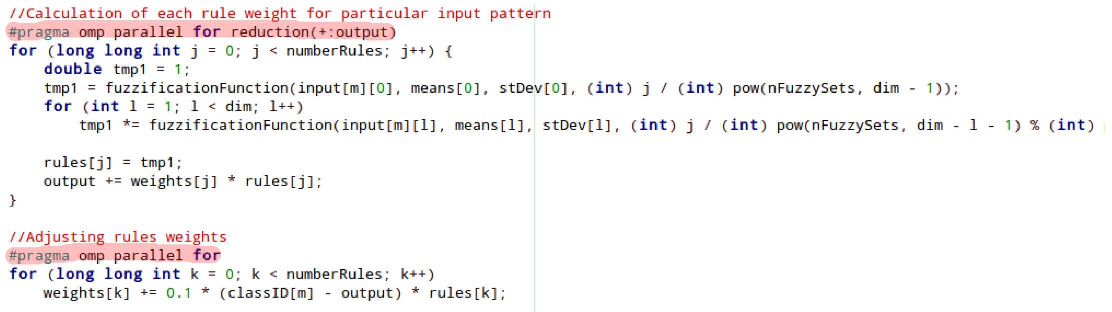
     Where μ - step of learning, d_i - class of the input patter predicted by ANN, y_i - actual class of the input pattern, $f(network)$ - output value of the activation function, x_i - input pattern [76, 93].
   - Initial Gradient Descent step: 0.1

- membership function: Gaussian $\mu(x_i) = \frac{1}{e^{\frac{(x-a)^2}{2b^2}}}$
- activation function: Sigmoid: $d_i = \frac{1}{1+e^{-f(network)}}$

5. Rules tuning and selection using neurons weights
6. Validation of trained model

While developing the rules construction, we found that there exists narrow performance places in ANN learning. This is caused by consequent execution of mathematical operations targeted on weights update and membership values recalculation for each of the rules. However, the problem can be easily divided into sub-problems. Modern hardware and software technologies offer flexible and powerful computational resources, particularly paralleling. As it can be seen that if several of sub-problems execute in parallel, then less time is needed to solve the whole problem of ANN learning. After extensive studying of related literature, we decided to choose following technologies and corresponding API:

- **OpenMP**. Specification for multi-threading on PC for C/C++ and Fortran, which supports both data and task parallelisms [114]. The main advantage is flexibility in dividing calculation tasks into several sub-tasks, independently from processor architecture. Moreover, there are no need to use API or implement additional functionality. OpenMP consists of *#pragma* compiler directives, which describe specific implementation details. The Figure 40 shows how OpenMP is used in order to improve performance of ANN learning. As it can be seen, outer for-loop uses paralleling into several sub-loops. Local copy of variable "output" is calculating for each sub-loop. When calculations are finishes, global "output" is calculated from local copies. As results, total increase of speed is almost equal to amount of thread (or cores).



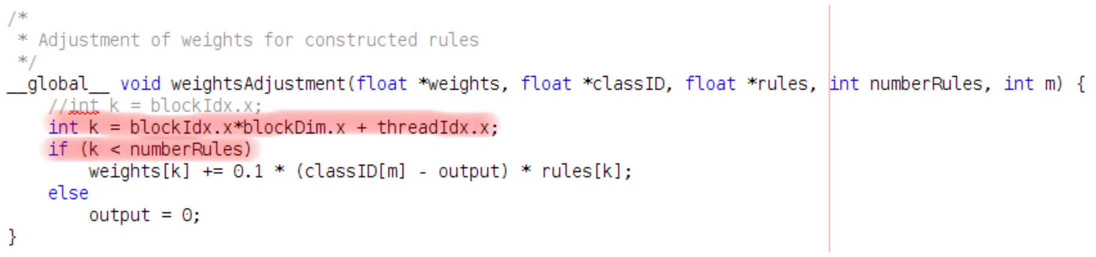
```
//Calculation of each rule weight for particular input pattern
#pragma omp parallel for reduction(+:output)
for (long long int j = 0; j < numberRules; j++) {
    double tmp1 = 1;
    tmp1 = fuzzificationFunction(input[m][0], means[0], stDev[0], (int) j / (int) pow(nFuzzySets, dim - 1));
    for (int l = 1; l < dim; l++)
        tmp1 *= fuzzificationFunction(input[m][l], means[l], stDev[l], (int) j / (int) pow(nFuzzySets, dim - l - 1) % (int)

    rules[j] = tmp1;
    output += weights[j] * rules[j];
}

//Adjusting rules weights
#pragma omp parallel for
for (long long int k = 0; k < numberRules; k++)
    weights[k] += 0.1 * (classID[m] - output) * rules[k];
```

Figure 40: Sample of ANN learning implementation using OpenMP in C++

- **CUDA native code**. CUDA targeted on scientific calculations and computing as it is stated on the web-site of NVIDIA[6] company [115]. Graphic card available to authors supports only CUDA version 1.2, which is limited only to a single-precision float operations. In our case ANN learning does not require double-precision and compatible for usage with such

[6]http://www.nvidia.com

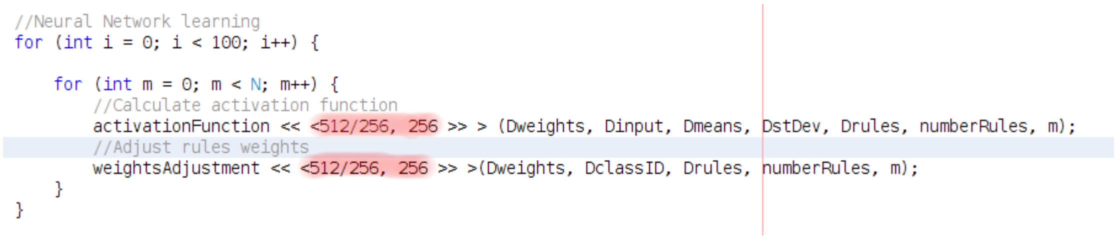
version of CUDA. Despite the fact that CUDA native code is compatible with C/C++, it requires to use specific Nvidia compiler - *nvcc*. There is the example of developed kernel functions in the Figure 41. GPU has multiple kernel that allows to split for loops into several parallel tasks.



```
/*
 * Adjustment of weights for constructed rules
 */
__global__ void weightsAdjustment(float *weights, float *classID, float *rules, int numberRules, int m) {
    //int k = blockIdx.x;
    int k = blockIdx.x*blockDim.x + threadIdx.x;
    if (k < numberRules)
        weights[k] += 0.1 * (classID[m] - output) * rules[k];
    else
        output = 0;
}
```

Figure 41: Sample of rules weights adjustment function using naive CUDA in C++

Then linear combiner with activation function and weights adjustments function are called inside ANN learning phase. As it could be seen from the Figure 42, learning phase contain two kernel functions, which execute in parallel. In order to achieve good results, we need to specify amount of threads and corresponding number of blocks as kernel function parameters <BLOCK_NUM, THREAD_NUM>

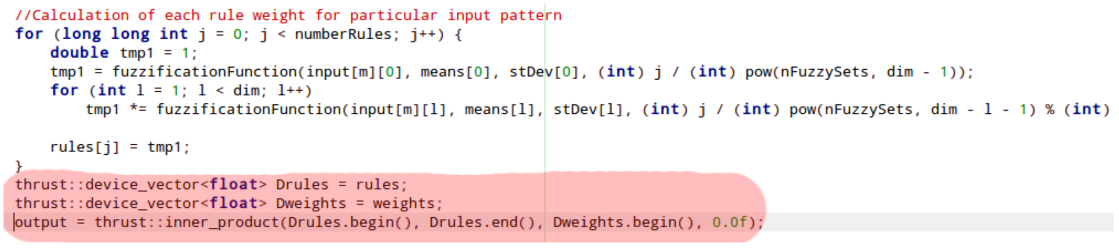


```
//Neural Network learning
for (int i = 0; i < 100; i++) {

    for (int m = 0; m < N; m++) {
        //Calculate activation function
        activationFunction << <512/256, 256 >> > (Dweights, Dinput, Dmeans, DstDev, Drules, numberRules, m);
        //Adjust rules weights
        weightsAdjustment << <512/256, 256 >> >(Dweights, DclassID, Drules, numberRules, m);
    }
}
```

Figure 42: Sample of ANN learning implementation (weights adjustment) using naive CUDA in C++

CUDA utilizes multiple threads on GPU (in our case 512) in contrast to a few threads on CPU. That is why we are able to execute kernel functions in multiple parallel threads. As amount of threads are large, then the calculations related to each rule are executed in parallel. On considerable amount of rules, we noticed significant speed-up in ANN learning. Additional information related to used GPU specification and CUDA characteristics can be found in the Appendix G.

- **Thrust**. This is a library that consistent and unified storage containers (such that vectors) and fast algorithms for such vectors processing [116]. It has the same purpose as Standard Template Library (STL) for C++ - simplifying and supporting of CUDA-related [115] software development. Thrust provides level of abstraction above native CUDA code. For ANN learning optimization, we applied GPU acceleration in computing of activation function. This operation consists of inner product calculation of weights and rules membership vectors. Implementation of Thrust code is depicted in the Figure 43. By transferring both vectors data to a GPU memory, then several cores inside GPU are used to compute inner product. We noticed that speed-up can be observed only on big amount of data in vectors, because they need to be transferred to GPU, then processed and transferred back. It causes additional delays, even in spite of fast parallel execution of inner product. Thrust is more aimed on processing of stream data, sorting and transformation of vectors. As a result, it is not suitable in case of specific atomic mathematical operations on each vector's element that requires multiple CPU-GPU interactions.



```
//Calculation of each rule weight for particular input pattern
for (long long int j = 0; j < numberRules; j++) {
    double tmp1 = 1;
    tmp1 = fuzzificationFunction(input[m][0], means[0], stDev[0], (int) j / (int) pow(nFuzzySets, dim - 1));
    for (int l = 1; l < dim; l++)
        tmp1 *= fuzzificationFunction(input[m][l], means[l], stDev[l], (int) j / (int) pow(nFuzzySets, dim - l - 1) % (int)

    rules[j] = tmp1;
}
thrust::device_vector<float> Drules = rules;
thrust::device_vector<float> Dweights = weights;
output = thrust::inner_product(Drules.begin(), Drules.end(), Dweights.begin(), 0.0f);
```

Figure 43: Sample of weights adjustment using Thrust library in C++

It can be summarized that parallel computing is powerful computational approach. Moreover, it was proved that proposed method has abilities to perform calculations in parallel. This requires complex optimization of data manipulation and calculation procedures. Nevertheless, it is feasible to achieve a huge speed-up on multi-core CPUs and GPUs systems. In our view, automated analysis of the mobile applications obtains more benefits while using parallel computing.

5.2.4 Performance concerns

In this subsection we present all performance aspects of mobile application testing. Further it is shown the advantage of using the testing laboratory in automated mobile application testing. As it was mentioned previously, mobile devices spend much time on booting, which makes them hardly applicable for big datasets testing in dynamic environment.

Single application testing time of existing not automated solution DroidBox is around 10 minutes [99]. Our testing laboratory with implemented features uses around 1.5 minutes to test one application. This is because significant optimization work was made while implementing the laboratory. Particularly, application execution in emulated environment and ANN learning were adjusted.

The experiments were executed with following configuration. For emulated environment we used VM with 1024MB of RAM memory and 256M VM heap[7]. Intel and AMD Virtualization (for both testing machines) and GeForce N210 acceleration were enabled. It should be mentioned that only Android version higher than 4.0.x supports GPU acceleration. There were performed several launches and average times of Android 4.2.2 emulator boot is as presented in the Table 10.

In order to compare performance of real device, we were able to test booting time on Archos 101 Internet Tablet (CPU ARM 1Ghz, RAM 256MB). The total time of Android 2.2.1 booting on

[7]Type of memory used for dynamical allocation

Processor	ARM EABI v7	native x86
Athlon X2 4400+ 2.3 2.3GHz x2 (1)	130	32
Intel Core2 6300 1.86GHz x2 (2)	124	29
Intel Core2 Duo T8100 2.10GHz x2 (3)	108	22

Table 10: Booting time of an Android 4.2.2 SDK emulator, seconds

this devices is 40-50 seconds, which is bigger than time for emulated Android 2.3.3 (for details see the Table 11.

Processor	ARM EABI v7	native x86
Athlon X2 4400+ 2.3 2.3GHz x2 (1)	53	21
Intel Core2 6300 1.86GHz x2 (2)	58	24
Intel Core2 Duo T8100 2.10GHz x2 (3)	45	19

Table 11: Booting time of an Android 2.3.3 SDK emulator, seconds

While performing application static and dynamic tests, there were extracted different data from emulated device. The Figure 12 shows diversity in data sizes for benign and malicious applications.

Applications	Total	Traffic	Function calls	Databases	Shared preferences
Benign	825	552	15	10	0.113
Malicious	1679	978	114	5	0.094

Table 12: Characteristics of the data pulled during dynamic tests for 252 benign and 360 malicious, MB

Average test time of application decreased from about 200 seconds to 80 seconds per each application while using emulated environment optimization. In total it took N hours to test benign and N hours to test malicious applications.

Another side of optimization is ANN learning. We tried various paralleling techniques and obtained the results, shown in the Figure 13. It can be seen that OpenMP improved performance significantly, while Thrust spent excess time on transferring data between CPU and GPU.

To summarize achieved results, we make next statements. Modern computational approaches and technologies help to optimize time complexity of developed theoretical base. Applying parallel computing makes possible to achieve better speed than it was projected for the single-thread program.

5.3 On-site defence perspective

Before we consider only analysis of mobile applications in emulated environment of personal computer. At the same time, developed security metrics and rules extraction method can be applied for malware detection on real mobile device. However, there might be shortage in extracted features used for security metrics construction due to access limitations. All other data are available for analysis including applications data storage and functions calls.

In our view such defense could be organized as C++ library using JNI to interact with Android OS that based on Java [30]. There are no need to use emulated environment as result.

Used technology	**125 rules (3 var)**	**3125 rules (5 var)**	**15625 rules (6 var)**
Seq. execution	6	253	1508
OpenMP	5	161	860
CUDA native	**3**	**4**	**5**
Thrust	8	233	1903

Table 13: Amount of time taken by ANN learning process for different amount of metrics (variables) with five terms, seconds (user time from a Linux *time* command)

Unfortunately OpenMP, Thrust and CUDA native code are not supported Android OS features at the moment of writing the master thesis. However, NVIDIA announced that it will be produced a tablet Tegra 5 in 2014 with CUDA support [117]. Developing of on-site detection and analysis program can be defined as future work.

6 Summary of Findings & Implications

In this master thesis, we investigated automatic fuzzy-rule extraction for malware detection in mobile devices. Particularly, there were answered four research questions. In order to answer them, the big dataset with around thousand malicious and benign application for mobile devices was composed. To the knowledge, such dataset did not exist before. That means it should be developed by ourselves.

6.1 Overview of main results

The first research question was "What kind of security metrics could be applied for malware detection and what is the detection reliability of such metrics?". In order to answer this research question, we performed analysis and study of corresponding literature. Security metric is a collection of features from a particular domain related to information security. It was found that there exists developed security metrics theory for the PC, which are not appropriate for the mobile platforms. Therefore, it was proposed the method of metrics construction from raw features. Additionally, we utilize expert knowledge and RELIEF feature-selection method in order to evaluate significance of particular feature in each metrics. As a result, five security metrics were extracted from 36 features. This is less than tens of features previously used for malware detection [118]. In the Chapter 4 it was proved that detection process is more understandable than SVM, while classification accuracy is not significantly affected.

The second question was defined as "How do user profiles (various sensitive and private information stored on the device) affect malware behavior and what kind of data are stored/-transmitted by malware?". To get answer on this question, it was study ways and types of user information that can be stored on the devices. Common user profile with sensitive information (contacts, messages, etc) was generated and used further during malicious and benign applications testing. After processing of the test data it was shown that there is sensitive information leakage, yet only a single malicious application sent such data in the open format. We can state that presence of sensitive information affects malware detection, yet the influence is negligible.

The third question was declared as following: "Are the results of static and dynamic testing of mobile applications reliable for automated malware detection?". While seeking answer regarding this question, we conducted study of the literature related to manual analysis. Furthermore, we rebuilt experimental part that was based on previous work [16]. It was improved testing process (both static and dynamic phases) and extended amount of features that are extracted for each application. The executed tests showed that automatically extracted data have the same meaning for malware detection as data from manual analysis. However, manual analysis is much slower and required malware reverse engineering knowledge. Furthermore, in case of used binary library it is necessary to apply advanced disassemblers like IDA pro[1]. Finally, we say that proposed

[1]`https://www.hex-rays.com/products/ida/index.shtml`

methodology and developed test routine are much efficient in time and complexity perspective than manual analysis with the same reliability.

The last of the fourth research questions is "Is it possible to automatically extract corresponding advanced fuzzy rules and provide then fair detection rate?". Firstly, it was proved that fuzzy rules can be not only used in malware detection, yet can be interpretable for human. We propose robust and efficient malware detection method based on NF approach. The rules were not only extracted from the training dataset, yet also there were selected the most significant according to weights in neural-network. It allows to use smaller amount of rules without significant affect on the detection. For results comparison were used advanced methods SVM and EM as presented in the Chapter 4. Nevertheless, proposed method for rules extraction is presented. It provides reliable detection rate with human-understandable statistical model. Secondly, there was implemented module for rules construction in C++. The performed tests demonstrated that our method can be used in malware detection with appropriate detection rate.

The thesis also uncovered that parallel computing is able to improve performance of the proposed detection method. As an additional part of experiments, we conducted implementation part along with parallel computing support. Then complexity results show that utilization of parallel abilities of modern CPUs for ANN learning can not compete with NVIDIA CUDA technology. This is because GPU with CUDA support allows to execute the sub-tasks in thousands of threads, while CPUs have just tens of threads. It was proved by authors that even complex learning phases can be considerably optimized in time complexity domain. During experiments we achieved speed up in hundreds of times in compare to single CPU execution. We can conclude that usage of sophisticated theoretical methods along with modern computational technologies and optimization provides effective and fast solutions for malware detection.

Summary

In order to conclude findings, it can be mentioned *achieved findings* in following areas:

- A comprehensive and full-scale database of benign (460) and malicious (521) mobile applications for Android platform.
- Unique dataset that consists of extracted information from static and dynamic (traffic, function calls) tests. There were tested 353 malicious and 251 benign applications. *Note:* amount of tested applications is less than amount of collected. This is because same application has different names in various software market. After MD5 hash sum comparison of collected applications we found duplicates and therefore decided to test only unique applications.
- Dataset of 40 features extracted from performed tests with 604 labeled samples (two classes).
- Dataset of five constructed security metrics from the features with 604 labeled samples (two classes).

Theoretical and corresponding implementation achievements are next:

- Security metrics theory and extraction method.

- A novel theoretical justification for human-understandable fuzzy-rule extraction in malware detection using NF approach.
- Improved "Testing" and "Analysis" parts of previously developed by authors testing laboratory [16].
- Implemented in C++ rules extraction method (with OpenMP and CUDA support).
- A starting vector and base for future work on parallel and optimization of the proposed rules extraction approach.

6.2 Theoretical implications

In this Section theoretical implications of the performed research are investigated. The thesis seeks for the method of deriving malware detection rules that are both reliable and interpretable by human. Corresponding security metrics were extracted for these purposes from the applications. In addition to this, it was studied performance of the other binary classification methods in order to compare detection error. This Section provides theoretical implications of the researched area and findings.

It was found that applying security metrics can decrease complexity of statistical model used for malware detection. This is because dimensionality of metrics vector for each application is less in a few times then the feature vector. This is important aspect, because complexity of the detection method affect time complexity of model construction. It could happen that statistical learning from features will be infeasible due to dimensionality.

Despite the fact that there are already known research on security metrics and their application for PC and networks [119, 9, 120, 121, 47], there are still no theory for mobile devices. Only few of the security metrics can be projected from PC domain to mobile platform like CPU utilization. Other metrics are available only on mobile platform like stored databases and shared preferences in XML. Another difference of existing research on security metrics is that there are only used expert knowledge for metrics construction from the features. The author used attribute selection approach RELIEF for obtaining corresponding weights that requires minimum of expert knowledge.

As it was presented in the proof of concept in the Chapter 3, the method is able to utilize less detection rules without significant loss of accuracy. We can say that proposed detection methodology was successful because of there was used real dataset and the method provided good results on it. Finally, the derived rules can be used for teaching the security unaware or vulnerable users of mobile devices.

6.3 Practical Implications

The used dataset was composed from APK packages that are suing for applications installation of Android platform. Such applications were collected from publicly available samples of malicious an benign applications for Android platform. Also the practical findings of the study are totally replicable under the chosen methods that were described in the Chapter 3.

We understood that the features from applications tests can contain irrelevant data and noise components, which influence building of security metrics. However, have not caused huge error rates in malware detection neither in proposed method nor in other ML classification methods like SVM or EM. All performed tests in proof-of-concept in the Chapter 3 and in real malware detection in the Chapter 4 presented the method can be used as human-interpretable malware detection. Moreover, common user without deep technical knowledge of information security can understand the idea behind fuzzy rules. It offers insight of prerequisites for malware success even under deployed protection mechanisms and installed AV software.

The thesis found that application of parallel optimization of the method decreases time complexity by several orders of magnitude. We studied the method's complexity in the Chapter 5 and all further conclusions are scientifically proved. This is an interdisciplinary research that was inspired by seeking for reducing time complexity without damaging the accuracy. We found that CUDA can be considered as the best options in paralleling of the ANN learning process. During work on implementation we obtained valuable experience that reveals drawbacks and limitations of the multiprocessing. As a result, we state that parallel computing can significantly decrease delays while performing rules mining in dynamic and proactive environment. This is very important while using malware detection in critical infrastructures or for sensitive data protection.

6.4 Further work

To the authors hope this master thesis introduces a sizeable step to forensically sound malware detection using fuzzy rules in pro-active environment. Some of the existing difficulties and perspectives for future were discussed in master thesis already. We want to concentrate on most important fields of work that can be researched.

- *Extraction of additional features*

  During the work on the master thesis, there were extracted nearly forty different raw features from each application during static and dynamic testing. However, there are still perspective of extraction much more specific features. In our view, detection based on bigger amount of features is more robust.

- *Metrics composition*

  As was discussed previously in the Chapter 3, RELIEF method was chosen in order to extract merits for each particular feature in a single metric. However, later it was shown in the Chapter 4 that the extracted fuzzy rules based on the metrics do not provide the best results in malware classification. Therefore, further research should be performed on feature selection. Particularly, we foresee utilization of generic feature selection measure [84] and SVM attributes evaluation.

