Detection of Smartphone Malware

Eingereicht von
Diplom-Informatiker
Aubrey-Derrick Schmidt

Von der Fakultät IV – Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
– Dr.-Ing. –

genehmigte Dissertation

Promotionsausschuß:

Vorsitzender:	Prof. Dr. Jean-Pierre Seifert
Berichter:	Prof. Dr.-Ing. Sahin Albayrak
Berichter:	Prof. Dr. Fernando C. Colon Osorio

Tag der wissenschaftlichen Aussprache: 28.06.2011

Berlin 2011

D 83

Acknowledgements

On completion of my Ph.D. thesis I would like to sincerely thank all those who supported me in realizing and finishing my work.

First of all, I am heartily thankful to my supervisors and Ph.D. Committee spending time and effort on me. *Prof. Dr.-Ing. Sahin Albayrak* and *Ph.D. Ahmet Camtepe* always were a shining example for scientific success to me. Throughout all of the stages of my thesis, they helped me to keep track on the right research direction, seriously revised all of my work, and patiently discussed and resolved issues not only related to my work. I am also deeply moved by their serious and honest attitude towards academic work. Additionally, I really appreciate their will for hosting and motivating me all the time while working at DAI-Laboratory at Technische Universität Berlin. I want to honestly thank them for their friendly, personal, and self-sacrificing will to help me in any situation throughout my time at the DAI-Laboratory. When meeting *Prof. Dr. Fernando C. Colon Osorio* on Malware Conference 2009 in Montreal the first time, I was really impressed by his will to put scientific discussion into the focus of the conference. This honest approach towards research interaction and progress allowed me to get to know several interesting and, more important, very kind researchers giving me valuable input for current and future research problems and directions. Moreover, his immediate commitment to join my thesis committee made me very proud, happy, and thankful since his impressive expertise and experience in malware and security research were a valuable source for my work.

Furthermore, I thank to the Competence Center Security of the DAI-Laboratory for backing and discussing my approaches presented in this work. While working in this group I learned a lot with and from our team members, especially from Jan Clausen, Leonid Batyuk, Karsten Bsufka, Rainer Bye, Joel Chinnow, Stephan Schmidt, Arik Messerman, Tarik Mus-

tafic, and Thomas Bläsing. I also appreciate all the evenings spent with our undergraduate student workers Florian Lamour, Jakob Strafer, Thomas Hausschild, Dennis Grunewald, Karsten Raddatz, Osman Kiraz, and Ali Yüksel.

My deepest thanks go to my beloved wife and children. Thank you for giving me so much support.

Abstract

Due to technological progress, mobile phones evolved into technically and functionally sophisticated devices called *smartphones*. Providing comprehensive capabilities, *smartphones* are getting increasingly popular not only for the targeted users but all. Since 2004, several malwares appeared targeting these devices. General countermeasures to smartphone malwares are currently limited to signature-based anti-virus scanners which efficiently detect *known* malwares, but they have serious shortcomings with *new* and *unknown* malwares creating a window of opportunity for attackers. As smartphones become a host for sensitive data and applications, extended malware detection mechanisms not basing on signatures are necessary complying with the resource constraints of current mobile devices.

In this work, we tackle the field of smartphone malware. We give a clear definition on what a *smartphone* actually is since an industry standard does not exist. For understanding the threat of malwares targeting smartphones, we present an updated list including all published malwares that were recognized by anti-virus companies until the end of 2010.

We introduce the fields of *dynamic* and *static analysis*. In the field of *dynamic analysis*, a monitoring system is introduced gathering behavior- and system-based information that are processed by a remote system using machine learning for anomaly detection. Furthermore, a monitoring and detection architecture for Linux-based smartphones is presented which is used to trace execution of binaries for extracting invoked system calls.

In the field of *static analysis*, we discuss its applicability to the domain of different smartphone platforms, namely Symbian OS and Android. In both cases, function and system calls are used that are extracted from binaries in a static manner. Results of the analyses are promising and showed competitive character in comparison with standard state-of-the-art learning algorithms, such as Naive Bayes.