- *On-side defense*

  Since this work was targeted the field of malware analysis in proactive automated environment, we did not concentrate much attention on device-side malware detection. Nevertheless, extracted features and later metrics are feasible to extract from inside mobile OS as well. As a result, C++ library should be developed using rule-construction module. Then, through the JNI interface it is connected to Android application in order to provide fast data exchange.

- *On-line learning testing*

  The important thing is not only to develop strong theoretical base but also to utilize strong and efficient optimization while implementing. In our view, there should be provided additional tests in order to test time complexity and applicability of the method in dynamic and changeable environment.

- *Expert system perspective*

  The master thesis studied the testing, analysis and detection of the malware on mobile devices. Despite the reliable detection, the malware prevention and isolation should be performed as well.

Bibliography

[1] The pros and cons of behavioral based, signature based and whitelist based security (online). URL: `http://www.windowsecurity.com/articles-tutorials/misc_network_security/Pros-Cons-Behavioral-Signature-Whitelist-Security.html` (Visited 09.04.2013). 1, 11, 40

[2] Why relying on antivirus signatures is simply not enough anymore (online). URL: `http://blog.webroot.com/2012/02/23/why-relying-on-antivirus-signatures-is-simply-not-enough-anymore/` (Visited 09.04.2013). 1, 11, 40

[3] Morris worm - source code, investigations and study (online). 2012. URL: `http://ftp.cerias.purdue.edu/pub/doc/morris_worm/` (Visited 27.05.2013). 2

[4] Dunham, K. 2009. *Mobile Malware Attacks and Defense*. Syngress Publishing. 2, 9

[5] Hoog, A. 2011. *Android Forensics: Investigation, Analysis and Mobile Security for Google Android*. Syngress Publishing, 1st edition. 2, 19

[6] Mobile app security study (online). URL: `https://viaforensics.com/resources/reports/mobile-app-security-study/` (Visited 10.05.2013). 2

[7] Hamon, V. Operational cryptology and virology lab (online). 2012. URL: `http://cvo-lab.blogspot.fr/2012/08/android-malware-smszombie-in-depth.html` (Visited 05.12.2012). 2

[8] Asef: Android security evaluation framework (online). URL: `http://code.google.com/p/asef/` (Visited 16.11.2012). 2

[9] Payne, S. C. A guide to security metrics. Technical report, 2006. 2, 12, 24, 79

[10] Kononenko, I. & Kukar, M. 2007. *Machine Learning and Data Mining: Introduction to Principles and Algorithms*. Horwood Publishing Limited. 2, 13, 14, 25, 36, 37, 41, 48, 58, 59

[11] Nwokedi Idika, A. P. M. A survey of malware detection techniques. 2007. 3

[12] Marpaung, J., Sain, M., & Lee, H.-J. feb. 2012. Survey on malware evasion techniques: State of the art and challenges. In *Advanced Communication Technology (ICACT), 2012 14th International Conference on*, 744 –749. 3, 21, 54

[13] Singh, R., Kumar, H., & Singla, R. October 2011. Review of soft computing in malware detection. *Special issues on IP Multimedia Communications*, (1), 55–60. Published by Foundation of Computer Science, New York, USA. 3, 16, 29

[14] Altaher, A., Almomani, A., & Ramadass, S. 2012. Application of adaptive neuro-fuzzy inference system for information secuirty. *Journal of Computer Science*, 8(6), 983–986. 3, 15, 29

[15] Zhang, Y., Pang, J., Yue, F., & Cui, J. 2010. Fuzzy neural network for malware detect. In *Intelligent System Design and Engineering Application (ISDEA), 2010 International Conference on*, volume 1, 780–783. doi:10.1109/ISDEA.2010.314. 3, 16, 29

[16] Shalaginov, A. Testing laboratory for mobile applications and malware analysis. IMT4882 Specialization Course Project Report, Gjøvik University College, dec 2012. 3, 29, 54, 66, 67, 77, 79

[17] Xie, L., Zhang, X., Seifert, J.-P., & Zhu, S. 2010. pbmds: a behavior-based malware detection system for cellphone devices. In *Proceedings of the third ACM conference on Wireless network security*, WiSec '10, 37–48, New York, NY, USA. ACM. URL: http://doi.acm.org/10.1145/1741866.1741874, doi:10.1145/1741866.1741874. 4

[18] Burguera, I., Zurutuza, U., & Nadjm-Tehrani, S. 2011. Crowdroid: behavior-based malware detection system for android. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, SPSM '11, 15–26, New York, NY, USA. ACM. URL: http://doi.acm.org/10.1145/2046614.2046619, doi:10.1145/2046614.2046619. 4

[19] Adolphi, B. Cross-platform evaluation of mobile app hardening. Master's thesis, Gjøvik University College, Norway, 2012. 5, 11

[20] Google play (online). URL: https://play.google.com/store (Visited 20.03.2013). 5, 29

[21] Kaspersky security bulletin 2012. malware evolution (online). URL: http://www.securelist.com/en/analysis/204792254/Kaspersky_Security_Bulletin_2012_Malware_Evolution (Visited 16.03.2013). 5

[22] Apple app store (online). URL: http://www.apple.com/iphone/from-the-app-store/ (Visited 20.03.2013). 5, 29

[23] Designing a defense for mobile applications. examining an ecosystem of risk. Technical report, 2013. 9

[24] Mobile malware report: How users drive the mobile threat landscape. Technical report, 2013. 9, 13

[25] Boksasp, T. & Utnes, E. Android apps and permissions: Security and privacy risks. Master's thesis, NTNU - Trondheim, Norwegian University of Science and Technology, Norway, 2012. 9, 10

[26] Felt, A. P. & Wagner, D. 2011. Phishing on mobile devices. In *In W2SP*. 9

[27] Asokan, N. 2007. Phishingand mobile phones. USEC '07 "The Future of Phishing"panel discussion. 9

[28] Van der Merwe, A. J. & Seker, R. Mobile phishing (online). 2004. URL: http://hufee.meraka.org.za/Hufeesite/staff/the-hufee-group/altas-documents/Twopager-Nov142k4.pdf (Visited 17.12.2012). 9

[29] Castillo, C. A. Android malware past, present, and future (online). URL: http://www.mcafee.com/us/resources/white-papers/wp-android-malware-past-present-future.pdf. 10, 13

[30] Android sdk (online). URL: http://developer.android.com/index.html (Visited 10.11.2012). 10, 18, 19, 52, 54, 69, 70, 74

[31] Send_sms capability leak in android open source project (aosp), affecting gingerbread, ice cream sandwich, and jelly bean (online). Xuxian Jiang. URL: http://www.csc.ncsu.edu/faculty/jiang/send_sms_leak.html (Visited 01.06.2013). 10

[32] Grace, M., Zhou, Y., Wang, Z., & Jiang, X. feb 2012. Systematic detection of capability leaks in stock Android smartphones. In *Proceedings of the 19th Network and Distributed System Security Symposium (NDSS)*. URL: http://www.csc.ncsu.edu/faculty/jiang/pubs/NDSS12_WOODPECKER.pdf. 10

[33] Android vs ios infographic (online). URL: http://www.veracode.com/resources/android-ios-security (Visited 10.04.2013). 10

[34] Android and security (online). URL: http://googlemobile.blogspot.no/2012/02/android-and-security.html (Visited 08.04.2013). 10

[35] Android based phones rules the world's smartphones market (online). URL: http://www.androidnova.org/tag/mobile-os-usage-statistics/ (Visited 12.04.2013). 10

[36] Android security overview (online). URL: http://source.android.com/tech/security/ (Visited 02.02.2013). 10

[37] Svajcer, V. Technical report. 10, 11

[38] Circumventing google's bouncer, android's anti-malware system (online). URL: http://www.extremetech.com/computing/130424-circumventing-googles-bouncer-androids-anti-malware-system (Visited 03.04.2013). 11, 29

[39] ios security. Technical report, 2012. 11

[40] Windows phone apps (online). URL: http://www.windowsphone.com/en-us/store (Visited 26.05.2013). 11

[41] Miscrosoft msdn - how device security affects application execution (online). URL: http://msdn.microsoft.com/ (Visited 26.05.2013). 11

[42] Whole product dynamic: Real world protection test report, anti-virus comparative - october 2012. Technical report, 2012. 11

[43] Test report: Anti-malware solutions for android. Technical report, 2012. 11

[44] Moser, A., Kruegel, C., & Kirda, E. dec. 2007. Limits of static analysis for malware detection. In *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, 421 –430. doi:10.1109/ACSAC.2007.21. 11

[45] Logical signatures in cmalav 0.94 (online). 2008. URL: http://vrt-blog.snort.org/2008/09/logical-signatures-in-clamav-094.html (Visited 27.12.2012). 12

[46] Chris I. Cain, E. C. Establishing a security metrics program. Technical report, 2011. 12

[47] Rathbun, D. Gathering security metrics and reaping the rewards. Technical report, 2009. 12, 79

[48] Torgersen, G. Online data explosion brings new forensic collection techniques (online). URL: http://www.evidencemagazine.com/index.php?option=com_content&task=view&id=1184&Itemid=9 (Visited 20.04.2013). 13

[49] Sahs, J. & Khan, L. aug. 2012. A machine learning approach to android malware detection. In *Intelligence and Security Informatics Conference (EISIC), 2012 European*, 141 –147. doi:10.1109/EISIC.2012.34. 13, 14, 29

[50] Shamili, A., Bauckhage, C., & Alpcan, T. 2010. Malware detection on mobile devices using distributed machine learning. In *Pattern Recognition (ICPR), 2010 20th International Conference on*, 4348–4351. doi:10.1109/ICPR.2010.1057. 13, 14

[51] Bose, A., Hu, X., Shin, K. G., & Park, T. 2008. Behavioral detection of malware on mobile handsets. In *Proceedings of the 6th international conference on Mobile systems, applications, and services*, MobiSys '08, 225–238, New York, NY, USA. ACM. URL: http://doi.acm.org/10.1145/1378600.1378626, doi:10.1145/1378600.1378626. 13

[52] Shabtai, A., Fledel, Y., & Elovici, Y. dec. 2010. Automated static code analysis for classifying android applications using machine learning. In *Computational Intelligence and Security (CIS), 2010 International Conference on*, 329 –333. doi:10.1109/CIS.2010.77. 13, 14

[53] Franke, K. *The influence of physical and biomechanical processes on the ink trace - Methodological foundations for the forensic analysis of signatures*. PhD thesis, University of Groningen, 2005. 15

[54] Mathworks - fuzzy inference process (online). URL: http://www.mathworks.se/help/fuzzy/fuzzy-inference-process.html (Visited 28.05.2013). xiii, 15

[55] Amiri, F., Lucas, C., & Yazdani, N. 2009. Anomaly detection using neuro fuzzy system. 16

[56] Jang, J.-S. 1993. Anfis: adaptive-network-based fuzzy inference system. *IEEE Transactions on Systems, Man and Cybernetics*, 23(3), 665–685. doi:10.1109/21.256541. 16, 34

[57] Stoffel, K., Cotofrei, P., & Han, D. 2010. Fuzzy methods for forensic data analysis. In *Soft Computing and Pattern Recognition (SoCPaR), 2010 International Conference of*, 23–28. doi:10.1109/SOCPAR.2010.5685848. 16

[58] Stoffel, K., Cotofrei, P., & Han, D. 2012. Fuzzy clustering based methodology for multi-dimensional data analysis in computational forensic domain. In *International Journal of Computer Information Systems and Industrial Management Applications*, 400–410. 16, 52

[59] Sulaiman Al amro, K. A. & Ajlan, A. A. A comparative study of computational intelligence in computer security and forensics. 2011. 16

[60] Under the hood of android emulator (appcert) (online). URL: https://wiki.diebin.at/Under_the_hood_of_Android_Emulator_(appcert) (Visited 02.05.2013). 18

[61] Android memory usage (online). URL: http://elinux.org/Android_Memory_Usage (Visited 20.04.2013). 20, 55

[62] Understanding statistics: Data types, variables, and p value. Technical report, 2010. 21

[63] Default android permissions explained, security tips, and avoiding malware (online). 2010. URL: http://androidforums.com/android-applications/36936-android-permissions-explained-security-tips-avoiding-malware.html (Visited 16.03.2013). 21

[64] Chia, P. H., Yamamoto, Y., & Asokan, N. 2012. Is this app safe?: a large scale study on application permissions and risk signals. In *Proceedings of the 21st international conference on World Wide Web*, WWW '12, 311–320, New York, NY, USA. ACM. URL: http://doi.acm.org/10.1145/2187836.2187879, doi:10.1145/2187836.2187879. 21

[65] Krassas, N. Android malware analysis (online). 2011. URL: http://resources.infosecinstitute.com/android-malware-analysis/ (Visited 20.05.2013). 21

[66] Gray, R. M. Entropy and information theory (online). 1990. URL: http://ee.stanford.edu/~gray/it.pdf (Visited 20.05.2013). 22, 24

[67] Antonio Arauzo-Azofra, J. M. B. & Castro, J. L. 2004. A feature set measure based on relief. 25

[68] Bau, J., Bursztein, E., Gupta, D., & Mitchell, J. 2010. State of the art: Automated black-box web application vulnerability testing. In *Security and Privacy (SP), 2010 IEEE Symposium on*, 332–345. doi:10.1109/SP.2010.27. 25

[69] Androguard: reverse engineering, malware and goodware analysis of android applications (online). URL: http://code.google.com/p/androguard/ (Visited 28.11.2012). 25

[70] Kiseleva, A., Kiselev, G., Sergeev, A., & Shalaginov, A. 2011. Processing the input data in multimodal applications. *Scientific and Technical Journal "Electronics and Communications"*, 2, 86–92. Kiev, Ukraine. 27

[71] Sand, L. A. Information-based dependency matching for behavioral malware analysis. Master's thesis, Gjovik University College, Norway, 2012. 27

[72] Strace (online). URL: `http://linux.die.net/man/1/strace` (Visited 07.05.2013). xiii, 27, 54, 55

[73] Barnett, J. A. 1982. Some issues of control in expert system. In *IEEE International Conference on Cybernetics and Society*, 421 –430. 28

[74] Kusnetzky, D. What is "big data?" (online). 2010. URL: `http://www.zdnet.com/blog/virtualization/what-is-big-data/1708`. 29, 41

[75] Scarpiniti, M. 2009. Neural networks. lesson 9 - fuzzy logic. Infocom Dept. - "Sapienza" university of Rome. 29, 30, 32

[76] Fuller, R. 1995. *Neural Fuzzy Systems*. Abo Akademi University. 29, 30, 32, 33, 34, 36, 70

[77] Looney, C. G. & Dascalu, S. 2007. A simple fuzzy neural network. In *Proceedings of the ISCA 20th International Conference on Computer Applications in Industry and Engineering, CAINE 2007, November 7-9, 2007, San Francisco, California, USA*, 12–16. ISCA. 30, 34

[78] Holeňa, M. 2005. Extraction of fuzzy logic rules from data by means of artificial neural networks. *Kybernetika*, 41(3), [297]–314. URL: `http://eudml.org/doc/33755`. 30

[79] Fuzzy logic description detailed (online). URL: `http://wing.comp.nus.edu.sg/pris/FuzzyLogic/DescriptionDetailed2.html` (Visited 15.04.2013). xiii, 31

[80] Lee, M. A. & Wessel, D. 1993/04/16 1993. Real-time neuro-fuzzy systems for adaptive control of musical processes. In *Applications of Fuzzy Logic Technology*, volume 2061, 464–75, Boston, MA. URL: `http://cnmat.berkeley.edu/publications/real_time_neuro_fuzzy_systems_adaptive_control_musical_processes`. xiii, 35

[81] S. Bouharati, K. Benmahammed, D. H. & El-Assaf, Y. 2008. Application of artificial neuro-fuzzy logic inference system for predicting the microbiological pollution in fresh water. In *Journal of Applied Sciences*, volume 8, 309–315. xiii, 37

[82] Wilamowski, B. 2003. Neural network architectures and learning. In *Industrial Technology, 2003 IEEE International Conference on*, volume 1, TU1–T12 Vol.1. `doi:10.1109/ICIT.2003.1290197`. 36

[83] Gradient descent method (online). URL: `http://fourier.eng.hmc.edu/e176/lectures/NM/node20.html` (Visited 20.05.2013). xiii, 38, 39

[84] Nguyen, H. T., Franke, K., & Petrovic, S. aug. 2010. Towards a generic feature-selection measure for intrusion detection. In *Pattern Recognition (ICPR), 2010 20th International Conference on*, 1529 –1532. `doi:10.1109/ICPR.2010.378`. 38, 80

[85] Fisher, R. A. Iris data set (online). 1986. URL: `Downloadedfromhttp://archive.ics.uci.edu/ml/datasets/Iris`. 39

[86] Akbarzadeh, V., Sadeghian, A., & dos Santos, M. V. 2008. Derivation of relational fuzzy classification rules using evolutionary computation. In *FUZZ-IEEE*, 1689–1693. IEEE. URL: http://dblp.uni-trier.de/db/conf/fuzzIEEE/fuzzIEEE2008.html#AkbarzadehSS08. xiii, 39, 40, 43

[87] Borovinskiy, V. (online). 39

[88] Jagielska, I., Matthews, C., & Whitfort, T. 1999. An investigation into the application of neural networks, fuzzy logic, genetic algorithms, and rough sets to automated knowledge acquisition for classification problems. *Neurocomputing*, 24(1–3), 37 – 54. URL: http://www.sciencedirect.com/science/article/pii/S0925231298000903, doi:10.1016/S0925-2312(98)00090-3. 39

[89] Discriminating between iris species (online). URL: http://www.r-bloggers.com/discriminating-between-iris-species/ (Visited 12.02.2013). xiii, 40

[90] Kiseleva, A. & Shalaginov, A. 2010. Hidden markov model for dealing with context application. In *Proceedings of the XVIII Ukrainian-Polish Conference "CAD in Machinery Design. Implementation and Education problems*", 20–22, Lvov, Ukraine; Warsaw, Poland. 41

[91] Chiang, H.-S. & Tsaur, W.-J. aug. 2010. Mobile malware behavioral analysis and preventive strategy using ontology. In *Social Computing (SocialCom), 2010 IEEE Second International Conference on*, 1080 –1085. doi:10.1109/SocialCom.2010.160. 41

[92] Wang, L. High dimensional data analysis (online). URL: http://lilywang.myweb.uga.edu/Research/highdimension.pdf (Visited 04.05.2013). 42, 51

[93] Pricing and hedging derivative securities with neural networks: Bayesian regularization, early stopping, and bagging (online). URL: http://neuron.csie.ntust.edu.tw/homework/94/neuron/Homework3/M9409204/discuss.htm (Visited 12.04.2013). xiii, 44, 70

[94] Weka (online). URL: http://www.cs.waikato.ac.nz/~ml/weka/ (Visited 08.12.2012). xvii, 48, 50, 51, 58

[95] Daubert - viaforensics (online). URL: https://viaforensics.com/computer-forensic-ediscovery-glossary/what-is-daubert.html (Visited 16.05.2013). 52

[96] yaffs2utils - utilities to create/extract a yaffs2 image on linux (online). URL: http://code.google.com/p/yaffs2utils/ (Visited 19.04.2013). 52

[97] Forensic analysis of the android file system yaffs2 (online). 2011. URL: http://ro.ecu.edu.au/cgi/viewcontent.cgi?article=1100&context=adf (Visited 01.06.2013). 52

[98] Are (android reverse engineering) (online). URL: http://maj3sty.tistory.com/993 (Visited 09.05.2013). 53

[99] Droidbox: Android application sandbox (online). URL: http://code.google.com/p/droidbox/ (Visited 25.11.2012). 53, 73

[100] Zhou, Y. & Jiang, X. may 2012. Dissecting android malware: Characterization and evolution. In *Security and Privacy (SP), 2012 IEEE Symposium on*, 95 –109. doi:10.1109/SP.2012.16. 54

[101] Tcpdump - command-line packet analyser (online). URL: http://www.tcpdump.org/ (Visited 07.05.2013). 54

[102] Wireshark - network traffic analyser (online). URL: http://www.wireshark.org/ (Visited 07.05.2013). 54, 55

[103] Passeri, P. One year of android malware (full list) (online). 2011. URL: http://hackmageddon.com/2011/08/11/one-year-of-android-malware-full-list/ (Visited 10.11.2012). 55

[104] Current android malware (online). 2012. URL: http://forensics.spreitzenbarth.de/android-malware/ (Visited 10.11.2012). 55

[105] Sms zombie in depth (online). 2012. URL: http://cvo-lab.blogspot.fr/2012/08/android-malware-smszombie-in-depth.html (Visited 07.12.2012). 55

[106] Domaintools (online). URL: http://whois.domaintools.com/ (Visited 03.05.2013). 55

[107] Abbass, H. A., Bacardit, J., Butz, M. V., & Llorà, X. Online adaptation in learning classifier systems: Stream data mining (online). 2004. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.89.9243&rep=rep1&type=pdf (Visited 03.04.2013). 57

[108] Parkour, M. Mobile malware dump from july 2011 (online). URL: http://contagiodump.blogspot.no/2011/03/take-sample-leave-sample-mobile-malware.html (Visited 16.12.2012). 63

[109] S.Adamchik, V. Algorithmic complexity (online). 2011. URL: http://www.cs.cmu.edu/~adamchik/15-121/lectures/Algorithmicy/complexity.html (Visited 24.05.2013). 64

[110] Taft, D.-P. I. G. Amdahl's law (online). URL: http://www.gordon-taft.net/Amdahl_Law.html (Visited 30.05.2013). 65

[111] Kvm - kernel based virtual machine (online). URL: http://www.linux-kvm.org (Visited 12.04.2013). 68, 69

[112] Grisenthwaite, R. Armv8technology preview (online). 2011. URL: http://www.arm.com/files/downloads/ARMv8_Architecture.pdf (Visited 10.05.2013). 69

[113] J.Stan Cox, Piyush Agarwal, N. G. H. H. Ibm websphere application server 64-bit performance demystified (online). 2007. URL: ftp://ftp.software.ibm.com/software/webserver/appserv/was/64bitPerf.pdf (Visited 12.05.2013). 69

[114] The openmp api specification for parallel programming (online). URL: http://openmp.org/ (Visited 20.05.2013). 71

[115] Nvidia compute unified device architecture (online). URL: https://developer.nvidia.com/category/zone/cuda-zone (Visited 20.05.2013). xiv, 71, 72, 139

[116] Thrust - code at the speed of light (online). URL: code.google.com/p/thrust/ (Visited 21.05.2013). 72

[117] Latif, L. Nvidia says tegra 5 'logan' will support cuda (online). URL: http://www.theinquirer.net/inquirer/news/2255917/nvidia-says-tegra-5-logan-will-support-cuda (Visited 18.05.2013). 75