Zusammenfassung

Aufgrund des technologischen Fortschritts haben sich klassische Mobilfunkgeräte zu mobilen Computern entwickelt, welche innovative Techniken und Funktionen aufweisen. Aufgrund dieser Merkmale steigt der Verbreitungsgrad der *Smartphone* genannten Geräte kontinuierlich, wobei das Interesse nicht nur bei *gewünschten* Nutzergruppen gestiegen ist; seit dem Jahr 2004 konnte ein starker Anstieg an Schadsoftware für *Smartphones* identifiziert werden. Aktuelle Gegenmanahmen zu Schadsoftware für *Smartphones* beschränken sich auf Signatur-basierte Verfahren, welche in der Lage sind, *bekannte* Schadsoftware effizient zu erkennen. *Unbekannte* Schadsoftware kann aufgrund der fehlenden Signatur aber nicht erkannt werden, was wiederum ein Zeitfenster für schadhafte Aktionen öffnet. Aufgrund der steigenden Bedeutung der Smartphones und der darauf gespeicherten Daten für die jeweiligen Nutzer, ist es erforderlich, die Möglichkeit neuer signaturloser Ansätze, welche unbekannte Schadsoftware für Smartphone-basierte Umgebungen erkennen, zu untersuchen.

In dieser Arbeit betrachten wir das Forschungsfeld der *Smartphone*-basierten Schadsoftware. Wir geben eine klare Definition des Begriffs *Smartphone*, da es hierzu keine einheitliche Meinung, noch einen gemeinsamen Industriestandard gibt. Um die Gefahr von Schadsoftware für *Smartphones* besser nachvollziehen zu können, präsentieren wir zudem eine Zusammenstellung aller veröffentlichten Schadsoftware bis zum Ende des Jahres 2010.

Unsere vorgestellten signaturlosen Ansätze basieren auf Methoden aus dem Feld der *statischen* und *dynamischen Analyse*. In dem Feld der *dynamischen Analyse* stellen wir ein System vor, das Verhaltens- und System-basierte Informationen sammelt, welche auf einem entfernten System mit Hilfe von Verfahren des Maschinellen Lernens im Sinne der Anomalieerkennung verarbeitet werden. Diesem System führte zu einer allgemeineren Architektur zur Überwachung von Linux-basierten *Smartphones*, welche

wir nutzen, um *Systemaufrufe* aus Binärdateien zu extrahieren. Die Systemaufrufe wiederum werden genutzt, um Schadsoftware von *normaler* Software zu unterscheiden, welches wir in einem Baum-basierten Ansatz beschreiben.

Neben den Ansätzen der *dynamischen Analyse* diskutieren wir die Anwendbarkeit von *statischer Analyse* auf das Feld der Schadsoftwareerkennung in Smartphoneumgebungen, wobei Symbian OS und Android als Beispielplattformen dienen. In beiden Fällen extrahieren wir auf *statische* Art und Weise Funktions- und Systemaufrufe aus ausführbarem Code, um diese zu analysieren. Die Analysen geben Rückschlüsse auf die Absichten der untersuchten Datei, wobei die erzielten Ergebnisse vielversprechend sind.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation and Approach

Mobile phones have become the central computing and communication device today. Since August 2006, more mobile phones than inhabitants are registered in Germany [227]. As the capabilities of these devices increase, they are not simple voice-centric handsets any more; rather they represent one step towards realizing the vision of Mark Weiser [237] called *ubiquitous computing*. In this vision, Weiser describes that classical computers will be replaced by small, intelligent, distributed, and networked devices that will be integrated into everyday objects and activities. This replacement can be already observed in shops and warehouses using tags for monitoring and controlling items. But also the evolution of *smartphones* can be seen as part of this vision since they represent a possibility to making use of technical and computational capabilities in mobile context. *Smartphone* is a commonly used term for describing current comprehensive mobile phones where no global industry definition exists. A common understanding of this term is that these devices provide state-of-the-art technical characteristics as well as software development environments that allow creation of third-party applications.