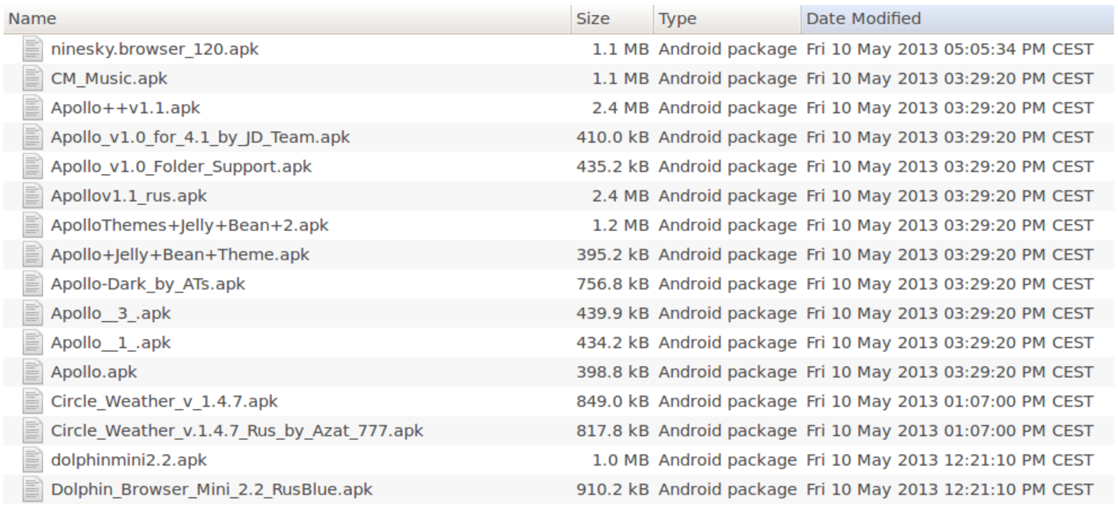
[118] Dini, G., Martinelli, F., Saracino, A., & Sgandurra, D. 2012. Madam: a multi-level anomaly detector for android malware. In *Proceedings of the 6th international conference on Mathematical Methods, Models and Architectures for Computer Network Security: computer network security*, MMM-ACNS'12, 240–253, Berlin, Heidelberg. Springer-Verlag. URL: http://dx.doi.org/10.1007/978-3-642-33704-8_21, doi:10.1007/978-3-642-33704-8_21. 77

[119] Establishing a security metrics program. Technical report, Chris I. Cain, Erik Couture. 79

[120] Aiello, M. Security metrics. 2005. 79

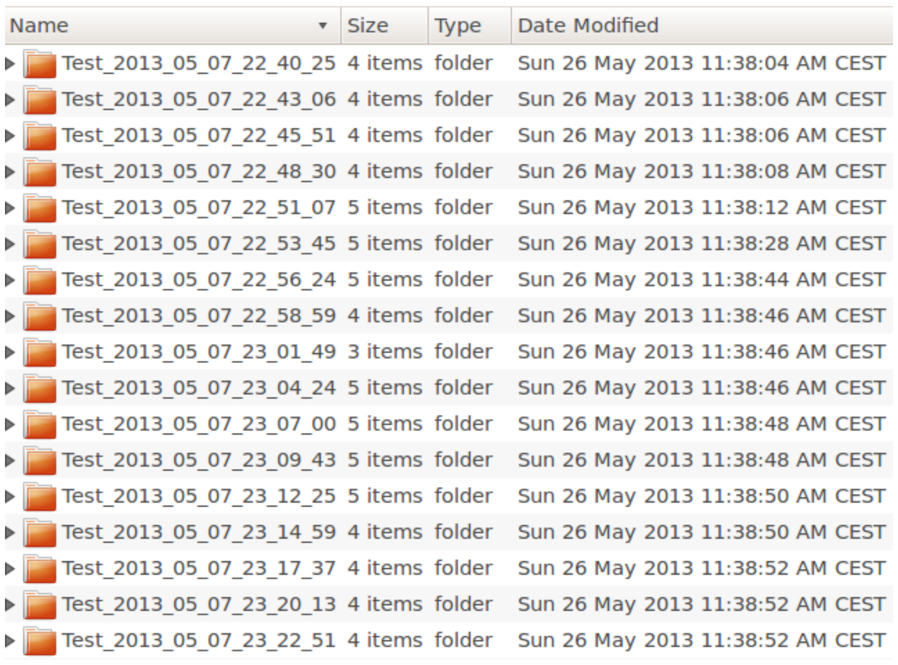
[121] Bayuk, J. april 2011. Alternative security metrics. In *Information Technology: New Generations (ITNG), 2011 Eighth International Conference on*, 943 –946. doi:10.1109/ITNG.2011.162. 79

A Data sets



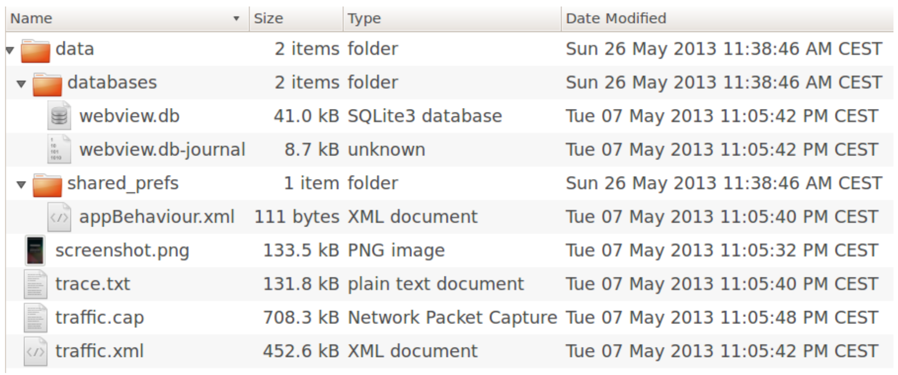
Name	Size	Type	Date Modified
ninesky.browser_120.apk	1.1 MB	Android package	Fri 10 May 2013 05:05:34 PM CEST
CM_Music.apk	1.1 MB	Android package	Fri 10 May 2013 03:29:20 PM CEST
Apollo++v1.1.apk	2.4 MB	Android package	Fri 10 May 2013 03:29:20 PM CEST
Apollo_v1.0_for_4.1_by_JD_Team.apk	410.0 kB	Android package	Fri 10 May 2013 03:29:20 PM CEST
Apollo_v1.0_Folder_Support.apk	435.2 kB	Android package	Fri 10 May 2013 03:29:20 PM CEST
Apollov1.1_rus.apk	2.4 MB	Android package	Fri 10 May 2013 03:29:20 PM CEST
ApolloThemes+Jelly+Bean+2.apk	1.2 MB	Android package	Fri 10 May 2013 03:29:20 PM CEST
Apollo+Jelly+Bean+Theme.apk	395.2 kB	Android package	Fri 10 May 2013 03:29:20 PM CEST
Apollo-Dark_by_ATs.apk	756.8 kB	Android package	Fri 10 May 2013 03:29:20 PM CEST
Apollo__3_.apk	439.9 kB	Android package	Fri 10 May 2013 03:29:20 PM CEST
Apollo__1_.apk	434.2 kB	Android package	Fri 10 May 2013 03:29:20 PM CEST
Apollo.apk	398.8 kB	Android package	Fri 10 May 2013 03:29:20 PM CEST
Circle_Weather_v_1.4.7.apk	849.0 kB	Android package	Fri 10 May 2013 01:07:00 PM CEST
Circle_Weather_v.1.4.7_Rus_by_Azat_777.apk	817.8 kB	Android package	Fri 10 May 2013 01:07:00 PM CEST
dolphinmini2.2.apk	1.0 MB	Android package	Fri 10 May 2013 12:21:10 PM CEST
Dolphin_Browser_Mini_2.2_RusBlue.apk	910.2 kB	Android package	Fri 10 May 2013 12:21:10 PM CEST

Figure 44: Sample of folder with collected benign applications



Name	Size	Type	Date Modified
Test_2013_05_07_22_40_25	4 items	folder	Sun 26 May 2013 11:38:04 AM CEST
Test_2013_05_07_22_43_06	4 items	folder	Sun 26 May 2013 11:38:06 AM CEST
Test_2013_05_07_22_45_51	4 items	folder	Sun 26 May 2013 11:38:06 AM CEST
Test_2013_05_07_22_48_30	4 items	folder	Sun 26 May 2013 11:38:08 AM CEST
Test_2013_05_07_22_51_07	5 items	folder	Sun 26 May 2013 11:38:12 AM CEST
Test_2013_05_07_22_53_45	5 items	folder	Sun 26 May 2013 11:38:28 AM CEST
Test_2013_05_07_22_56_24	5 items	folder	Sun 26 May 2013 11:38:44 AM CEST
Test_2013_05_07_22_58_59	4 items	folder	Sun 26 May 2013 11:38:46 AM CEST
Test_2013_05_07_23_01_49	3 items	folder	Sun 26 May 2013 11:38:46 AM CEST
Test_2013_05_07_23_04_24	5 items	folder	Sun 26 May 2013 11:38:46 AM CEST
Test_2013_05_07_23_07_00	5 items	folder	Sun 26 May 2013 11:38:48 AM CEST
Test_2013_05_07_23_09_43	5 items	folder	Sun 26 May 2013 11:38:48 AM CEST
Test_2013_05_07_23_12_25	5 items	folder	Sun 26 May 2013 11:38:50 AM CEST
Test_2013_05_07_23_14_59	4 items	folder	Sun 26 May 2013 11:38:50 AM CEST
Test_2013_05_07_23_17_37	4 items	folder	Sun 26 May 2013 11:38:52 AM CEST
Test_2013_05_07_23_20_13	4 items	folder	Sun 26 May 2013 11:38:52 AM CEST
Test_2013_05_07_23_22_51	4 items	folder	Sun 26 May 2013 11:38:52 AM CEST

Figure 45: Sample of folders with corresponding applications information after tests execution



Name	Size	Type	Date Modified
data	2 items	folder	Sun 26 May 2013 11:38:46 AM CEST
databases	2 items	folder	Sun 26 May 2013 11:38:46 AM CEST
webview.db	41.0 kB	SQLite3 database	Tue 07 May 2013 11:05:42 PM CEST
webview.db-journal	8.7 kB	unknown	Tue 07 May 2013 11:05:42 PM CEST
shared_prefs	1 item	folder	Sun 26 May 2013 11:38:46 AM CEST
appBehaviour.xml	111 bytes	XML document	Tue 07 May 2013 11:05:40 PM CEST
screenshot.png	133.5 kB	PNG image	Tue 07 May 2013 11:05:32 PM CEST
trace.txt	131.8 kB	plain text document	Tue 07 May 2013 11:05:40 PM CEST
traffic.cap	708.3 kB	Network Packet Capture	Tue 07 May 2013 11:05:48 PM CEST
traffic.xml	452.6 kB	XML document	Tue 07 May 2013 11:05:42 PM CEST

Figure 46: Sample of the information extracted for each particular application

Listing A.1: Sample of ARFF file with extracted features

```
@relation maliciousAndBenignAppFeatures

@attribute id_featureSet numeric
@attribute id_app numeric
@attribute id_test numeric
@attribute sdkVersion numeric
@attribute targetSdkVersion numeric
@attribute app_label_length numeric
@attribute package_name_length numeric
@attribute filesize numeric
@attribute permissions_highest numeric
@attribute permissions_avg numeric
@attribute permissions_number numeric
@attribute pull_data_size numeric
@attribute log_launch_size numeric
@attribute cpu_usage_peak numeric
@attribute cpu_usage_avg numeric
@attribute cpu_usage_stdev numeric
@attribute thr_usage_peak numeric
@attribute thr_usage_avg numeric
@attribute thr_usage_stdev numeric
@attribute vss_usage_peak numeric
@attribute vss_usage_avg numeric
@attribute vss_usage_stdev numeric
@attribute rss_usage_peak numeric
@attribute rss_usage_avg numeric
@attribute rss_usage_stdev numeric
@attribute shared_prefs numeric
@attribute shared_prefs_size numeric
@attribute databases numeric
@attribute databases_size numeric
@attribute files numeric
@attribute files_size numeric
@attribute package_entropy numeric
@attribute package_number_files numeric
@attribute manifest_size numeric
@attribute res_folder_size numeric
@attribute assets_folder_size numeric
@attribute classes_dex_size numeric
@attribute classes_dex_entropy numeric
@attribute execution_time numeric
@attribute class {0,1}

@data
1, 201, 891, 4, 4, 19, 10, 82174, 2, 0.428571, 7, 12, 19080, 0,
0, 0, 9, 9, 0,
0.481159, 0.481159, 0, 0.025627, 0.025627, 0, 0, 0, 0, 0, 2, 8,
6.38049, 27,
4120, 144, 0, 40276, 5.61094, 85.85, 1

  2, 450, 890, 4, 4, 19, 10, 103945, 2, 0.428571, 7, 12, 18526, 0,
   0, 0, 9,
8.57143, 1.91663, 0.481152, 0.45824, 0.102465, 0.025597, 0.024378,
 0.005451, 0,
0, 0, 0, 2, 8, 6.67897, 27, 4120, 168, 0, 40276, 5.61094, 84.49, 1

  3, 449, 889, 4, 4, 19, 10, 131978, 2, 0.428571, 7, 12, 14490, 0,
   0, 0, 9, 9,
0, 0.481159, 0.481159, 0, 0.025612, 0.025612, 0, 0, 0, 0, 0, 2, 8,
 6.86501, 27,
4120, 192, 0, 40276, 5.61094, 83.03, 1

  4, 201, 891, 4, 4, 19, 10, 82174, 2, 0.428571, 7, 12, 19080, 0,
  0, 0, 9, 9, 0,
0.481159, 0.481159, 0, 0.025627, 0.025627, 0, 0, 0, 0, 0, 2, 8,
6.38046, 27,
4120, 144, 0, 40276, 5.61094, 85.85, 1

  5, 450, 890, 4, 4, 19, 10, 103945, 2, 0.428571, 7, 12, 18526, 0,
   0, 0, 9,
8.57143, 1.91663, 0.481152, 0.45824, 0.102465, 0.025597, 0.024378,
 0.005451, 0,
0, 0, 0, 2, 8, 6.67895, 27, 4120, 168, 0, 40276, 5.61094, 84.49, 1

  6, 449, 889, 4, 4, 19, 10, 131978, 2, 0.428571, 7, 12, 14490, 0,
   0, 0, 9, 9,
0, 0.481159, 0.481159, 0, 0.025612, 0.025612, 0, 0, 0, 0, 0, 2, 8,
 6.865, 27,
4120, 192, 0, 40276, 5.61094, 83.03, 1

  7, 464, 898, 4, 4, 19, 10, 797581, 2, 0.272727, 12, 16, 18480,
  2, 0.25,
0.622495, 9, 9, 0, 0.489651, 0.488813, 0.002514, 0.028187,
0.027884, 0.000454,
3, 12, 0, 0, 0, 0, 5.54015, 603, 4684, 2420, 0, 49400, 5.73651,
99.49, 1

  8, 208, 897, 4, 4, 19, 10, 95298, 2, 0.428571, 7, 12, 22778, 0,
  0, 0, 9, 9, 0,
0.481159, 0.481159, 0, 0.025612, 0.025612, 0, 0, 0, 0, 0, 2, 8,
6.61806, 27,
4120, 168, 0, 40276, 5.61094, 84.64, 1

  9, 463, 896, 4, 4, 19, 10, 92111, 2, 0.428571, 7, 12, 14560, 0,
  0, 0, 9, 9, 0,
0.481152, 0.481152, 0, 0.0256, 0.0256, 0, 0, 0, 0, 0, 2, 8,
6.51616, 27, 4120,
168, 0, 40276, 5.61094, 86.1, 1

  10, 462, 895, 4, 4, 19, 10, 136607, 2, 0.428571, 7, 12, 14348,
  0, 0, 0, 9, 9,
0, 0.481152, 0.481152, 0, 0.025616, 0.025616, 0, 0, 0, 0, 0, 2, 8,
 6.98069, 27,
4120, 216, 0, 40276, 5.61094, 82.71, 1

  11, 459, 894, 4, 4, 19, 10, 131910, 2, 0.428571, 7, 12, 22501,
  0, 0, 0, 9,
8.57143, 1.91663, 0.481159, 0.458247, 0.102467, 0.025604,
0.024385, 0.005453, 0,
0, 0, 0, 2, 8, 6.87135, 27, 4120, 192, 0, 40276, 5.61094, 84, 1

  12, 458, 893, 4, 4, 19, 10, 128363, 2, 0.428571, 7, 12, 18799,
  0, 0, 0, 9, 9,
0, 0.481152, 0.481152, 0, 0.025608, 0.025608, 0, 0, 0, 0, 0, 2, 8,
 6.77124, 27,
4120, 192, 0, 40276, 5.61094, 83.18, 1

  13, 457, 892, 4, 4, 19, 10, 88928, 2, 0.428571, 7, 12, 14637, 0,
   0, 0, 9, 9,
0, 0.481152, 0.481152, 0, 0.025597, 0.025597, 0, 0, 0, 0, 0, 2, 8,
 6.414, 27,
4120, 168, 0, 40276, 5.61094, 86.09, 1

  14, 201, 891, 4, 4, 19, 10, 82174, 2, 0.428571, 7, 12, 19080, 0,
   0, 0, 9, 9,
0, 0.481159, 0.481159, 0, 0.025627, 0.025627, 0, 0, 0, 0, 0, 2, 8,
 6.3804, 27,
4120, 144, 0, 40276, 5.61094, 85.85, 1

  15, 450, 890, 4, 4, 19, 10, 103945, 2, 0.428571, 7, 12, 18526,
  0, 0, 0, 9,
8.57143, 1.91663, 0.481152, 0.45824, 0.102465, 0.025597, 0.024378,
 0.005451, 0,
0, 0, 0, 2, 8, 6.6789, 27, 4120, 168, 0, 40276, 5.61094, 84.49, 1

  16, 449, 889, 4, 4, 19, 10, 131978, 2, 0.428571, 7, 12, 14490,
  0, 0, 0, 9, 9,
0, 0.481159, 0.481159, 0, 0.025612, 0.025612, 0, 0, 0, 0, 0, 2, 8,
 6.86497, 27,
4120, 192, 0, 40276, 5.61094, 83.03, 1

  17, 448, 888, 4, 4, 19, 10, 103138, 2, 0.428571, 7, 12, 22635,
  0, 0, 0, 9,
8.57143, 1.91663, 0.481159, 0.458247, 0.102467, 0.025581,
0.024363, 0.005448, 0,
0, 0, 0, 2, 8, 6.63956, 27, 4120, 168, 0, 40276, 5.61094, 84.13, 1
```

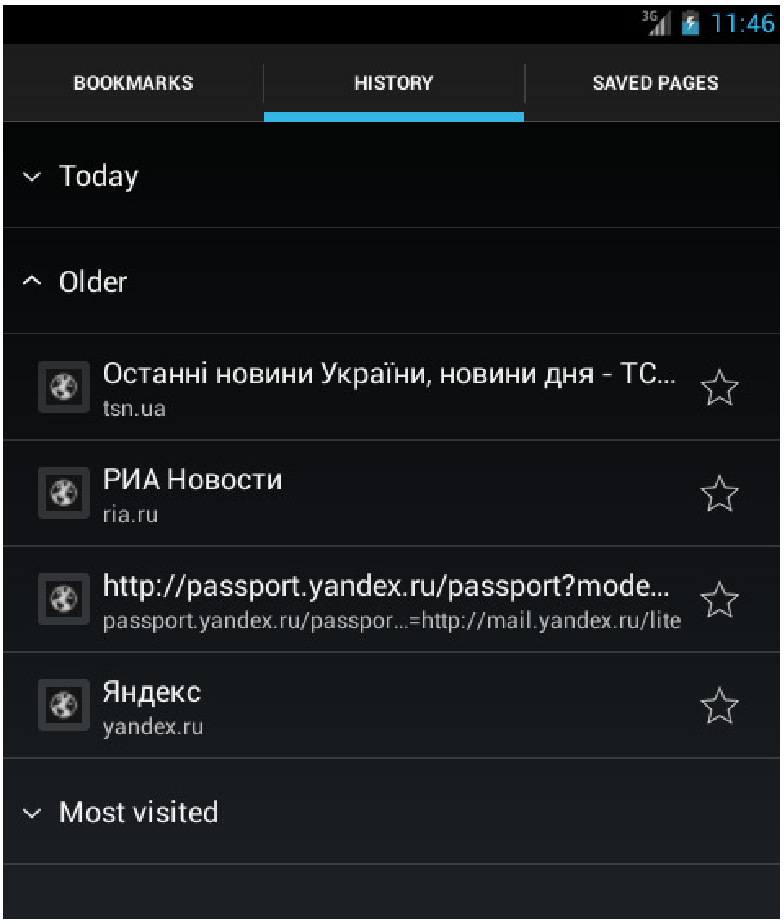
```
  18, 447, 887, 4, 4, 19, 10, 90538, 2, 0.428571, 7, 12, 18898, 0,
    0, 0, 11, 11,
0, 0.483143, 0.483143, 0, 0.025642, 0.025642, 0, 0, 0, 0, 0, 2, 8,
 6.44543, 27,
4120, 168, 0, 40276, 5.61094, 83.95, 1

  19, 445, 886, 4, 4, 19, 10, 113902, 2, 0.428571, 7, 12, 22764,
  0, 0, 0, 9, 9,
0, 0.481152, 0.481152, 0, 0.025616, 0.025616, 0, 0, 0, 0, 0, 2, 8,
 6.76884, 27,
4120, 192, 0, 40276, 5.61094, 85.48, 1

  20, 444, 885, 4, 4, 19, 10, 101622, 2, 0.428571, 7, 12, 14855,
  0, 0, 0, 9, 9,
0, 0.481152, 0.481152, 0, 0.025604, 0.025604, 0, 0, 0, 0, 0, 2, 8,
 6.60965, 27,
4120, 168, 0, 40276, 5.61094, 84.5, 1
```

Listing A.2: Sample of ARFF file with derived security metrics

```
@relation maliciousAndBenignAppSecurityMetrics

@attribute METRICpermissions numeric
@attribute METRICstatic numeric
@attribute METRICsdk numeric
@attribute METRICresources numeric
@attribute METRICdynamic numeric
@attribute class {0,1}

@data
0.373286,4040.440198,0.432,0.101561,267.60125,1
0.373286,4672.347495,0.432,0.101176,259.83845,1
0.373286,5485.845146,0.432,0.101561,203.32715,1
0.373286,4040.440196,0.432,0.101561,267.60125,1
0.373286,4672.347494,0.432,0.101176,259.83845,1
0.373286,5485.845145,0.432,0.101561,203.32715,1
0.512364,25206.584083,0.432,0.106987,259.34045,1
0.373286,4421.580353,0.432,0.101561,319.3672,1
0.373286,4329.150424,0.432,0.10156,204.3225,1
0.373286,5620.622012,0.432,0.101561,201.33755,1
0.373286,5483.873577,0.432,0.101176,315.486,1
0.373286,5381.003769,0.432,0.101561,263.6539,1
0.373286,4236.836477,0.432,0.10156,205.40045,1
0.373286,4040.440192,0.432,0.101561,267.60125,1
0.373286,4672.34749,0.432,0.101176,259.83845,1
0.373286,5485.845143,0.432,0.101561,203.32715,1
0.373286,4648.941815,0.432,0.101176,317.36265,1
0.373286,4283.528614,0.432,0.117681,265.04375,1
0.373286,4961.634606,0.432,0.101561,319.1754,1
0.373286,4604.975781,0.432,0.10156,208.4445,1
```

B User profile's details

For purposes of examination of pulled from dynamic analysis data, we developed following artificial information. This information is considered to be sensitive in the real world. This information was entered in the most popular and the most used applications on mobile platform:



Останні новини України, новини дня - ТС...
tsn.ua

РИА Новости
ria.ru

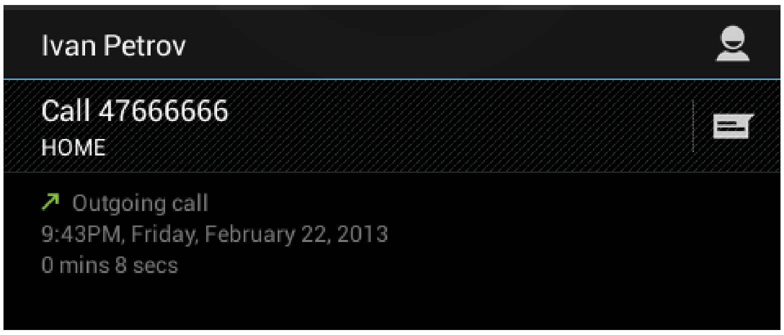
http://passport.yandex.ru/passport?mode...
passport.yandex.ru/passpor...=http://mail.yandex.ru/lite

Яндекс
yandex.ru

Figure 47: Sample of a browser history in Android

In the Figure 52 the output of the tree utility[1] for extracted information from userdata.img is presented.