With the increasing capabilities of such phones, more and more malicious software (malware) targeting these devices have emerged. In 2004, the first articles about malware for *smartphones* [53, 169] appeared describing mobile devices as the next generation of targets. Since then, the number of malware increased every month, and variants for various *smartphone* plat-

forms appeared, e.g. Symbian OS, Windows Mobile, and Android. Our assumption is that the evolution of malware for mobile devices might take a similar direction as the evolution of PC malware. Thus, similar problems will have to be encountered, e.g. missing signatures for unknown threats and new malware appearing at high frequency. For instance, Bulygin [35] showed that a MMS worm targeting random phone book numbers can infect more than 700,000 devices in about three hours. Another interesting work was presented by Oberheide *et al.* [162] who state that the average time required for a signature-based anti-virus engine to become capable of detecting new threats is 48 days. These numbers request extended security measures for *smartphones* as a malware can seriously damage an infected device within seconds.

Since Symbian OS was the major target of *smartphone* malware, Symbian introduced mandatory application signing in their OS in 2006[1] for coping with this problem. Application signing was performed by third-party companies where submitted applications are checked for meeting a certain set of requirements, e.g. proper memory handling and certificate level. The corresponding certificate basically grants access to different kinds of API calls basing on the privilege level. Although the signing mechanism was able to prevent distribution of malware targeting the 3rd version of Symbian OS S60 for about two years, finally it got broken by Mulliner [155]. After this publication describing the way of breaking the system, new malware appeared[2]. This event underlines the need for extended security measures for *smartphones* that are capable of detecting new and unknown malware.

Therefore, this thesis investigates and evaluates alternative approaches to signatures which are capable of detecting new malware for *smartphones* without using signatures. Here, we distinguish between approaches that do require execution of malware for analysis (*dynamic analysis*) and approaches that do not require execution (*static analysis*). Both approaches have their advantages and drawbacks which will be described in the corresponding sections. We use common *smartphone* platforms, like Symbian OS, Windows Mobile, and Android, for our experiments which allows us to generate and analyze realistic data.

[1] First S60 3rd device shipped in March 2006 named Nokia 3250 requiring signing.

[2] `http://www.f-secure.com/weblog/archives/00001609.html`, visited 28.07.2010.

1.2 Contributions

In this work, we consider the thesis that *smartphone* malware can be detected without using signature-based approaches. Therefore, we investigated different topics within the domain of *smartphones* supporting this thesis. Contributions were made to the research fields of I) *smartphones* in general, II) *smartphone* malware analysis, and III) *smartphone* malware detection without using signatures. In detail, the following contributions are made:

Smartphones: *Smartphone* is a commonly used term for describing current comprehensive mobile phones where no global industry definition exists. The *smartphone* follows the vision of Mark Weiser [237] in providing ubiquitous computing to its users; therefore, it can be seen as a milestone in computing history. We present the evolution of *smartphones* and explain their differences to classic computing devices. Additionally, we present small studies describing the usage of *smartphones* in the year 2010.

Smartphone malware: *Smartphones* get increasingly popular which also attracted malware writers beginning from June 2004. From this point on, malware count increased steadily. For understanding the threat of malware for *smartphones*, we gathered a list including all known malicious software until the end of the year 2010 and present their key-characteristics. Furthermore, we present a listing of current research on countermeasures.

Smartphone malware detection using *dynamic analysis*: The essential point about dynamic analysis is that data is acquired at runtime in comparison to *static analysis* which does not require executing binaries for investigating them. This can have the advantage that incidents are detected in real time enabling the system to start appropriate countermeasures in time. Our contribution in this field is threefold. First, we present our novel approach on monitoring Windows Mobile and Symbian OS devices for anomaly detection. Second, we describe an architecture that enables monitoring and detection of anomalies on Linux-based Android devices. Third, we explain our approach on applying *dynamic analysis* to data gathered from Linux *smartphone* binaries used in system call trees for malware detection.