[1] `http://linux.die.net/man/1/tree`

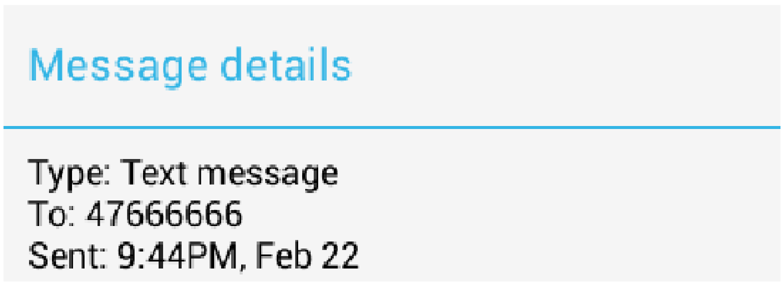


Ivan Petrov

Call 47666666
HOME

Outgoing call
9:43PM, Friday, February 22, 2013
0 mins 8 secs

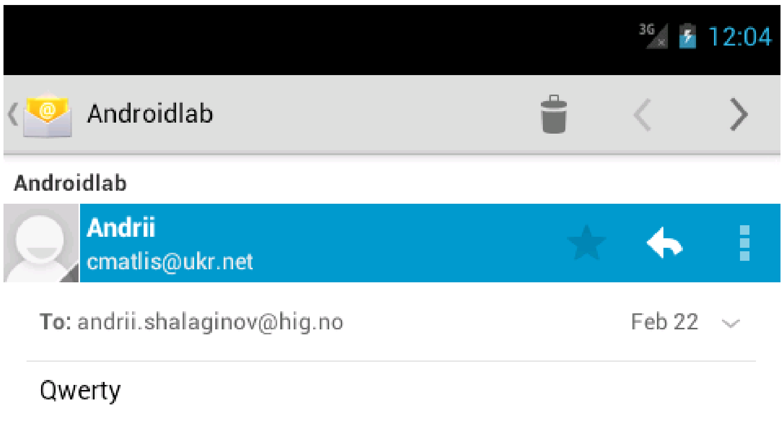
Figure 48: History of performed user calls



Message details

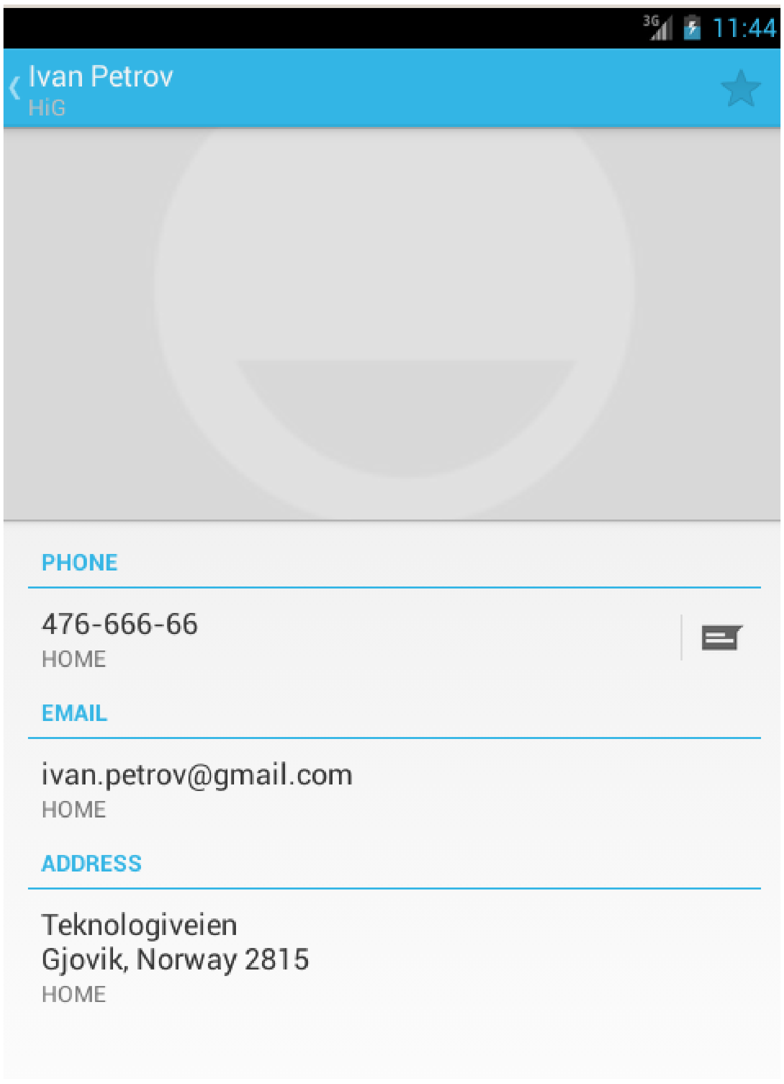
Type: Text message
To: 47666666
Sent: 9:44PM, Feb 22

Figure 49: User messsages



Androidlab
Andrii
cmatlis@ukr.net
To: andrii.shalaginov@hig.no Feb 22
Qwerty

Figure 50: Sent emails



PHONE

476-666-66
HOME

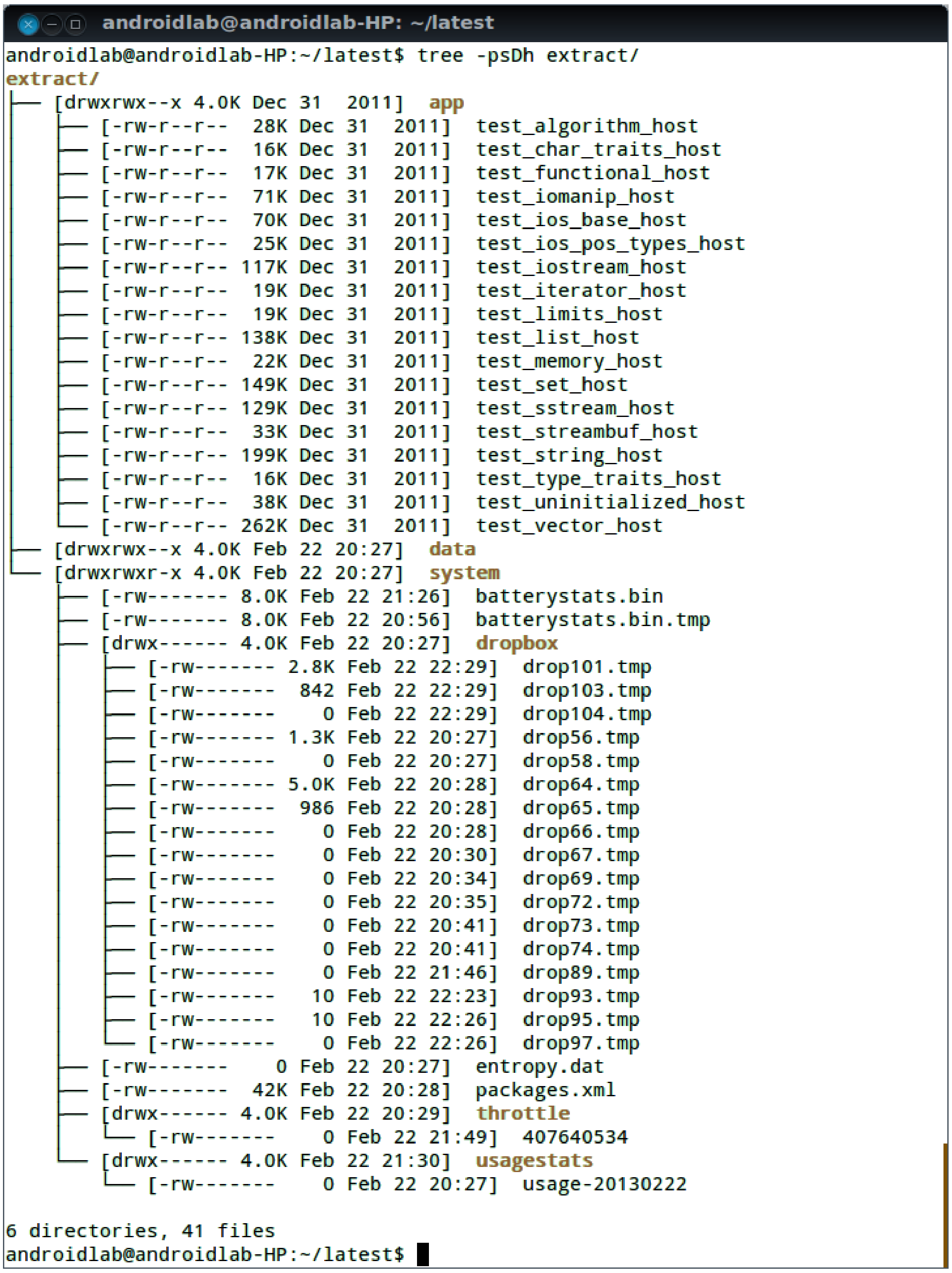
EMAIL

ivan.petrov@gmail.com
HOME

ADDRESS

Teknologiveien
Gjovik, Norway 2815
HOME

Figure 51: Contacts stored on mobile devices

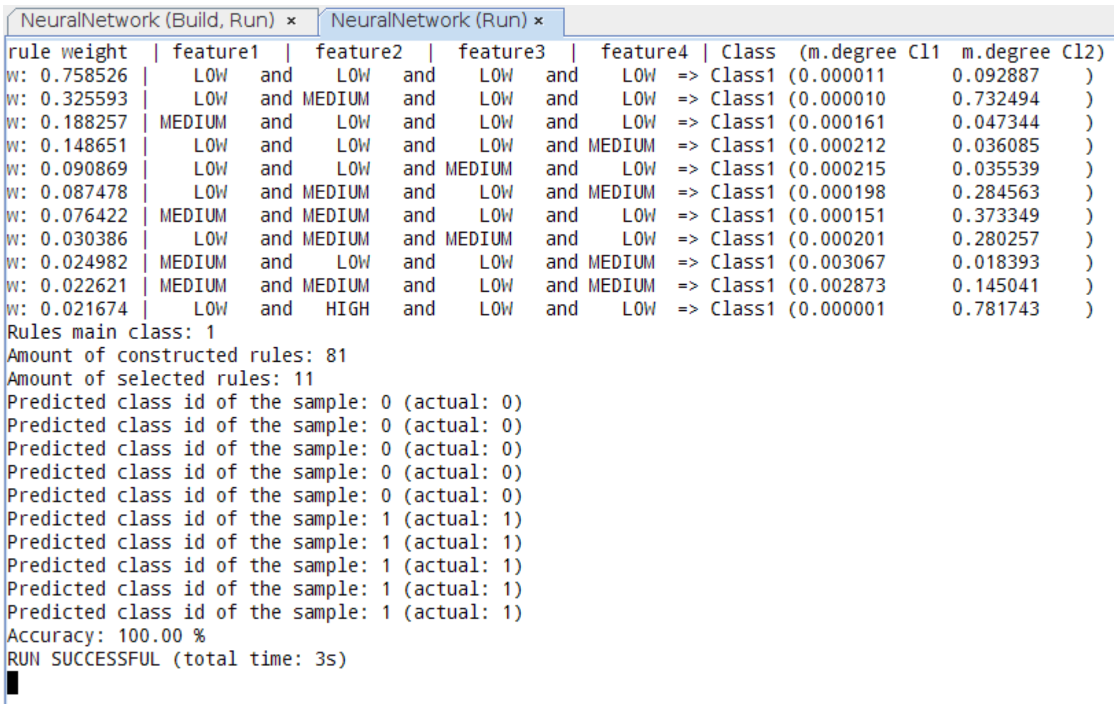


```
androidlab@androidlab-HP:~/latest$ tree -psDh extract/
extract/
├── [drwxrwx--x 4.0K Dec 31  2011]  app
│   ├── [-rw-r--r--  28K Dec 31  2011]  test_algorithm_host
│   ├── [-rw-r--r--  16K Dec 31  2011]  test_char_traits_host
│   ├── [-rw-r--r--  17K Dec 31  2011]  test_functional_host
│   ├── [-rw-r--r--  71K Dec 31  2011]  test_iomanip_host
│   ├── [-rw-r--r--  70K Dec 31  2011]  test_ios_base_host
│   ├── [-rw-r--r--  25K Dec 31  2011]  test_ios_pos_types_host
│   ├── [-rw-r--r-- 117K Dec 31  2011]  test_iostream_host
│   ├── [-rw-r--r--  19K Dec 31  2011]  test_iterator_host
│   ├── [-rw-r--r--  19K Dec 31  2011]  test_limits_host
│   ├── [-rw-r--r-- 138K Dec 31  2011]  test_list_host
│   ├── [-rw-r--r--  22K Dec 31  2011]  test_memory_host
│   ├── [-rw-r--r-- 149K Dec 31  2011]  test_set_host
│   ├── [-rw-r--r-- 129K Dec 31  2011]  test_sstream_host
│   ├── [-rw-r--r--  33K Dec 31  2011]  test_streambuf_host
│   ├── [-rw-r--r-- 199K Dec 31  2011]  test_string_host
│   ├── [-rw-r--r--  16K Dec 31  2011]  test_type_traits_host
│   ├── [-rw-r--r--  38K Dec 31  2011]  test_uninitialized_host
│   └── [-rw-r--r-- 262K Dec 31  2011]  test_vector_host
├── [drwxrwx--x 4.0K Feb 22 20:27]  data
└── [drwxrwxr-x 4.0K Feb 22 20:27]  system
    ├── [-rw------- 8.0K Feb 22 21:26]  batterystats.bin
    ├── [-rw------- 8.0K Feb 22 20:56]  batterystats.bin.tmp
    ├── [drwx------ 4.0K Feb 22 20:27]  dropbox
    │   ├── [-rw------- 2.8K Feb 22 22:29]  drop101.tmp
    │   ├── [-rw-------  842 Feb 22 22:29]  drop103.tmp
    │   ├── [-rw-------    0 Feb 22 22:29]  drop104.tmp
    │   ├── [-rw------- 1.3K Feb 22 20:27]  drop56.tmp
    │   ├── [-rw-------    0 Feb 22 20:27]  drop58.tmp
    │   ├── [-rw------- 5.0K Feb 22 20:28]  drop64.tmp
    │   ├── [-rw-------  986 Feb 22 20:28]  drop65.tmp
    │   ├── [-rw-------    0 Feb 22 20:28]  drop66.tmp
    │   ├── [-rw-------    0 Feb 22 20:30]  drop67.tmp
    │   ├── [-rw-------    0 Feb 22 20:34]  drop69.tmp
    │   ├── [-rw-------    0 Feb 22 20:35]  drop72.tmp
    │   ├── [-rw-------    0 Feb 22 20:41]  drop73.tmp
    │   ├── [-rw-------    0 Feb 22 20:41]  drop74.tmp
    │   ├── [-rw-------    0 Feb 22 21:46]  drop89.tmp
    │   ├── [-rw-------   10 Feb 22 22:23]  drop93.tmp
    │   ├── [-rw-------   10 Feb 22 22:26]  drop95.tmp
    │   └── [-rw-------    0 Feb 22 22:26]  drop97.tmp
    ├── [-rw-------    0 Feb 22 20:27]  entropy.dat
    ├── [-rw-------  42K Feb 22 20:28]  packages.xml
    ├── [drwx------ 4.0K Feb 22 20:29]  throttle
    │   └── [-rw-------    0 Feb 22 21:49]  407640534
    └── [drwx------ 4.0K Feb 22 21:30]  usagestats
        └── [-rw-------    0 Feb 22 20:27]  usage-20130222

6 directories, 41 files
androidlab@androidlab-HP:~/latest$
```

Figure 52: Structure of the extracted information from userdata.img

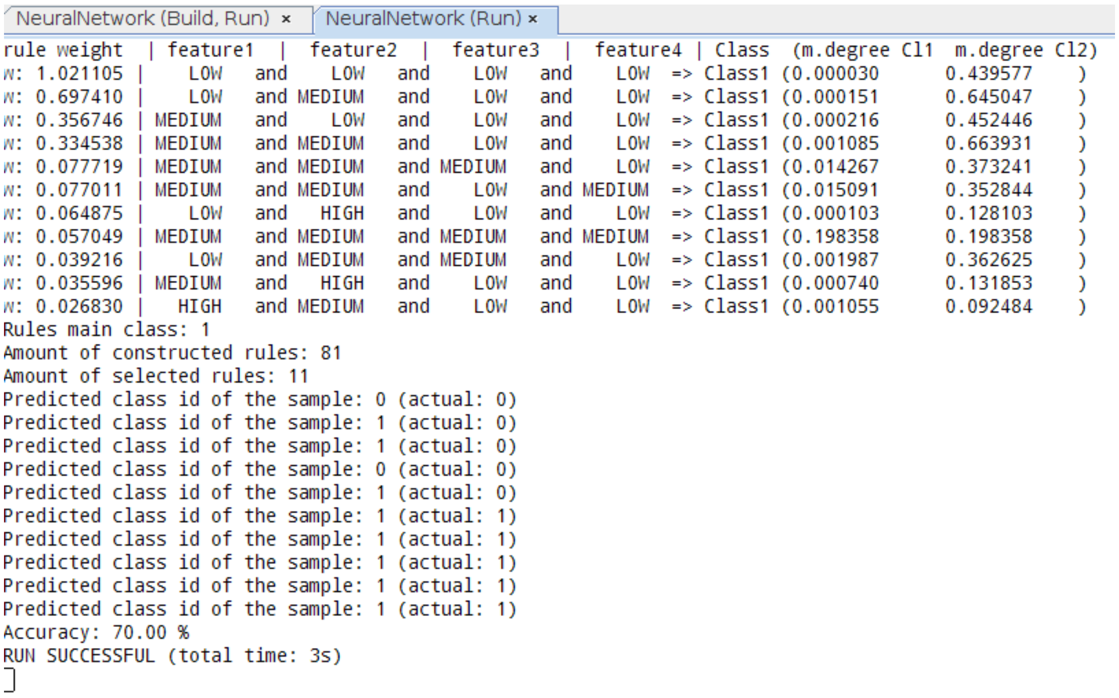
C Extracted rules for proof-of-concept experiment

Examples of extracted rules from Iris Data set processing are shown below. There were used three and five terms in each linguistic variable)security metrics).



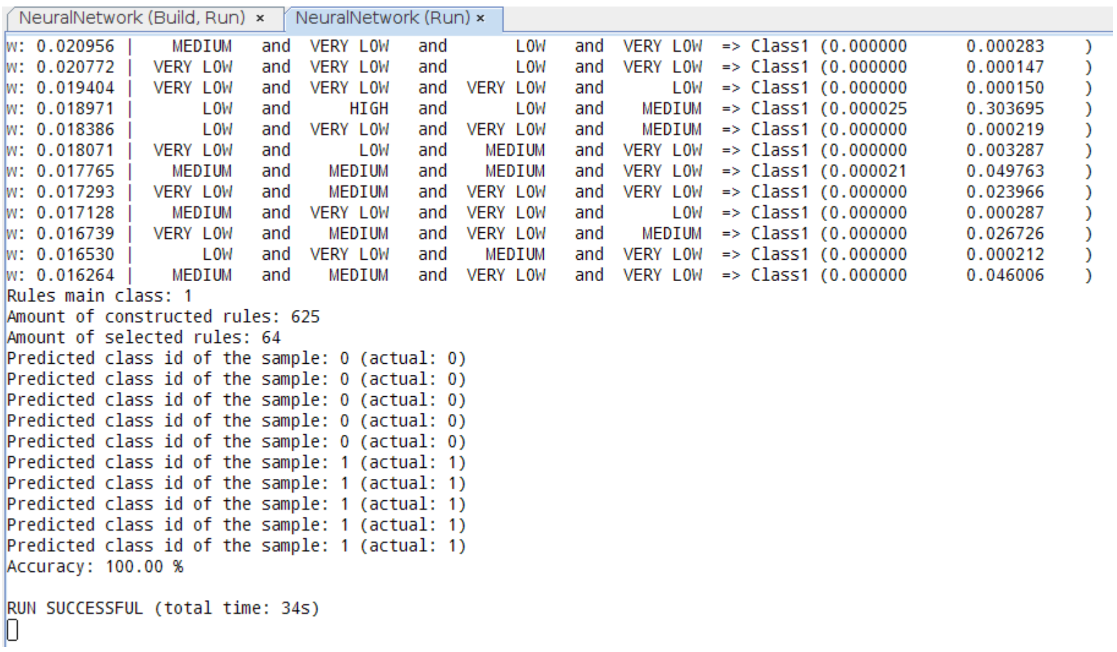
```
rule weight   | feature1  |   feature2  |   feature3  |   feature4 | Class  (m.degree Cl1  m.degree Cl2)
w: 0.758526 |     LOW   and     LOW   and     LOW   and     LOW   => Class1 (0.000011      0.092887     )
w: 0.325593 |     LOW   and MEDIUM   and     LOW   and     LOW   => Class1 (0.000010      0.732494     )
w: 0.188257 | MEDIUM   and     LOW   and     LOW   and     LOW   => Class1 (0.000161      0.047344     )
w: 0.148651 |     LOW   and     LOW   and     LOW   and MEDIUM   => Class1 (0.000212      0.036085     )
w: 0.090869 |     LOW   and     LOW   and MEDIUM   and     LOW   => Class1 (0.000215      0.035539     )
w: 0.087478 |     LOW   and MEDIUM   and     LOW   and MEDIUM   => Class1 (0.000198      0.284563     )
w: 0.076422 | MEDIUM   and MEDIUM   and     LOW   and     LOW   => Class1 (0.000151      0.373349     )
w: 0.030386 |     LOW   and MEDIUM   and MEDIUM   and     LOW   => Class1 (0.000201      0.280257     )
w: 0.024982 | MEDIUM   and     LOW   and     LOW   and MEDIUM   => Class1 (0.003067      0.018393     )
w: 0.022621 | MEDIUM   and MEDIUM   and     LOW   and MEDIUM   => Class1 (0.002873      0.145041     )
w: 0.021674 |     LOW   and    HIGH   and     LOW   and     LOW   => Class1 (0.000001      0.781743     )
Rules main class: 1
Amount of constructed rules: 81
Amount of selected rules: 11
Predicted class id of the sample: 0 (actual: 0)
Predicted class id of the sample: 0 (actual: 0)
Predicted class id of the sample: 0 (actual: 0)
Predicted class id of the sample: 0 (actual: 0)
Predicted class id of the sample: 0 (actual: 0)
Predicted class id of the sample: 1 (actual: 1)
Predicted class id of the sample: 1 (actual: 1)
Predicted class id of the sample: 1 (actual: 1)
Predicted class id of the sample: 1 (actual: 1)
Predicted class id of the sample: 1 (actual: 1)
Accuracy: 100.00 %
RUN SUCCESSFUL (total time: 3s)
```

Figure 53: Fuzzy rules for Setosa-Virginica classification problem, three terms in each linguistic variable



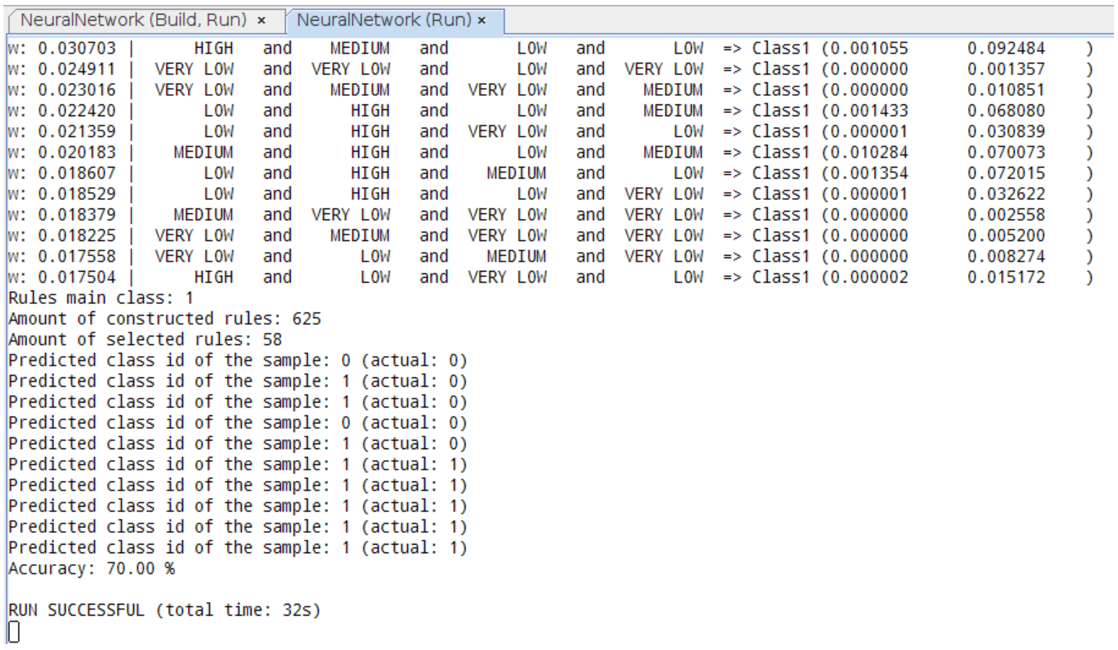
```
rule weight   | feature1  |  feature2  |  feature3  |  feature4 | Class  (m.degree Cl1  m.degree Cl2)
w: 1.021105 |    LOW   and    LOW   and    LOW   and    LOW  => Class1 (0.000030      0.439577     )
w: 0.697410 |    LOW   and MEDIUM   and    LOW   and    LOW  => Class1 (0.000151      0.645047     )
w: 0.356746 | MEDIUM   and    LOW   and    LOW   and    LOW  => Class1 (0.000216      0.452446     )
w: 0.334538 | MEDIUM   and MEDIUM   and    LOW   and    LOW  => Class1 (0.001085      0.663931     )
w: 0.077719 | MEDIUM   and MEDIUM   and MEDIUM   and    LOW  => Class1 (0.014267      0.373241     )
w: 0.077011 | MEDIUM   and MEDIUM   and    LOW   and MEDIUM  => Class1 (0.015091      0.352844     )
w: 0.064875 |    LOW   and   HIGH   and    LOW   and    LOW  => Class1 (0.000103      0.128103     )
w: 0.057049 | MEDIUM   and MEDIUM   and MEDIUM   and MEDIUM  => Class1 (0.198358      0.198358     )
w: 0.039216 |    LOW   and MEDIUM   and MEDIUM   and    LOW  => Class1 (0.001987      0.362625     )
w: 0.035596 | MEDIUM   and   HIGH   and    LOW   and    LOW  => Class1 (0.000740      0.131853     )
w: 0.026830 |   HIGH   and MEDIUM   and    LOW   and    LOW  => Class1 (0.001055      0.092484     )
Rules main class: 1
Amount of constructed rules: 81
Amount of selected rules: 11
Predicted class id of the sample: 0 (actual: 0)
Predicted class id of the sample: 1 (actual: 0)
Predicted class id of the sample: 1 (actual: 0)
Predicted class id of the sample: 0 (actual: 0)
Predicted class id of the sample: 1 (actual: 0)
Predicted class id of the sample: 1 (actual: 1)
Predicted class id of the sample: 1 (actual: 1)
Predicted class id of the sample: 1 (actual: 1)
Predicted class id of the sample: 1 (actual: 1)
Predicted class id of the sample: 1 (actual: 1)
Accuracy: 70.00 %
RUN SUCCESSFUL (total time: 3s)
```

Figure 54: Fuzzy rules for Versicolor-Virginica classification problem, three terms in each linguistic variable



```
w: 0.020956 |    MEDIUM   and  VERY LOW   and       LOW   and  VERY LOW  => Class1 (0.000000      0.000283     )
w: 0.020772 |  VERY LOW   and  VERY LOW   and       LOW   and  VERY LOW  => Class1 (0.000000      0.000147     )
w: 0.019404 |  VERY LOW   and  VERY LOW   and  VERY LOW   and       LOW  => Class1 (0.000000      0.000150     )
w: 0.018971 |       LOW   and      HIGH   and       LOW   and    MEDIUM  => Class1 (0.000025      0.303695     )
w: 0.018386 |       LOW   and  VERY LOW   and  VERY LOW   and    MEDIUM  => Class1 (0.000000      0.000219     )
w: 0.018071 |  VERY LOW   and       LOW   and    MEDIUM   and  VERY LOW  => Class1 (0.000000      0.003287     )
w: 0.017765 |    MEDIUM   and    MEDIUM   and    MEDIUM   and  VERY LOW  => Class1 (0.000021      0.049763     )
w: 0.017293 |  VERY LOW   and    MEDIUM   and  VERY LOW   and  VERY LOW  => Class1 (0.000000      0.023966     )
w: 0.017128 |    MEDIUM   and  VERY LOW   and  VERY LOW   and       LOW  => Class1 (0.000000      0.000287     )
w: 0.016739 |  VERY LOW   and    MEDIUM   and  VERY LOW   and    MEDIUM  => Class1 (0.000000      0.026726     )
w: 0.016530 |       LOW   and  VERY LOW   and    MEDIUM   and  VERY LOW  => Class1 (0.000000      0.000212     )
w: 0.016264 |    MEDIUM   and    MEDIUM   and  VERY LOW   and  VERY LOW  => Class1 (0.000000      0.046006     )
Rules main class: 1
Amount of constructed rules: 625
Amount of selected rules: 64
Predicted class id of the sample: 0 (actual: 0)
Predicted class id of the sample: 0 (actual: 0)
Predicted class id of the sample: 0 (actual: 0)
Predicted class id of the sample: 0 (actual: 0)
Predicted class id of the sample: 0 (actual: 0)
Predicted class id of the sample: 1 (actual: 1)
Predicted class id of the sample: 1 (actual: 1)
Predicted class id of the sample: 1 (actual: 1)
Predicted class id of the sample: 1 (actual: 1)
Predicted class id of the sample: 1 (actual: 1)
Accuracy: 100.00 %

RUN SUCCESSFUL (total time: 34s)
```

Figure 55: Fuzzy rules for Setosa-Virginica classification problem, five terms in each linguistic variable



```
w: 0.030703 |      HIGH  and    MEDIUM  and       LOW  and       LOW  => Class1 (0.001055      0.092484     )
w: 0.024911 |  VERY LOW  and  VERY LOW  and       LOW  and  VERY LOW  => Class1 (0.000000      0.001357     )
w: 0.023016 |  VERY LOW  and    MEDIUM  and  VERY LOW  and    MEDIUM  => Class1 (0.000000      0.010851     )
w: 0.022420 |       LOW  and      HIGH  and       LOW  and    MEDIUM  => Class1 (0.001433      0.068080     )
w: 0.021359 |       LOW  and      HIGH  and  VERY LOW  and       LOW  => Class1 (0.000001      0.030839     )
w: 0.020183 |    MEDIUM  and      HIGH  and       LOW  and    MEDIUM  => Class1 (0.010284      0.070073     )
w: 0.018607 |       LOW  and      HIGH  and    MEDIUM  and       LOW  => Class1 (0.001354      0.072015     )
w: 0.018529 |       LOW  and      HIGH  and       LOW  and  VERY LOW  => Class1 (0.000001      0.032622     )
w: 0.018379 |    MEDIUM  and  VERY LOW  and  VERY LOW  and  VERY LOW  => Class1 (0.000000      0.002558     )
w: 0.018225 |  VERY LOW  and    MEDIUM  and  VERY LOW  and  VERY LOW  => Class1 (0.000000      0.005200     )
w: 0.017558 |  VERY LOW  and       LOW  and    MEDIUM  and  VERY LOW  => Class1 (0.000000      0.008274     )
w: 0.017504 |      HIGH  and       LOW  and  VERY LOW  and       LOW  => Class1 (0.000002      0.015172     )
Rules main class: 1
Amount of constructed rules: 625
Amount of selected rules: 58
Predicted class id of the sample: 0 (actual: 0)
Predicted class id of the sample: 1 (actual: 0)
Predicted class id of the sample: 1 (actual: 0)
Predicted class id of the sample: 0 (actual: 0)
Predicted class id of the sample: 1 (actual: 0)
Predicted class id of the sample: 1 (actual: 1)
Predicted class id of the sample: 1 (actual: 1)
Predicted class id of the sample: 1 (actual: 1)
Predicted class id of the sample: 1 (actual: 1)
Predicted class id of the sample: 1 (actual: 1)
Accuracy: 70.00 %

RUN SUCCESSFUL (total time: 32s)
```

Figure 56: Fuzzy rules for Versicolor-Virginica classification problem, five terms in each linguistic variable

D Android application launch logs

Listing D.1: Sample of launch log of the Android application

```
D/AndroidRuntime( 1735): >>>>>> AndroidRuntime START
com.android.internal.os.RuntimeInit <<<<<<
D/AndroidRuntime( 1735): CheckJNI is ON
D/dalvikvm( 1735): Trying to load lib libjavacore.so 0x0
D/dalvikvm( 1735): Added shared lib libjavacore.so 0x0
D/dalvikvm( 1735): Trying to load lib libnativehelper.so 0x0
D/dalvikvm( 1735): Added shared lib libnativehelper.so 0x0
D/AndroidRuntime( 1735): Calling main entry com.android.commands.
am.Am
D/dalvikvm( 1735): Note: class Landroid/app/ActivityManagerNative;
 has 156
unimplemented (abstract) methods
I/ActivityManager( 1200): START u0 {flg=0x10000000
cmp=com.example.android.service/.Main} from pid 1735
D/PermissionCache(  786): checking android.permission.
READ_FRAME_BUFFER for
uid=1000 => granted (1292 us)
D/dalvikvm(  787): WAIT_FOR_CONCURRENT_GC blocked 1ms
I/ActivityManager( 1200): Start proc com.example.android.service
for activity
com.example.android.service/.Main: pid=1746 uid=10003 gids={50003,
 3003, 1015,
1028}
D/AndroidRuntime( 1735): Shutting down VM
D/dalvikvm( 1735): GC_CONCURRENT freed 95K, 17% free 483K/580K,
paused 0ms+0ms,
total 2ms
E/SurfaceFlinger(  786): ro.sf.lcd_density must be defined as a
build property
D/        (  786): HostConnection::get() New Host Connection
established
0xb8ea85e0, tid 877
D/dalvikvm(  787): GC_EXPLICIT freed 36K, 7% free 2323K/2472K,
paused 0ms+1ms,
total 107ms
D/dalvikvm(  787): GC_EXPLICIT freed <1K, 6% free 2323K/2472K,
paused 0ms+0ms,
total 52ms
E/Trace   ( 1746): error opening trace file: No such file or
directory (2)
```

```
D/dalvikvm(  787): GC_EXPLICIT freed <1K, 6% free 2323K/2472K,
paused 1ms+17ms,
total 18ms
D/        (  786): HostConnection::get() New Host Connection
established
0xb8ea8c00, tid 863
V/PhoneStatusBar( 1270): setLightsOn(true)
E/SurfaceFlinger(  786): ro.sf.lcd_density must be defined as a
build property
W/IInputConnectionWrapper( 1354): showStatusIcon on inactive
InputConnection
D/        ( 1746): HostConnection::get() New Host Connection
established
0xb8733960, tid 1746
I/ActivityManager( 1200): Displayed com.example.android.service/.
Main: +922ms
E/WVMExtractor(  789): Failed to open libwvm.so
D/AudioSink(  789): bufferCount (4) is too small and increased to
12
I/ActivityManager( 1200): START u0 {act=android.intent.action.VIEW
dat=http://14243444.com/send.php?a_id=000000000000000&telno
=15555215554&m_addr=
cmp=com.android.browser/.BrowserActivity} from pid 1746
D/dalvikvm( 1200): GC_FOR_ALLOC freed 236K, 11% free 4481K/4996K,
paused 14ms,
total 18ms
E/SurfaceFlinger(  786): ro.sf.lcd_density must be defined as a
build property
I/ActivityManager( 1200): moveTaskToBack: 3
I/ActivityManager( 1200): Start proc com.android.browser for
activity
com.android.browser/.BrowserActivity: pid=1790 uid=10025 gids
={50025, 3003,
1015, 1028}
E/Trace   ( 1790): error opening trace file: No such file or
directory (2)
I/ActivityThread( 1790): Pub com.android.browser;browser:
com.android.browser.provider.BrowserProvider2
I/ActivityThread( 1790): Pub com.android.browser.home:
com.android.browser.homepages.HomeProvider
I/ActivityThread( 1790): Pub com.android.browser.snapshots:
com.android.browser.provider.SnapshotProvider
W/ApplicationContext( 1790): Unable to create external files
directory
D/dalvikvm( 1790): GC_CONCURRENT freed 174K, 11% free 2537K/2832K,
 paused
8ms+0ms, total 20ms
E/ActivityThread( 1790): Failed to find provider info for com.
google.settings
```

```
E/ActivityThread( 1790): Failed to find provider info for com.
google.settings
D/dalvikvm( 1790): GC_FOR_ALLOC freed 150K, 9% free 2766K/3032K,
paused 11ms,
total 12ms
W/BrowserProvider( 1790): Upgrading database from version 23 to 24
E/SQLiteLog( 1790): (1) no such table: _sync_state_metadata
D/TilesManager( 1790): Starting TG #0, 0xb875ae60
D/dalvikvm( 1790): GC_CONCURRENT freed 74K, 7% free 3079K/3308K,
paused 8ms+0ms,
total 13ms
D/WebViewTimersControl( 1790): onBrowserActivityResume
D/WebViewTimersControl( 1790): Resuming webview timers,
view=com.android.browser.BrowserWebView{a6f48048 VFEDHVCL .F....I.
 0,0-0,0}
V/NFC      ( 1790): this device does not have NFC support
E/SurfaceFlinger(  786): ro.sf.lcd_density must be defined as a
build property
D/libEGL   ( 1790): loaded /system/lib/egl/libEGL_emulation.so
D/         ( 1790): HostConnection::get() New Host Connection
established
0xb87f6c80, tid 1790
D/libEGL   ( 1790): loaded /system/lib/egl/libGLESv1_CM_emulation.
so
D/libEGL   ( 1790): loaded /system/lib/egl/libGLESv2_emulation.so
D/         (  786): HostConnection::get() New Host Connection
established
0xb8eaf470, tid 873
W/EGL_emulation( 1790): eglSurfaceAttrib not implemented
D/OpenGLRenderer( 1790): Enabling debug mode 0
E/SQLiteLog( 1790): (1) no such table: Origins
D/WebKit   ( 1790): ERROR:
D/WebKit   ( 1790): Application Cache Storage: failed to execute
statement
"DELETE FROM Origins" error "no such table: Origins"
D/WebKit   ( 1790):
D/WebKit   ( 1790):
external/webkit/Source/WebCore/loader/appcache/
ApplicationCacheStorage.cpp(556)
: bool WebCore::ApplicationCacheStorage::executeSQLCommand(const
WTF::String&)
E/SQLiteLog( 1790): (1) no such table: DeletedCacheResources
I/ActivityManager( 1200): Displayed com.android.browser/.
BrowserActivity:
+1s305ms
D/TilesManager( 1790): new EGLContext from framework: b8823790
D/GLWebViewState( 1790): Reinit shader
D/GLWebViewState( 1790): Reinit transferQueue
```

```
W/IInputConnectionWrapper( 1790): showStatusIcon on inactive
InputConnection
D/dalvikvm( 1790): GC_CONCURRENT freed 164K, 8% free 3338K/3620K,
paused
5ms+0ms, total 13ms
D/dalvikvm( 1790): GC_FOR_ALLOC freed 15K, 9% free 3323K/3620K,
paused 9ms,
total 10ms
I/dalvikvm-heap( 1790): Grow heap (frag case) to 4.014MB for
694092-byte
allocation
D/dalvikvm( 1790): GC_FOR_ALLOC freed <1K, 7% free 4001K/4300K,
paused 12ms,
total 13ms
D/dalvikvm( 1790): GC_CONCURRENT freed <1K, 7% free 4001K/4300K,
paused 1ms+0ms,
total 7ms
D/        ( 1790): HostConnection::get() New Host Connection
established
0xb85b5050, tid 1814
D/        (  786): HostConnection::get() New Host Connection
established
0xb8eb0680, tid 869
D/ExchangeService( 1602): Received deviceId from Email app: null
D/ExchangeService( 1602): !!! deviceId unknown; stopping self and
retrying
D/ExchangeService( 1602): !!! EAS ExchangeService, onCreate
D/ExchangeService( 1602): !!! EAS ExchangeService, onStartCommand,
 startingUp =
false, running = false
D/ExchangeService( 1602): !!! EAS ExchangeService, onStartCommand,
 startingUp =
true, running = false
W/ActivityManager( 1200): Unable to start service Intent {
act=com.android.email.ACCOUNT_INTENT } U=0: not found
D/ExchangeService( 1602): !!! Email application not found;
stopping self
E/ActivityThread( 1602): Service com.android.exchange.
ExchangeService has leaked
ServiceConnection
com.android.emailcommon.service.ServiceProxy\$
ProxyConnection@a6f03090 that was
originally bound here
E/ActivityThread( 1602): android.app.ServiceConnectionLeaked:
Service
com.android.exchange.ExchangeService has leaked ServiceConnection
com.android.emailcommon.service.ServiceProxy\$
ProxyConnection@a6f03090 that was
originally bound here
```

```
E/ActivityThread( 1602):  at
android.app.LoadedApk\$ServiceDispatcher.<init>(LoadedApk.java:969
)
E/ActivityThread( 1602):  at
android.app.LoadedApk.getServiceDispatcher(LoadedApk.java:863)
E/ActivityThread( 1602):  at
android.app.ContextImpl.bindService(ContextImpl.java:1418)
E/ActivityThread( 1602):  at
android.app.ContextImpl.bindService(ContextImpl.java:1407)
E/ActivityThread( 1602):  at
android.content.ContextWrapper.bindService(ContextWrapper.java:473
)
E/ActivityThread( 1602):  at
com.android.emailcommon.service.ServiceProxy.setTask(ServiceProxy.
java:157)
E/ActivityThread( 1602):  at
com.android.emailcommon.service.ServiceProxy.setTask(ServiceProxy.
java:145)
E/ActivityThread( 1602):  at
com.android.emailcommon.service.ServiceProxy.test(ServiceProxy.
java:191)
E/ActivityThread( 1602):  at
com.android.exchange.ExchangeService\$7.run(ExchangeService.
java:1850)
E/ActivityThread( 1602):  at
com.android.emailcommon.utility.Utility\$2.doInBackground(Utility.
java:551)
E/ActivityThread( 1602):  at
com.android.emailcommon.utility.Utility\$2.doInBackground(Utility.
java:549)
E/ActivityThread( 1602):  at android.os.AsyncTask\$2.call(
AsyncTask.java:287)
E/ActivityThread( 1602):  at
java.util.concurrent.FutureTask.run(FutureTask.java:234)
E/ActivityThread( 1602):  at
java.util.concurrent.ThreadPoolExecutor.runWorker(
ThreadPoolExecutor.java:1080)
E/ActivityThread( 1602):  at
java.util.concurrent.ThreadPoolExecutor\$Worker.run(
ThreadPoolExecutor.java:573)
E/ActivityThread( 1602):  at java.lang.Thread.run(Thread.java:856)
E/StrictMode( 1602): null
E/StrictMode( 1602): android.app.ServiceConnectionLeaked: Service
com.android.exchange.ExchangeService has leaked ServiceConnection
com.android.emailcommon.service.ServiceProxy\$
ProxyConnection@a6f03090 that was
originally bound here
```