Smartphone malware detection using *static analysis*: *Static analysis* has the advantage that it is not bound to the execution of binaries in order to work. It solely relies on the binaries themselves which are investigated in a static manner. Our contribution to the field of static malware detection on *smartphones* is twofold. First, we perform *static analysis* on executables from the Android platform in order to extract their function calls using the command *readelf*. Function call lists are compared with malware executables by classifying them with machine learning approaches, such as PART, Prism and Nearest Neighbor Algorithms. Our approach includes an option to share results in a collaborative manner decreasing the amount of newly infected devices significantly. Second, function calls are clustered for indicating malicious applications.

1.3 Outline

In Chapter 2, we present the evolution and characteristics of *smartphones*. We discuss similarities and differences to stationary computers and define the characteristics of a phone which make it a *smartphone*. After presenting the *usage of smartphones*, we give detailed insights to their security.

In Chapter 3, our ongoing research on *smartphone* malware is presented. We describe their evolution and present corresponding research on countermeasures.

In Chapter 4, we use *dynamic analysis* for user behavior- and application behavior-based detection of *smartphone* malware. We show that *smartphones* basing on Symbian OS or Windows Mobile can be monitored for extracting system characteristics indicating malicious activities. Additionally, we show that tracing execution of binaries for monitoring system calls can also be used for malware detection.

In Chapter 5, we present our research applying *static analysis* to the domain of *smartphone* malware detection. In detail, we investigate whether static call occurrences of function and library calls can be used in order to detect malware for Android and Symbian OS.

This thesis is concluded in Chapter 6 by summarizing this work and highlighting the results and contributions.

1.4 Summary of Research Activities

Aspects of this dissertation were published as a journal article [198], peer-reviewed conference papers [32, 22, 21, 194, 196, 199, 6, 200, 197], technical reports [86, 192, 190, 191, 195], and a poster [193]. Additionally, content of this work was used to teach students in seminar and project courses, as well as to find problems to be addressed in bachelor, master, diploma theses.

Journal

[198] Aubrey-Derrick Schmidt, Frank Peters, Florian Lamour, Christian Scheel, Seyit Ahmet Camtepe, and Sahin Albayrak. Monitoring smartphones for anomaly detection. *Mobile Networks and Applications*, 14(1):92–106, 2009.

Conferences

[32] Thomas Bläsing, Leonid Batyuk, Aubrey-Derrick Schmidt, Seyit Ahmet Camtepe, and Sahin Albayrak. An Android Application Sandbox System for Suspicious Software Detection. In *Proceedings of the 5th International Conference on Malicious and Unwanted Software (Malware 2010)*, Nancy, France, 2010.

[22] Christian Bauckhage, Tansu Alpcan, and Aubrey-Derrick Schmidt. A probabilistic diffusion scheme for anomaly detection on smartphones. In *Proceedings of the Fourth International Workshop in Information Security Theory and Practice 2010 (WISTP10)*, pages 3146, 2010.

[21] Leonid Batyuk, Aubrey-Derrick Schmidt, Hans-Gunther Schmidt, Ahmet Camtepe, and Sahin Albayrak. Developing and benchmarking native linux applications on Android. In *Proceedings of the 2nd Mobile Wireless Middleware, Operating Systems, and Applications (MOBILWARE 09)*, 2009.

[194] Aubrey-Derrick Schmidt, Rainer Bye, Hans-Gunther Schmidt, Jan Clausen, Osman Kiraz, Kamer Yüksel, Ahmet Camtepe, and Sahin Albayrak. Static analysis of executables for collaborative malware detection on android. In *Proceedings of the IEEE International Congress*

on Communication (ICC) 2009 - Communication and Information Systems Security Symposium, pages 15, Dresden, Germany, June 2009.

[196] Aubrey-Derrick Schmidt, Jan Hendrik Clausen, Seyit Ahmet Camtepe, and Sahin Albayrak. Detecting symbian os malware through static function call analysis. In *Proceedings of the 4th IEEE International Conference on Malicious and Unwanted Software (Malware 2009)*, pages 1522. IEEE, 2009.