E Features selection for Security Metrics

feature	average merit	average rank
permissions_highest	0.077 ± 0.006	1 ± 0
permissions_avg	0.038 ± 0.003	2 ± 0
permissions_number	0.029 ± 0.003	3 ± 0

Table 14: Calculated feature merits (weights) using RELIEF for 'METRICpermissions' security metric

feature	average merit	average rank
package_entropy	0.068 ± 0.003	1 ± 0
manifest_size	0.059 ± 0.002	2 ± 0
classes_dex_entropy	0.048 ± 0.003	3 ± 0
classes_dex_size	0.035 ± 0.002	4.2 ± 0.4
package_number_files	0.029 ± 0.003	5.3 ± 0.64
filesize	0.029 ± 0.007	5.5 ± 0.67
res_folder_size	0.022 ± 0.002	7 ± 0
assets_folder_size	0.006 ± 0.001	8 ± 0

Table 15: Calculated feature merits (weights) using RELIEF for 'METRICstatic' security metric

feature	average merit	average rank
vss_usage_peak	0.03 ± 0.004	1.2 ± 0.4
vss_usage_avg	0.03 ± 0.004	1.8 ± 0.4
rss_usage_avg	0.015 ± 0.001	3 ± 0
rss_usage_peak	0.012 ± 0.001	4 ± 0
thr_usage_avg	0.004 ± 0.001	5.3 ± 0.46
thr_usage_peak	0.004 ± 0.001	5.7 ± 0.46
rss_usage_stdev	0.003 ± 0	7.4 ± 0.49
cpu_usage_peak	0.002 ± 0.001	7.9 ± 0.94
cpu_usage_stdev	0.001 ± 0	9.5 ± 0.67
thr_usage_stdev	0.001 ± 0	10.5 ± 1.2
vss_usage_stdev	0.001 ± 0	10.6 ± 1.5
cpu_usage_avg	0.001 ± 0	11.1 ± 0.7

Table 16: Calculated feature merits (weights) using RELIEF for 'METRICresources' security metric

F Implemented source code

Listing F.1: Implementation of Android Emulator Start/Stop phases in test_cycle.php

```php
/**
* Launch emulator with defined parameters chosen through the web panel
*/
private function startAVD() {
  global $config_table, $vm_avds_table, $android_sdk_home, $android_adb, $android_user_config, $android_emulator, $vm_avds_table;
  shell_exec($android_sdk_home . "/" . $android_adb . " kill-server");
  sleep(10);
  shell_exec($android_sdk_home . "/" . $android_adb . " devices");
  //Check if there are anny emulators in the memory - otherwise do no show the Launch button
  if (strlen(shell_exec($android_sdk_home . "/" . $android_adb . " devices | sed -n 2p")) <= 1 && strlen(shell_exec("ps -A|grep emulator")) <= 1) {
      //IMPORTANT to create the link
      if (is_dir('/tmp/.android'))
          shell_exec("rm /tmp/.android");
      shell_exec("ln -s " . $android_user_config . " /tmp/");

      //Insert into DB Started Emulator ID
      //Check if AVD is in the runing loop
      if (isset($_POST['avd_to_run'])) {
          $startAVDname = $_POST['avd_to_run'];
          $query = "INSERT INTO $config_table
                            (`item`,`value`) VALUES
                            ('launchedAVD','" . mysql_real_escape_string($startAVDname) . "')
                            ON DUPLICATE KEY UPDATE `value`= '" . mysql_real_escape_string($startAVDname) . "'; ";
          //Check whether it is possible to execute query
          if (mysql_query($query) === FALSE)
              $this->output.="<font color='#CD0000'>Impossible to execute INSERT/UPDATE query!</font> <br>";

          //Insert into DB Started Emulator API
          $query = "INSERT INTO $config_table
```

```
                    (`item`,`value`) VALUES
                    ('launchedAVD_API', (SELECT `api_level`
                    FROM " . $vm_avds_table . "  WHERE `
                    avd_name`='" . mysql_real_escape_string(
                    $startAVDname) . "'))
                    ON DUPLICATE KEY UPDATE `value`= (SELECT
                     `api_level` FROM " . $vm_avds_table . "
                          WHERE `avd_name`='" .
                    mysql_real_escape_string($startAVDname)
                    . "'); ";
        //Check whether it is possible to execute query
        if (mysql_query($query) === FALSE)
            $this->output.="<font color='#CD0000'>Impossible to
            execute INSERT/UPDATE query!</font> <br>";
    }else {
        $query = "SELECT * FROM $config_table WHERE `item`='
        launchedAVD';";
        $res = mysql_query($query);
        $row = mysql_fetch_assoc($res);
        $startAVDname = $row['value'];
    }

    //Read test configuration from DB
    $query = "SELECT `value` FROM $config_table WHERE item='
    testConfiguration'";
    $res = mysql_query($query);
    $row = mysql_fetch_row($res);

    if ($row != FALSE && count($row) > 0) {
        $config_array = unserialize($row[0]);
    }

    //Launch Android Emulator in an detached screen as
    background process
    //-snapstorage <file>              file that contains all
    state snapshots (default <datadir>/snapshots.img)
    //-no-snapstorage                  do not mount a snapshot
    storage file (this disables all snapshot functionality)
    //-snapshot <name>                 name of snapshot within
    storage file for auto-start and auto-save (default 'default-
    boot')
    //-no-snapshot                     perform a full boot and do
    not do not auto-save, but qemu vmload and vmsave operate on
    snapstorage
    //-no-snapshot-save                do not auto-save to
    snapshot on exit: abandon changed state
    //-no-snapshot-load                do not auto-start from
    snapshot: perform a full boot
```

```
      //-snapshot-list                  show a list of available snapshots
      //-no-snapshot-update-time        do not do try to correct snapshot time on restore

      print_r($config_array);

      $avd_parameters = "";
      if (isset($config_array['disable_audio']))
          $avd_parameters.="-noaudio ";
      if (isset($config_array['enable_gpu']))
          $avd_parameters.="-gpu on ";
      if (isset($config_array['disable_boot_anim']))
          $avd_parameters.="-no-boot-anim ";
      if (isset($config_array['disable_window']))
          $avd_parameters.="-no-window ";
      if (isset($config_array['traffic'])) {
          $avd_parameters.="-tcpdump traffic.cap ";
          echo shell_exec("rm " . $android_sdk_home . "/tools/traffic.cap");
      }
      if (isset($config_array['use_profile'])) //IMPORTANT TO UNLOCK DEVICE which producec userdata
          $avd_parameters.="-initdata /home/andymir/websites/androidlab/userdata-qemu.img ";
      //INCREASE time of AVD load, but erase all data, more natural environment
      $avd_parameters.="-no-snapshot -wipe-data ";

      shell_exec("export DISPLAY=:0; screen -d -m " . $android_sdk_home . "/" . $android_emulator . " -avd " . $startAVDname . " " . $avd_parameters);
      //echo $android_sdk_home."/".$android_emulator." -avd ".$_POST['avd_to_run']." ".$avd_parameters;
      $this->output.="Android Emulator was launched successfully! It will be ready in a few minutes!</font>";
  }else
      $this->output.="<font color='#CD0000'>There is an Android Emulator entity in the memory. Try to stop first!</font>";
}

/**
* Stop runnning emulator and existing instances
*/
private function stopAVD() {
  global $android_sdk_home, $android_adb, $config_table;
  //Ensure that adb server is running
  shell_exec($android_sdk_home . "/" . $android_adb . " devices");
  sleep(2);
```

```

    //Kill all running emulators
    echo shell_exec($android_sdk_home . "/" . $android_adb . " emu
    kill");
    echo shell_exec("killall emulator64-x86");
    sleep(5);
    echo shell_exec("killall emulator64-x86");
    echo shell_exec("killall emulator");
    sleep(5);
    echo shell_exec("killall emulator");
    echo shell_exec("killall adb");
    sleep(5);
    echo shell_exec("killall adb");

    $this->output.="Android Emulator was stopped successfully!";
}
```

Listing F.2: Implementation of App's Install, Launch, UI Test, Uninstall phases in test_cycle.php

```
//INSTALL App
//uninstall app initially if installed
//If AVD is not fully booted, then quit and abort test cycle
$uninstall_result = shell_exec($android_sdk_home . "/" .
$android_adb . " uninstall " . $package_name);
while (strpos($uninstall_result, "Error: Could not access the
Package Manager.  Is the system running?") !== FALSE) {
    sleep(20);
    $uninstall_result = shell_exec($android_sdk_home . "/" .
    $android_adb . " uninstall " . $package_name);
}

$test_app_process_echo.= "
    ----------------------------------------------------------------------<
    br>Initial Uninstall!<br>" . $uninstall_result;

//Clear log buffer
shell_exec($android_sdk_home . "/" . $android_adb . " logcat -c ")
;
//install
$test_app_process_echo.= "
----------------------------------------------------------------------<
br>Install!<br>" . shell_exec($android_sdk_home . "/" .
$android_adb . " install " . $apk_file);
//
----------------------------------------------------------------------<
br>Install!<br>" . shell_exec($android_sdk_home . "/" .
$android_adb . " wait-for-device install " . $apk_file);
//Due to bug in Android SDK r21, we should filter the log output
if (isset($config_array['log_install'])) {
    $parsed_data = shell_exec($android_sdk_home . "/" .
    $android_adb . " logcat -d  ");
    $parsed_data = preg_replace('/((.*)Unexpected value from
    nativeGetEnabledTags: 0\r\n)/', '', $parsed_data);
    $log_install = $parsed_data;
}
//CREATE folder for EACH TEST
$folder_name = $test_data_folder . "Test_" . date('Y_m_d_H_i_s');
echo shell_exec("mkdir " . $folder_name);

//
-----------------------------------------------------------------------------------------

//RUN/STOP App
//Clear log buffer
shell_exec($android_sdk_home . "/" . $android_adb . " logcat -c ")
;
//launch
$test_app_process_echo.= "
                                ------------------------------------------------------
                                br>Start App!<br>";
$string_tmp = shell_exec($android_sdk_home . "/" . $android_adb .
" shell am start -n " . $package_name . "/" .
$launchable_activity_name);

//Cut noisy output of am start
$test_app_process_echo.= substr($string_tmp, 0, strpos($string_tmp
, "usage: am [start"));

//Start tracing if necessary
if (isset($config_array['trace'])) {
    //echo shell_exec($android_sdk_home . "/" . $android_adb . "
    shell rm /data/misc/trace.txt");

    shell_exec("screen -d -m " . $android_sdk_home . "/" .
    $android_adb . " shell strace -e trace=all -p$(" .
    $android_sdk_home . "/" . $android_adb . " shell ps | grep " .
     $package_name . " | awk '{ print $2 }') -tt -f -o /data/misc/
    trace.txt 2>&1");
}

print "<pre>";

//Read cpu/memory consumption
if (isset($config_array['read_cpu_usage'])) {
    $read_cpu_usage = array();
    //Need to detect which column is CPU,THR and VSS
    $tmp_testeg = shell_exec($android_sdk_home . "/" .
    $android_adb . " shell top -n 1 | grep CPU");
    print_r($tmp_testeg);
    for ($i = 0; $i < 20; $i++) {
        if (strpos($tmp_testeg, "PCY") !== FALSE && strpos(
        $tmp_testeg, "PR") !== FALSE) //Android >4.0
            $tmp = shell_exec($android_sdk_home . "/" .
            $android_adb . " shell top -n 1  | grep " .
            $package_name . " | awk '{print$3,$5,$6,$7}'");
        elseif (strpos($tmp_testeg, "PCY") !== FALSE && strpos(
        $tmp_testeg, "PR") === FALSE) //Android 2.3.3
            $tmp = shell_exec($android_sdk_home . "/" .
            $android_adb . " shell top -n 1  | grep " .
            $package_name . " | awk '{print$2,$4,$5,$6}'");
        elseif (strpos($tmp_testeg, "PCY") === FALSE && strpos(
        $tmp_testeg, "PR") === FALSE) //Android 2.2
            $tmp = shell_exec($android_sdk_home . "/" .
            $android_adb . " shell top -n 1  | grep " .
            $package_name . " | awk '{print$2,$4,$5,$6}'");

        print_r($tmp);
        if ($i == 0)
            $start_execution_time = getTime();
        if (strlen($tmp) > 1) {
            $tmp = explode(" ", $tmp);
            $read_cpu_usage[number_format((getTime() -
            $start_execution_time), 2)]['CPU'] = $tmp[0];
            $read_cpu_usage[number_format((getTime() -
            $start_execution_time), 2)]['THR'] = $tmp[1];
            $read_cpu_usage[number_format((getTime() -
            $start_execution_time), 2)]['VSS'] = $tmp[2];
            $read_cpu_usage[number_format((getTime() -
            $start_execution_time), 2)]['RSS'] = $tmp[3];
        }
    }
}else
    sleep(10);

print "</pre>";

//adb shell top -n 1 | grep com.android.keychain
//Adoird 4.0
//PID  PR  CPU% S  #THR     VSS     RSS      PCY UID      Name
//1201  0   0%  S    10     174272K  17912K  bg system   com.
android.keychain
//Android 2.3.3
//   PID CPU% S  #THR     VSS     RSS PCY UID      Name
//Android 2.2
//  PID CPU% S  #THR     VSS     RSS UID      Name
// 563   7% S    35 191320K  32160K system   system_server
//adb shell ps -p -c
```

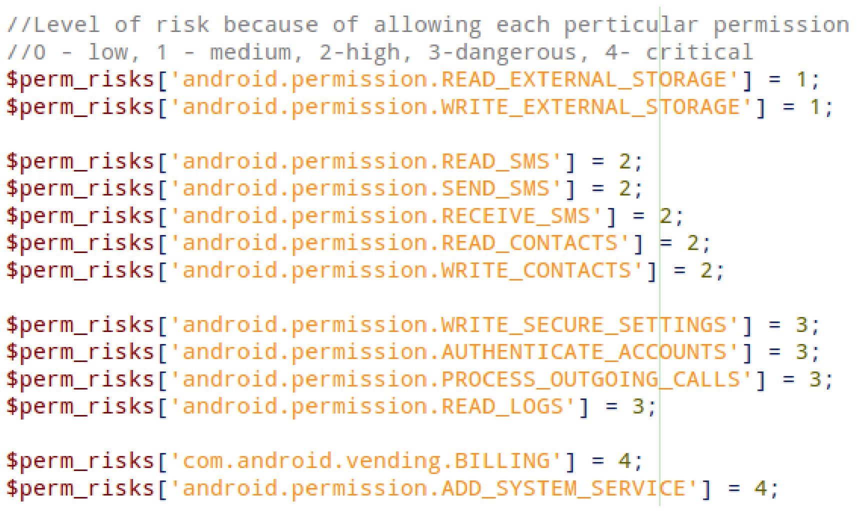
```
//-c show CPU (may not be available prior to Android 4.x) involved
//USER     PID   PPID  VSIZE  RSS   CPU PRIO  NICE  RTPRI SCHED WCHAN    PC         NAME
//echo $(adb shell ps | grep com.android.phone | awk '{ system("adb shell cat /proc/" $2 "/stat");}' | awk '{print $14+$15;}')
//SCREENING if necessary
if (isset($config_array['screenshot'])) {
    $test_app_process_echo .= "
                                        -----------------------------------------
                                        br>Screening!<br>";
    $test_app_process_echo .= shell_exec($android_sdk_home . "/" . $android_adb . " shell screencap /data/local/" . $selected_app_details['md5_name'] . ".png ");
    $test_app_process_echo .= shell_exec($android_sdk_home . "/" . $android_adb . " pull /data/local/" . $selected_app_details['md5_name'] . ".png " . $folder_name . "/screenshot.png ");
    if (file_exists($folder_name . "/screenshot.png"))
        $screenshot = true;
}

//Stop App
$test_app_process_echo .= "
                                -------------------------------------------------------
                                br>Stop!<br>" . shell_exec($android_sdk_home . "/" . $android_adb . " force-stop " . $package_name);
//Due to bug in Android SDK r21, we should filter the log output
if (isset($config_array['log_launch'])) {
    $parsed_data = shell_exec($android_sdk_home . "/" . $android_adb . " logcat -d  ");
    $parsed_data = preg_replace('/((.*)Unexpected value from nativeGetEnabledTags: 0\r\n)/', '', $parsed_data);
    $log_launch = $parsed_data;
}
//Ensure that app was stopped completely
shell_exec($android_sdk_home . "/" . $android_adb . " force-stop " . $package_name);
shell_exec($android_sdk_home . "/" . $android_adb . " shell kill $(" . $android_sdk_home . "/" . $android_adb . " shell ps | grep " . $package_name . " | awk '{ print $2 }')");
$android_sdk_home . "/" . $android_adb . " shell kill $(" . $android_sdk_home . "/" . $android_adb . " shell ps | grep " . $package_name . " | awk '{ print $2 }')";
//Stop Browser if was opened
shell_exec($android_sdk_home . "/" . $android_adb . " shell kill `" . $android_sdk_home . "/" . $android_adb . " shell ps | grep browser | awk '{ print $2 }'`");

```

```

//
-------------------------------------------------------------------------------
//UI TESTS
//TODO: ADD resources usa capturing
if (isset($config_array['ui_tester'])) {
    sleep(5);
    //Clear log buffer
    shell_exec($android_sdk_home . "/" . $android_adb . " logcat -
    c ");
    //Due to bug in Android SDK r21, we should filter the log
    output
    $test_app_process_echo.= "
                                        ---------------------------------------
                                        br>UI Tests<br>" .
                                        shell_exec(
                                        $android_sdk_home . "/" .
                                        $android_adb . "  shell
                                        monkey -p " .
                                        $package_name . " -v 500
                                        --throttle 50 --pct-touch
                                        70");
    //Create  log
    if (isset($config_array['log_test'])) {
        $parsed_data = shell_exec($android_sdk_home . "/" .
        $android_adb . " logcat -d  ");
        $parsed_data = preg_replace('/((.*)Unexpected value from
        nativeGetEnabledTags: 0\r\n)/', '', $parsed_data);
        $log_test = $parsed_data;
    }
}

//Finish Tracing if necessary
if (isset($config_array['trace'])) {
    shell_exec($android_sdk_home . "/" . $android_adb . " pull /
    data/misc/trace.txt " . $folder_name . "/");
}


//
-------------------------------------------------------------------------------
//PULL App data from Android Emulator
if ($config_array['pull_data']) {
    $test_app_process_echo.= "
                                        ---------------------------------------
                                        br>Pull Data!<br>" .
                                        shell_exec(
                                                    $android_sdk_home . "/" .
                                                    $android_adb . " pull /
                                                    data/data/" .
                                                    $package_name . " " .
                                                    $folder_name . "/data/ ");
    //check wether it exists
    if (is_dir($folder_name . "/data/"))
        $pull_data = true;
}

//
-------------------------------------------------------------------------------------

//BUGREPORT
if (isset($config_array['bugreport'])) {
    shell_exec($android_sdk_home . "/" . $android_adb . "
    bugreport > " . $folder_name . "/bugreport");
}

//
-------------------------------------------------------------------------------------

//UNINSTALL App
//Ensure that app was stopped completely
shell_exec($android_sdk_home . "/" . $android_adb . " force-stop "
 . $package_name);
shell_exec($android_sdk_home . "/" . $android_adb . " shell kill $
(" . $android_sdk_home . "/" . $android_adb . " shell ps | grep "
. $package_name . " | awk '{ print $2 }')");
$android_sdk_home . "/" . $android_adb . " shell kill $(" .
$android_sdk_home . "/" . $android_adb . " shell ps | grep " .
$package_name . " | awk '{ print $2 }')";
//Stop Browser if was opened
shell_exec($android_sdk_home . "/" . $android_adb . " shell kill `
" . $android_sdk_home . "/" . $android_adb . " shell ps | grep
browser | awk '{ print $2 }'`");

//Clear log buffer
shell_exec($android_sdk_home . "/" . $android_adb . " logcat -c ")
;
```

Listing F.3: Entropy analysis functionality in analysis.php

```
//Working with package
$package_entropy = 0;
$package_number_files = 0;
$manifest_size = 0;
$res_folder_size = 0;
$assets_folder_size = 0;
$classes_dex_size = 0;
```

```
$classes_dex_entropy = 0;
$execution_time = 0;

$apk_file = $permanent_folder . $row['md5_name'] . ".apk";
echo shell_exec("mkdir tmp/");
shell_exec("unzip " . $apk_file . " -d tmp/");
//entropy package
shell_exec("tar -cf tmp.tar tmp/");
$package_entropy = $this->entropy(file_get_contents("tmp.tar"));
shell_exec("rm tmp.tar");
//total files
$package_number_files = shell_exec("find tmp/ |wc -l");
//manifest size
$manifest_size = filesize("tmp/AndroidManifest.xml");
//res size
if (is_dir("tmp/res")) {
    $size = explode("\t", exec("du -hk " . "tmp/res"), 2);
    $size[0] == "tmp/res" ? $size[0] : 0;
    $res_folder_size = $size[0];
}
//assets size
if (is_dir("tmp/assets")) {
    $size = explode("\t", exec("du -hk " . "tmp/assets"), 2);
    $size[0] == "tmp/assets" ? $size[0] : 0;
    $assets_folder_size = $size[0] . "<br>";
}
//classes_dex size
$classes_dex_size = filesize("tmp/classes.dex");
//entropy dex
$classes_dex_entropy = $this->entropy(file_get_contents("tmp/
classes.dex"));
echo shell_exec("rm -rf tmp/");
```



```
//Level of risk because of allowing each perticular permission
//0 - low, 1 - medium, 2-high, 3-dangerous, 4- critical
$perm_risks['android.permission.READ_EXTERNAL_STORAGE'] = 1;
$perm_risks['android.permission.WRITE_EXTERNAL_STORAGE'] = 1;

$perm_risks['android.permission.READ_SMS'] = 2;
$perm_risks['android.permission.SEND_SMS'] = 2;
$perm_risks['android.permission.RECEIVE_SMS'] = 2;
$perm_risks['android.permission.READ_CONTACTS'] = 2;
$perm_risks['android.permission.WRITE_CONTACTS'] = 2;

$perm_risks['android.permission.WRITE_SECURE_SETTINGS'] = 3;
$perm_risks['android.permission.AUTHENTICATE_ACCOUNTS'] = 3;
$perm_risks['android.permission.PROCESS_OUTGOING_CALLS'] = 3;
$perm_risks['android.permission.READ_LOGS'] = 3;

$perm_risks['com.android.vending.BILLING'] = 4;
$perm_risks['android.permission.ADD_SYSTEM_SERVICE'] = 4;
```

Figure 57: Translation of permissions into risk levels

Listing F.4: Implementation of Mean and Standard Deviation calculation in C++

```cpp
//Calculate mean and standard deviation for both classes separately
//Mean
int numEntriesTmp1, numEntriesTmp2;
for (int j = 0; j < dim; j++) {
    tmp1 = 0;
    tmp2 = 0;
    numEntriesTmp1 = 0;
    numEntriesTmp2 = 0;
    for (int i = 0; i < N; i++) {
        if (classID[i] == 0) {
            tmp1 += input[i][j];
            numEntriesTmp1++;
        } else if (classID[i] == 1) {
            tmp2 += input[i][j];
            numEntriesTmp2++;
        }
    }
    tmp1 = tmp1 / numEntriesTmp1;
    means1.push_back(tmp1);
    tmp2 = tmp2 / numEntriesTmp2;
    means2.push_back(tmp2);
}

//StDev
for (int j = 0; j < dim; j++) {
    tmp1 = 0;
    tmp2 = 0;
    numEntriesTmp1 = 0;
    numEntriesTmp2 = 0;
    for (int i = 0; i < N; i++) {
        if (classID[i] == 0) {
            tmp1 += (input[i][j] - means1[j])*(input[i][j] -
                means1[j]);
            numEntriesTmp1++;

        } else if (classID[i] == 1) {
            tmp2 += (input[i][j] - means2[j])*(input[i][j] -
                means2[j]);
            numEntriesTmp2++;

        }
    }
    tmp1 = sqrt(tmp1 / numEntriesTmp1);
    stDev1.push_back(tmp1);
    tmp2 = sqrt(tmp2 / numEntriesTmp2);
    stDev2.push_back(tmp2);
}
```

Listing F.5: Implementation of ANN learning in C++ using OpenMP

```
//Weights Initialization
for (long long int j = 0; j < numberRules; j++)
    weights.push_back(1 / (numberRules));

//Neural Network learning
for (int i = 0; i < 1000; i++) {
    for (int m = 0; m < N; m++) {
        //Assigning degree of membership
        float output = 0;
#pragma omp parallel for reduction(+:output)
        for (long long int j = 0; j < numberRules; j++) {
            float tmp1 = 1;
            tmp1 = fuzzificationFunction(input[m][0], means[0],
            stDev[0], (int) j / (int) pow(nFuzzySets, dim - 1));
            for (int l = 1; l < dim; l++)
                tmp1 *= fuzzificationFunction(input[m][l], means[l
                ], stDev[l], (int) j / (int) pow(nFuzzySets, dim -
                 l - 1) % (int) pow(nFuzzySets, 1));
            rules[j] = tmp1;
            output += weights[j] * rules[j];
        }

        //Adjusting weighrs
#pragma omp parallel for
        for (long long int k = 0; k < numberRules; k++)
            weights[k] += 0.1 * (classID[m] - sigmoidFunction(
            output)) * rules[k];

    }
}
```

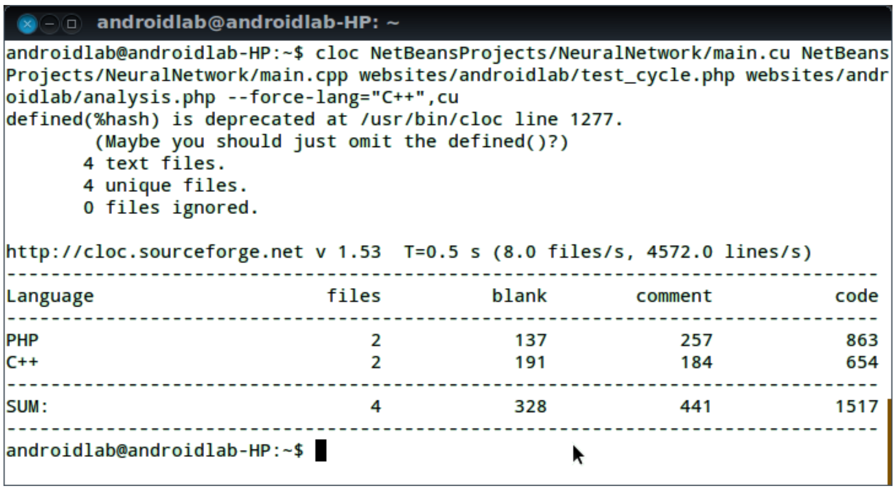
Listing F.6: Rules selection implementation in C++ using STL

```
//Sorting constructed rules according to fuzzy-neuro weight value
vector<short int> rulesAtomId;
map<float, vector<short int> > rulesMap;
for (long long int k = 0; k < numberRules; k++) {
    rulesAtomId.clear();
    rulesAtomId.push_back((int) k / (int) pow(nFuzzySets, dim - 1)
    );

    for (int l = 1; l < dim; l++) {
        rulesAtomId.push_back((int) k / (int) pow(nFuzzySets, dim
        - l - 1) % (int) pow(nFuzzySets, 1));
    }
    rulesMap.insert(std::pair<float, vector<short int> >(weights[k
    ] + 1e-5 * k, rulesAtomId));
}

//Assigning Class ID to extracted rules
std::map<float, vector<short int> >::reverse_iterator it;
vector< vector<short int> >extractedRules;
vector<short int>extractedRulesClasses;
vector<short int> ruleTmp;
for (it = rulesMap.rbegin(); it != rulesMap.rend(); it++) {
    ruleTmp.clear();
    float degC1 = 1, degC2 = 1;
    //Membership degree calculation for each class
    for (int l = 0; l < dim; l++) {
        degC1 *= fuzzificationFunction(means1[l], stDev[l], means[
        l], it->second[l]);
        degC2 *= fuzzificationFunction(means2[l], stDev[l], means[
        l], it->second[l]);
        ruleTmp.push_back(it->second[l]);
    }
    //Defining class
    if (degC1 > degC2) {
        extractedRulesClasses.push_back(0);
    } else {

        extractedRulesClasses.push_back(1);
    }
    extractedRules.push_back(ruleTmp);
}
```

Listing F.7: Sample of cubic spline approximation implemented in C++

```
void splineInterpolationProcessSplines(std::vector<float> &x, std
::vector<float> &y) {
    int n = x.size();

    //Check if sizes of vector are the same and content no less
    than 2 element
    if (n > 1 && n == y.size()) {
        std::vector<float> alpha;
        std::vector<float> beta;
        spline_tuple tmp;

        alpha.reserve(n);
        beta.reserve(n);
        splines.reserve(n);

        for (std::size_t i = 0; i < n; ++i) {
            tmp.x = x[i];
            tmp.a = y[i];
            splines.push_back(tmp);
        }

        //Calculate C coefficients
        //Ci-1 = 0
        splines[0].c = splines[n - 1].c = 0.;
        alpha.push_back(0);
        beta.push_back(0);

        for (std::size_t i = 1; i < n - 1; ++i) {
            float h_i = x[i] - x[i - 1], h_i1 = x[i + 1] - x[i];
            float A = h_i;
            float C = 2. * (h_i + h_i1);
            float B = h_i1;
            float F = 6. * ((y[i + 1] - y[i]) / h_i1 - (y[i] - y[i
 - 1]) / h_i);
            float z = (A * alpha[i - 1] + C);
            alpha.push_back(-B / z);
            beta.push_back((F - A * beta[i - 1]) / z);
        }

        for (std::size_t i = n - 2; i > 0; --i)
            splines[i].c = alpha[i] * splines[i + 1].c + beta[i];

        //Calculate B and D coefficients
        for (std::size_t i = n - 1; i > 0; --i) {
            float h_i = x[i] - x[i - 1];
            // Di = (Ci - Ci-1)/hi
            splines[i].d = (splines[i].c - splines[i - 1].c) / h_i
            ;

            //Bi = 1/2 * Hi * Ci - 1/6 * Hi^2 * Di + (Fi - Fi-1) /
              Hi
            splines[i].b = h_i * (2. * splines[i].c + splines[i -
            1].c) / 6. + (y[i] - y[i - 1]) / h_i;
        }

        //Clear temp vectors memory
        std::vector<float>().swap(alpha);
        std::vector<float>().swap(beta);

    } else {
        std::string str("Vectors have different sizes");
        //throw std::runtime_error(str.c_str());
    }
}

double splineInterpolationGetResult(float xArg) {
    if (!splines.size()) {
        std::string str("Spline is not defined");
        //throw std::runtime_error(str.c_str());
    }

    int n = splines.size();
    spline_tuple s;

    if (xArg <= splines[0].x)
        s = splines[1];
    else if (xArg >= splines[n - 1].x)
        s = splines[n - 1];
    else {
        //Search nearest spline
        std::size_t i = 0, j = n - 1;
        while (i + 1 < j) {
            std::size_t k = i + (j - i) / 2;
            if (xArg <= splines[k].x)
                j = k;
            else
                i = k;
        }
        s = splines[j];
    }

    //dx = (X -Xi)
    float dx = (xArg - s.x);

    //Calculate value of polynom, based on spline coefficients:
    //Si = Ai + Bi  * (X-Xi) + Ci/2  * (X-Xi)^2  + Di/6  * (X-Xi)
    ^3.
    return s.a + (s.b + (s.c / 2. + s.d * dx / 6.) * dx) * dx;
}
```

In order to estimate amount of developed code, we used CLOC[1] utility as it is depicted in the Figure 58.



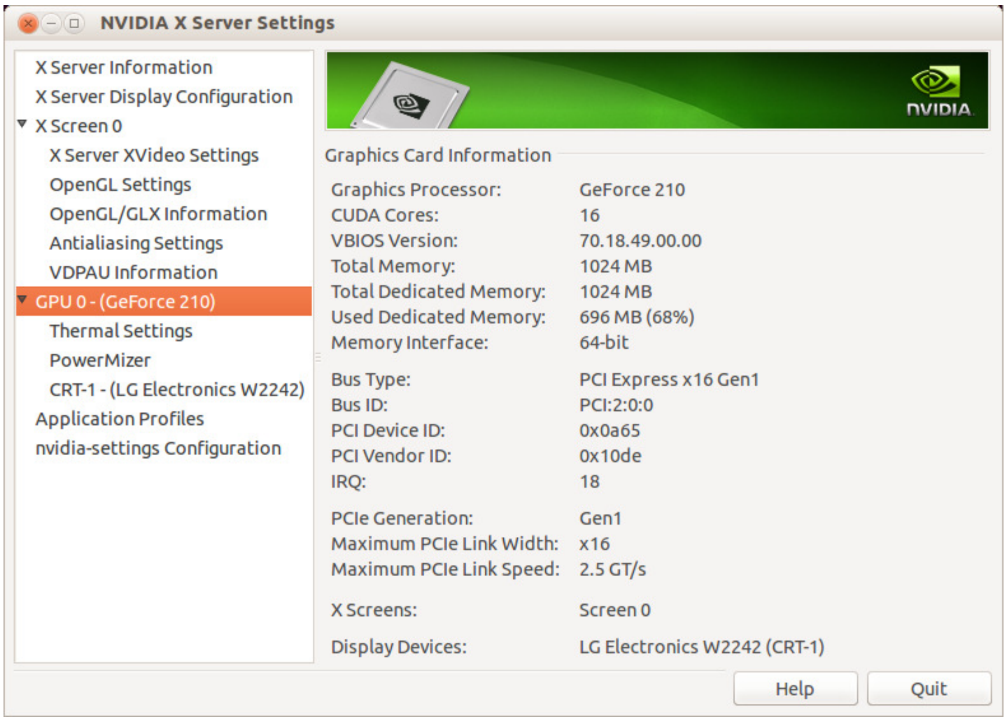
```
androidlab@androidlab-HP: ~
androidlab@androidlab-HP:~$ cloc NetBeansProjects/NeuralNetwork/main.cu NetBeans
Projects/NeuralNetwork/main.cpp websites/androidlab/test_cycle.php websites/andr
oidlab/analysis.php --force-lang="C++",cu
defined(%hash) is deprecated at /usr/bin/cloc line 1277.
        (Maybe you should just omit the defined()?)
       4 text files.
       4 unique files.
       0 files ignored.

http://cloc.sourceforge.net v 1.53  T=0.5 s (8.0 files/s, 4572.0 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
PHP                              2            137            257            863
C++                              2            191            184            654
-------------------------------------------------------------------------------
SUM:                             4            328            441           1517
-------------------------------------------------------------------------------
androidlab@androidlab-HP:~$
```

Figure 58: Amount of developed code

[1]http://cloc.sourceforge.net/

G Miscellaneous information

Nvidia-settings[1] Linux package provides graphic card characteristics.



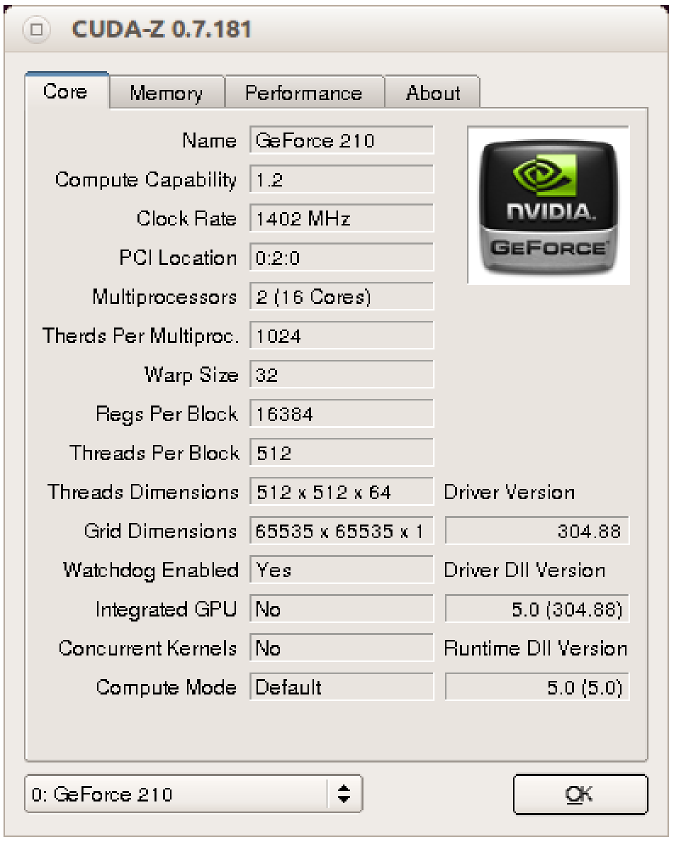
Graphics Card Information

Graphics Processor:	GeForce 210
CUDA Cores:	16
VBIOS Version:	70.18.49.00.00
Total Memory:	1024 MB
Total Dedicated Memory:	1024 MB
Used Dedicated Memory:	696 MB (68%)
Memory Interface:	64-bit
Bus Type:	PCI Express x16 Gen1
Bus ID:	PCI:2:0:0
PCI Device ID:	0x0a65
PCI Vendor ID:	0x10de
IRQ:	18
PCIe Generation:	Gen1
Maximum PCIe Link Width:	x16
Maximum PCIe Link Speed:	2.5 GT/s
X Screens:	Screen 0
Display Devices:	LG Electronics W2242 (CRT-1)

Figure 59: GeForce N210 parameters from Nvidia-settings

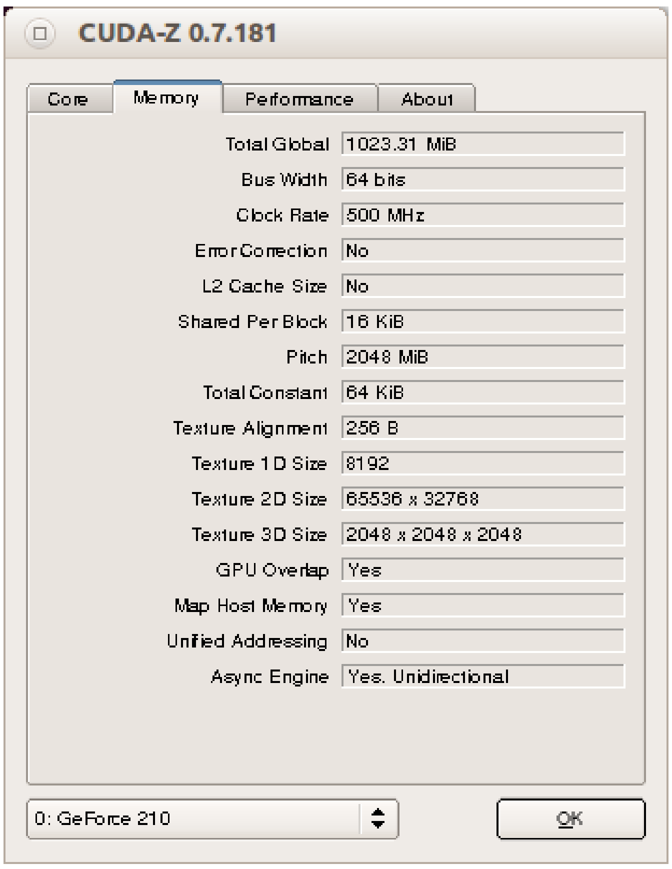
CUDA-Z[2] is a powerful info too for obtaining precise information about CUDA features in graphic card.

[1] http://packages.debian.org/en/squeeze/nvidia-settings
[2]http://cuda-z.sourceforge.net/



Name	GeForce 210	
Compute Capability	1.2	
Clock Rate	1402 MHz	
PCI Location	0:2:0	
Multiprocessors	2 (16 Cores)	
Therds Per Multiproc.	1024	
Warp Size	32	
Regs Per Block	16384	
Threads Per Block	512	
Threads Dimensions	512 x 512 x 64	Driver Version
Grid Dimensions	65535 x 65535 x 1	304.88
Watchdog Enabled	Yes	Driver Dll Version
Integrated GPU	No	5.0 (304.88)
Concurrent Kernels	No	Runtime Dll Version
Compute Mode	Default	5.0 (5.0)

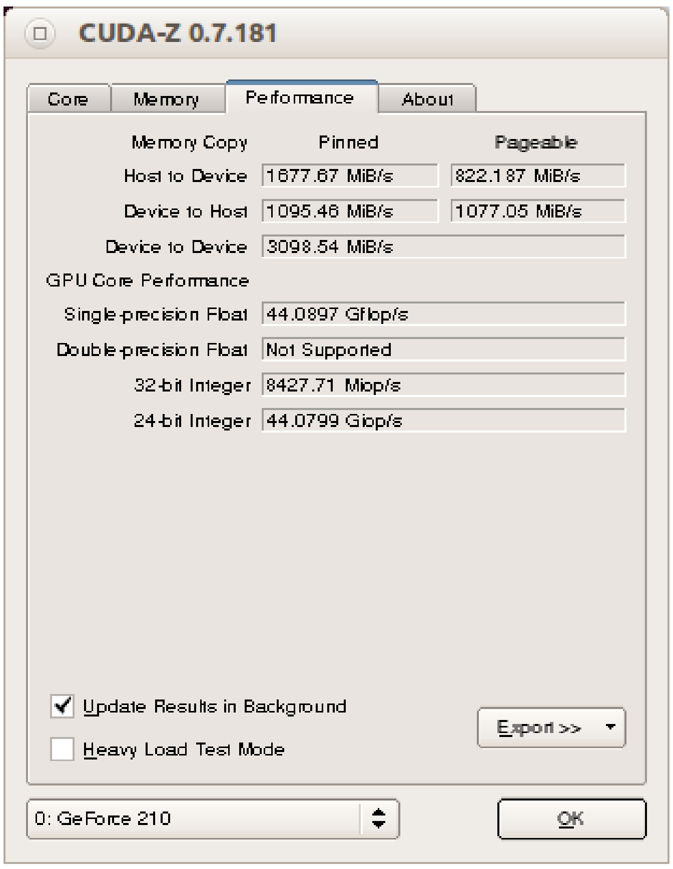
Figure 60: GeForce N210 memory characteristics in CUDA-z



Total Global	1023.31 MiB
Bus Width	64 bits
Clock Rate	500 MHz
Error Correction	No
L2 Cache Size	No
Shared Per Block	16 KiB
Pitch	2048 MiB
Total Constant	64 KiB
Texture Alignment	256 B
Texture 1D Size	8192
Texture 2D Size	65536 x 32768
Texture 3D Size	2048 x 2048 x 2048
GPU Overlap	Yes
Map Host Memory	Yes
Unified Addressing	No
Async Engine	Yes, Unidirectional

0: GeForce 210

Figure 61: Information about GeForce N210 in CUDA-z



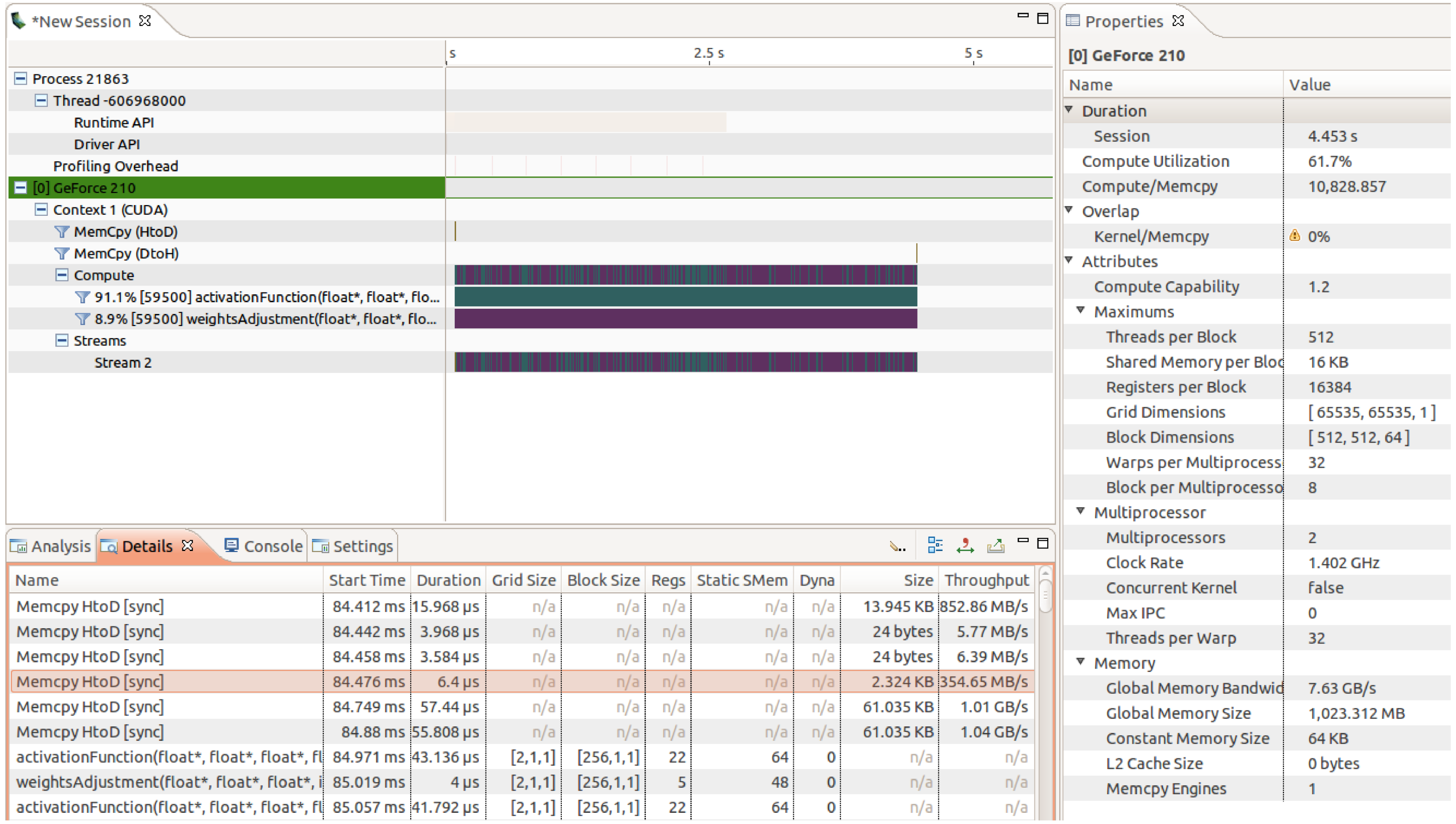
Memory Copy	Pinned	Pageable
Host to Device	1677.67 MiB/s	822.187 MiB/s
Device to Host	1095.46 MiB/s	1077.05 MiB/s
Device to Device	3098.54 MiB/s	

GPU Core Performance	
Single-precision Float	44.0897 Gflop/s
Double-precision Float	Not Supported
32-bit Integer	8427.71 Miop/s
24-bit Integer	44.0799 Giop/s

[✓] Update Results in Background
[ ] Heavy Load Test Mode

0: GeForce 210

Figure 62: GeForce N210 performace measures in CUDA-z



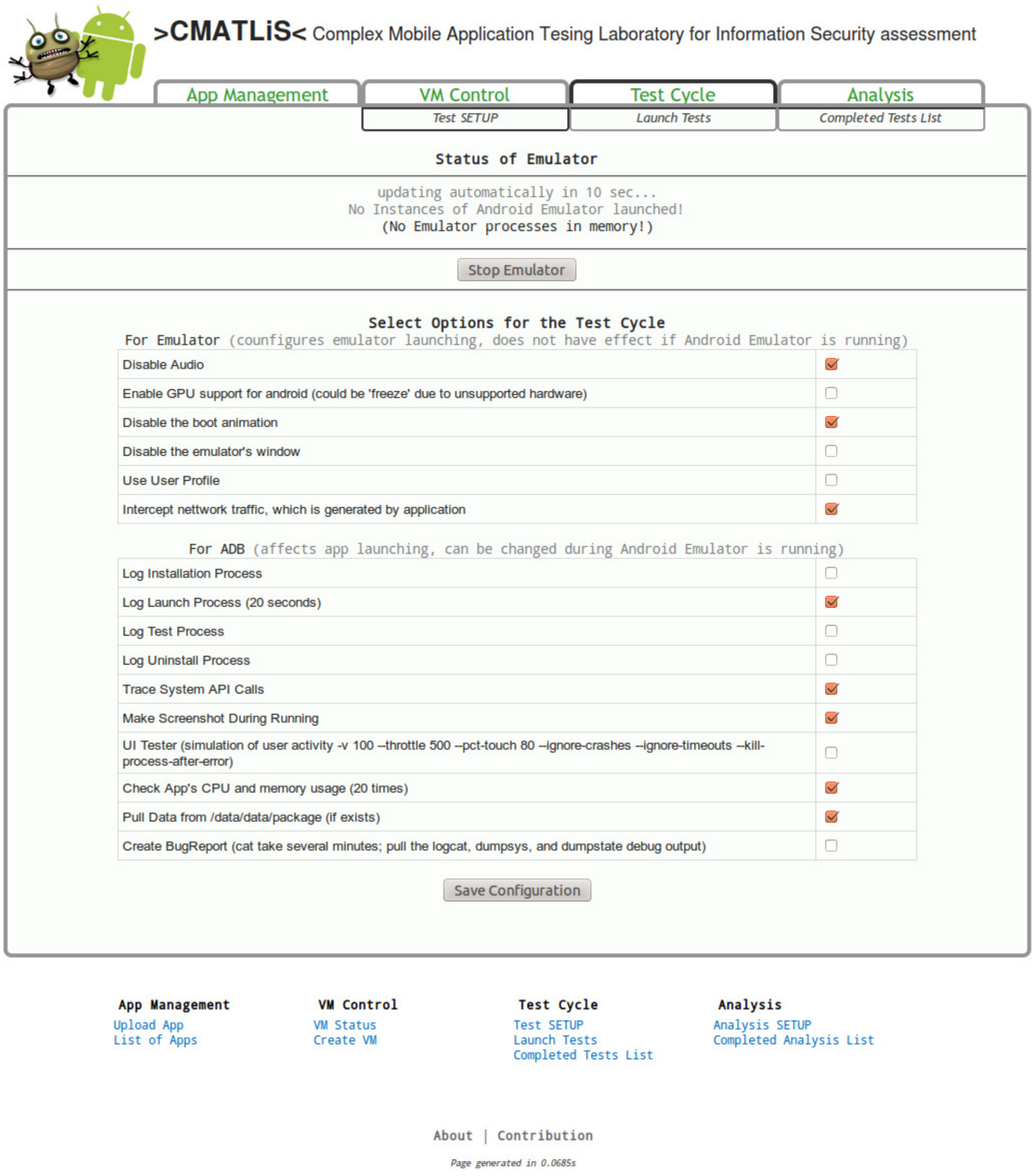
Process 21863
Thread -606968000
Runtime API
Driver API
Profiling Overhead
[0] GeForce 210
Context 1 (CUDA)
MemCpy (HtoD)
MemCpy (DtoH)
Compute
91.1% [59500] activationFunction(float*, float*, flo...
8.9% [59500] weightsAdjustment(float*, float*, flo...
Streams
Stream 2

Name	Start Time	Duration	Grid Size	Block Size	Regs	Static SMem	Dyna	Size	Throughput
Memcpy HtoD [sync]	84.412 ms	15.968 μs	n/a	n/a	n/a	n/a	n/a	13.945 KB	852.86 MB/s
Memcpy HtoD [sync]	84.442 ms	3.968 μs	n/a	n/a	n/a	n/a	n/a	24 bytes	5.77 MB/s
Memcpy HtoD [sync]	84.458 ms	3.584 μs	n/a	n/a	n/a	n/a	n/a	24 bytes	6.39 MB/s
Memcpy HtoD [sync]	84.476 ms	6.4 μs	n/a	n/a	n/a	n/a	n/a	2.324 KB	354.65 MB/s
Memcpy HtoD [sync]	84.749 ms	57.44 μs	n/a	n/a	n/a	n/a	n/a	61.035 KB	1.01 GB/s
Memcpy HtoD [sync]	84.88 ms	55.808 μs	n/a	n/a	n/a	n/a	n/a	61.035 KB	1.04 GB/s
activationFunction(float*, float*, float*, fl	84.971 ms	43.136 μs	[2,1,1]	[256,1,1]	22	64	0	n/a	n/a
weightsAdjustment(float*, float*, float*, i	85.019 ms	4 μs	[2,1,1]	[256,1,1]	5	48	0	n/a	n/a
activationFunction(float*, float*, float*, fl	85.057 ms	41.792 μs	[2,1,1]	[256,1,1]	22	64	0	n/a	n/a

[0] GeForce 210

Name	Value
Duration	
Session	4.453 s
Compute Utilization	61.7%
Compute/Memcpy	10,828.857
Overlap	
Kernel/Memcpy	0%
Attributes	
Compute Capability	1.2
Maximums	
Threads per Block	512
Shared Memory per Bloc	16 KB
Registers per Block	16384
Grid Dimensions	[65535, 65535, 1]
Block Dimensions	[512, 512, 64]
Warps per Multiprocess	32
Block per Multiprocesso	8
Multiprocessor	
Multiprocessors	2
Clock Rate	1.402 GHz
Concurrent Kernel	false
Max IPC	0
Threads per Warp	32
Memory	
Global Memory Bandwid	7.63 GB/s
Global Memory Size	1,023.312 MB
Constant Memory Size	64 KB
L2 Cache Size	0 bytes
Memcpy Engines	1

Figure 63: Profiling of ANN learning with CUDA support by means of Nvidia Profiler [115] that shows execution time distribution among various operations



>CMATLiS< Complex Mobile Application Tesing Laboratory for Information Security assessment

Status of Emulator

```
updating automatically in 10 sec...
No Instances of Android Emulator launched!
(No Emulator processes in memory!)
```

Select Options for the Test Cycle

For Emulator (counfigures emulator launching, does not have effect if Android Emulator is running)

Disable Audio	✓
Enable GPU support for android (could be 'freeze' due to unsupported hardware)	☐
Disable the boot animation	✓
Disable the emulator's window	☐
Use User Profile	☐
Intercept nettwork traffic, which is generated by application	✓

For ADB (affects app launching, can be changed during Android Emulator is running)

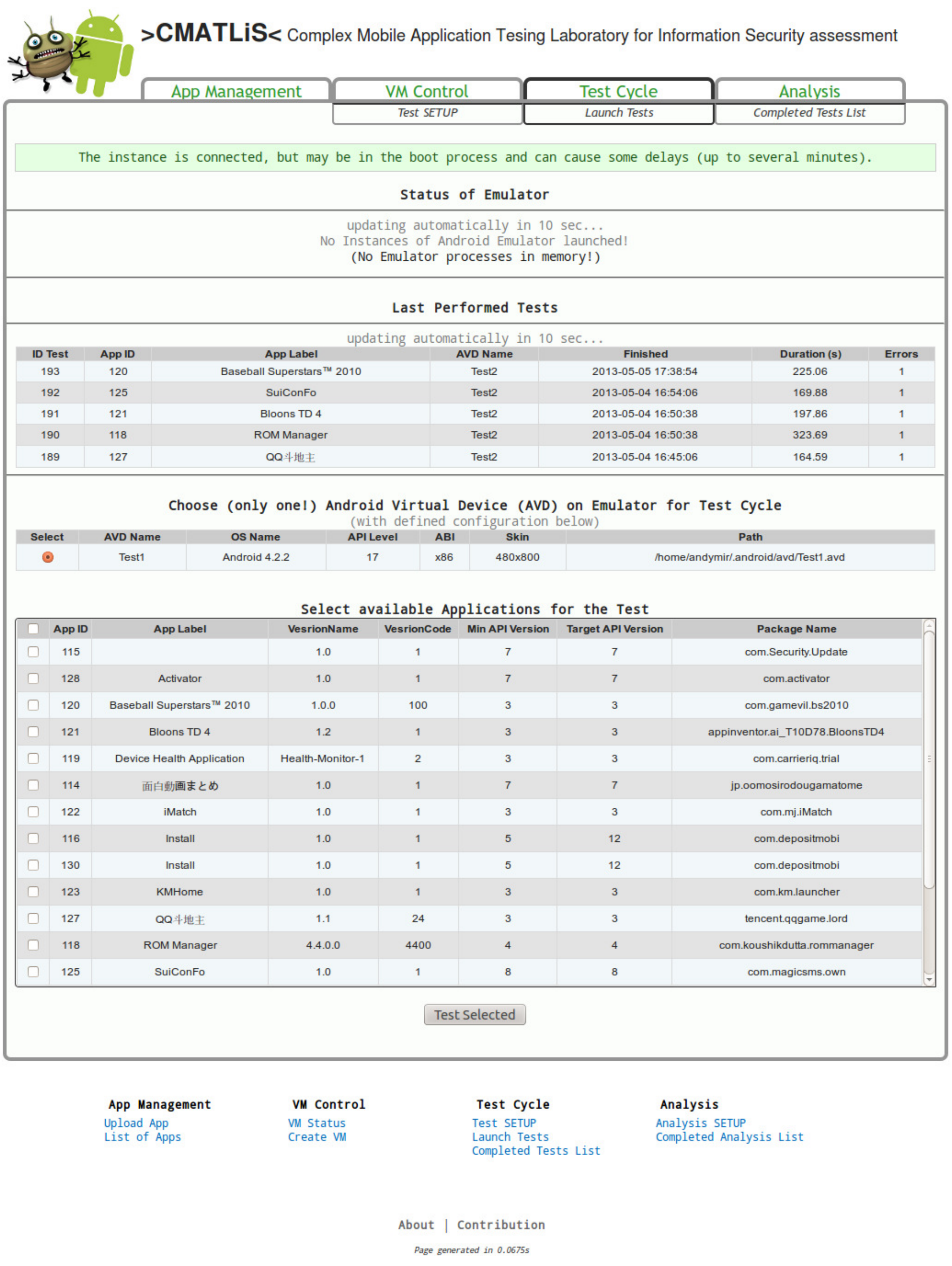
Log Installation Process	☐
Log Launch Process (20 seconds)	✓
Log Test Process	☐
Log Uninstall Process	☐
Trace System API Calls	✓
Make Screenshot During Running	✓
UI Tester (simulation of user activity -v 100 –throttle 500 –pct-touch 80 –ignore-crashes –ignore-timeouts –kill-process-after-error)	☐
Check App's CPU and memory usage (20 times)	✓
Pull Data from /data/data/package (if exists)	✓
Create BugReport (cat take several minutes; pull the logcat, dumpsys, and dumpstate debug output)	☐

App Management	VM Control	Test Cycle	Analysis
Upload App	VM Status	Test SETUP	Analysis SETUP
List of Apps	Create VM	Launch Tests	Completed Analysis List
		Completed Tests List	

About | Contribution

Page generated in 0.0685s

Figure 64: Sample of configuration-settings page, which was implemented in the testing laboratory


>CMATLiS< Complex Mobile Application Tesing Laboratory for Information Security assessment
App Management
VM Control
Test Cycle
Analysis
Test SETUP
Launch Tests
Completed Tests List

The instance is connected, but may be in the boot process and can cause some delays (up to several minutes).

Status of Emulator

updating automatically in 10 sec...
No Instances of Android Emulator launched!
(No Emulator processes in memory!)

Last Performed Tests

updating automatically in 10 sec...

ID Test	App ID	App Label	AVD Name	Finished	Duration (s)	Errors
193	120	Baseball Superstars™ 2010	Test2	2013-05-05 17:38:54	225.06	1
192	125	SuiConFo	Test2	2013-05-04 16:54:06	169.88	1
191	121	Bloons TD 4	Test2	2013-05-04 16:50:38	197.86	1
190	118	ROM Manager	Test2	2013-05-04 16:50:38	323.69	1
189	127	QQ斗地主	Test2	2013-05-04 16:45:06	164.59	1

Choose (only one!) Android Virtual Device (AVD) on Emulator for Test Cycle
(with defined configuration below)

Select	AVD Name	OS Name	API Level	ABI	Skin	Path
●	Test1	Android 4.2.2	17	x86	480x800	/home/andymir/.android/avd/Test1.avd

Select available Applications for the Test

☐	App ID	App Label	VesrionName	VesrionCode	Min API Version	Target API Version	Package Name
☐	115		1.0	1	7	7	com.Security.Update
☐	128	Activator	1.0	1	7	7	com.activator
☐	120	Baseball Superstars™ 2010	1.0.0	100	3	3	com.gamevil.bs2010
☐	121	Bloons TD 4	1.2	1	3	3	appinventor.ai_T10D78.BloonsTD4
☐	119	Device Health Application	Health-Monitor-1	2	3	3	com.carrieriq.trial
☐	114	面白動画まとめ	1.0	1	7	7	jp.oomosirodougamatome
☐	122	iMatch	1.0	1	3	3	com.mj.iMatch
☐	116	Install	1.0	1	5	12	com.depositmobi
☐	130	Install	1.0	1	5	12	com.depositmobi
☐	123	KMHome	1.0	1	3	3	com.km.launcher
☐	127	QQ斗地主	1.1	24	3	3	tencent.qqgame.lord
☐	118	ROM Manager	4.4.0.0	4400	4	4	com.koushikdutta.rommanager
☐	125	SuiConFo	1.0	1	8	8	com.magicsms.own

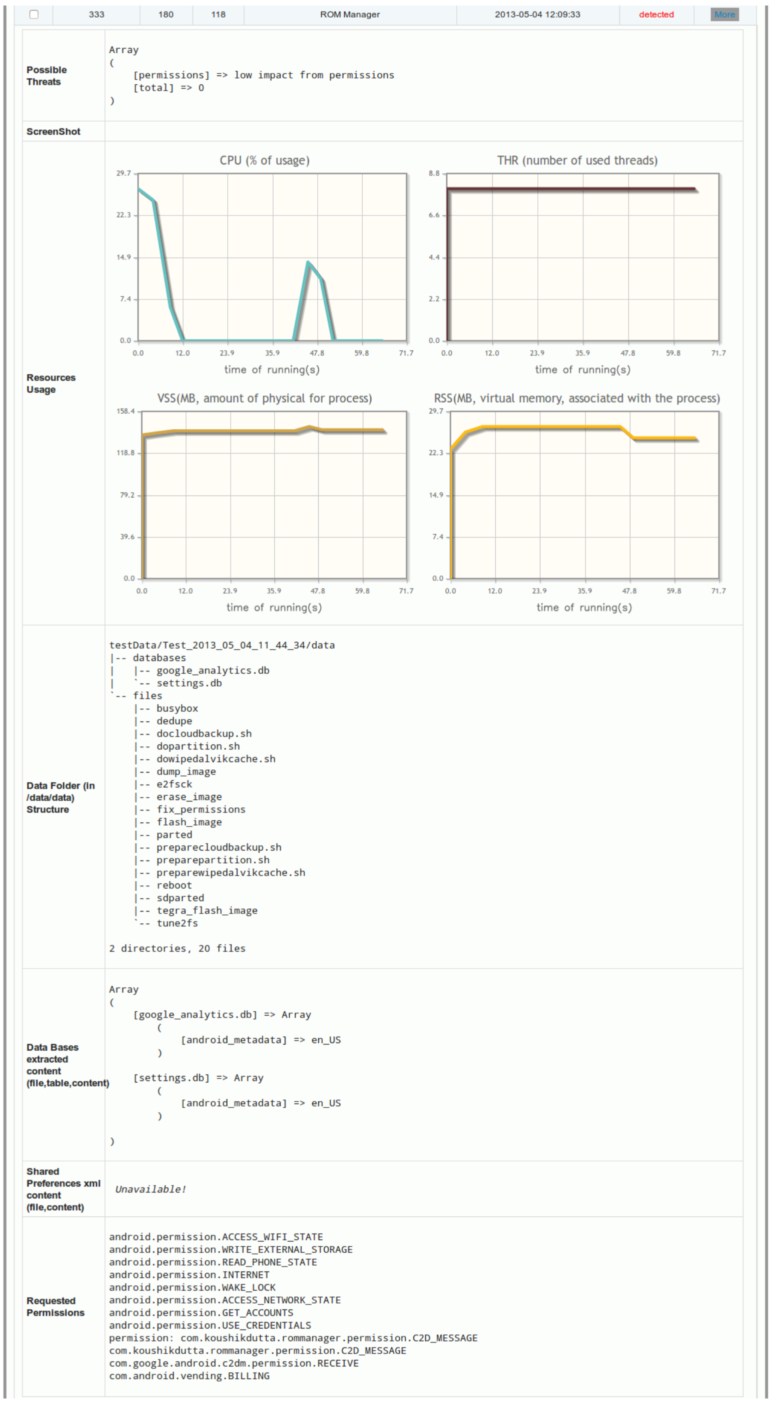
Test Selected

App Management	VM Control	Test Cycle	Analysis
Upload App	VM Status	Test SETUP	Analysis SETUP
List of Apps	Create VM	Launch Tests	Completed Analysis List
		Completed Tests List	

About | Contribution

Page generated in 0.0675s

Figure 65: Sample of application-testing page in the testing laboratory



☐	333	180	118	ROM Manager	2013-05-04 12:09:33	detected

Possible Threats

```
Array
(
    [permissions] => low impact from permissions
    [total] => 0
)
```

ScreenShot

Resources Usage

Data Folder (in /data/data) Structure

```
testData/Test_2013_05_04_11_44_34/data
|-- databases
|   |-- google_analytics.db
|   `-- settings.db
`-- files
    |-- busybox
    |-- dedupe
    |-- docloudbackup.sh
    |-- dopartition.sh
    |-- dowipedalvikcache.sh
    |-- dump_image
    |-- e2fsck
    |-- erase_image
    |-- fix_permissions
    |-- flash_image
    |-- parted
    |-- preparecloudbackup.sh
    |-- preparepartition.sh
    |-- preparewipedalvikcache.sh
    |-- reboot
    |-- sdparted
    |-- tegra_flash_image
    `-- tune2fs

2 directories, 20 files
```

Data Bases extracted content (file,table,content)

```
Array
(
    [google_analytics.db] => Array
        (
            [android_metadata] => en_US
        )

    [settings.db] => Array
        (
            [android_metadata] => en_US
        )

)
```

Shared Preferences xml content (file,content)

Unavailable!

Requested Permissions

```
android.permission.ACCESS_WIFI_STATE
android.permission.WRITE_EXTERNAL_STORAGE
android.permission.READ_PHONE_STATE
android.permission.INTERNET
android.permission.WAKE_LOCK
android.permission.ACCESS_NETWORK_STATE
android.permission.GET_ACCOUNTS
android.permission.USE_CREDENTIALS
permission: com.koushikdutta.rommanager.permission.C2D_MESSAGE
com.koushikdutta.rommanager.permission.C2D_MESSAGE
com.google.android.c2dm.permission.RECEIVE
com.android.vending.BILLING
```

Figure 66: Sample of application-analysis page in the testing laboratory