[199] Aubrey-Derrick Schmidt, Hans-Gunther Schmidt, Leonid Batyuk, Jan Hendrik Clausen, Seyit Ahmet Camtepe, Sahin Albayrak, and Can Yildizli. Smartphone malware evolution revisited: Android next target? In *Proceedings of the 4th IEEE International Conference on Malicious and Unwanted Software (Malware 2009)*, pages 17. IEEE, 2009.

[6] Sahin Albayrak, Katja Luther, Rainer Bye, Stephan Schmidt, Aubrey-Derrick Schmidt, and Karsten Bsufka. Autonomous security - eine neuartige architektur für netzwerkbasierte intrusion detection und response. In Christian Paulsen, editor, In *Tagungsband des 15. DFN Workshop Sicherheit in vernetzten Systemen*, pages G1 G19. DFNCert Services GmbH, 2008. ISBN:978-3-833-4-7381-4.

[200] Aubrey-Derrick Schmidt, Hans-Gunther Schmidt, Jan Clausen, Kamer Ali Yüksel, Osman Kiraz, Ahmet Camtepe, and Sahin Albayrak. Enhancing security of linux-based android devices. In *Proceedings of 15th International Linux Kongress*. Lehmann, October 2008.

[197] Aubrey-Derrick Schmidt, Frank Peters, Florian Lamour, and Sahin Albayrak. Monitoring smartphones for anomaly detection. In *Proceedings of the 1st international conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications (MOBILWARE 08)*, pages 16, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

Technical Reports

[86] Hans-Gunther Schmidt, Karsten Raddatz, Aubrey-Derrick Schmidt, Seyit Ahmet Camtepe, and Sahin Albayrak. Google Android - A Com-

prehensive Introduction. Technical Report Technical Report TUB-DAI 03/09-01, Technische Universität Berlin, August 2009.

[192] Aubrey-Derrick Schmidt. [Abstract] Smartphone Malware Evolution Revisited. In Proceedings of the Fourth GI Graduate Workshop on Reactive Security (SPRING), number SR-2009-01 in GI SIG SIDAR Technical Reports, Stuttgart, Germany, 15 September 2009.

[190] Aubrey-Derrick Schmidt and Sahin Albayrak. Malicious software for smartphones. Technical Report TUB-DAI 02/08-01, Technische Universität Berlin, DAI-Labor, February 2008.

[191] Aubrey-Derrick Schmidt. [Abstract] Anomaly detection on smartphones. Proceedings of the Third GI Graduate Workshop on Reactive Security (SPRING), number SR-2008-01 in GI SIG SIDAR Technical Reports, Stuttgart, Germany, August 2008.

[195] Aubrey-Derrick Schmidt, Rainer Bye, Hans-Gunther Schmidt, Kamer Ali Yüksel, Osman Kiraz, Jan Clausen, Karsten Raddatz, Ahmet Camtepe, and Sahin Albayrak. Monitoring android for collaborative anomaly detection: A first architectural draft. Technical Report TUB-DAI 08/08-02, Technische Universität Berlin - DAI-Labor, August 2008.

Poster

[193] Aubrey-Derrick Schmidt, Static Smartphone Malware Detection. In proceedings of the 5th Security Research Conference (Future Security 10), Berlin, Germany, 2010

Supervised Master Theses

- Bo Su, A Taxonomy of Intrusion and Malware Detection Systems for Smartphones, Technische Universität Berlin, November 2010
- Jinchao Shen, Smartphone Security, Technische Universität Berlin, November 2010
- Yanlong Li, A new Approach to User Authentication on Smartphones, Technische Universität Berlin, December 2010

- Yanfang Song, Evolution of Smartphone Malware, Technische Universität Berlin, July 2011
- Xiaoshuan Gu, A Taxonomy of Intrusion Detection Systems, Technische Universität Berlin, July 2011
- Xin Zhang, A Taxonomy of Malware Detection Systems, Technische Universität Berlin, July 2011

Supervised Diploma Theses

- Tobias Himmelbach, An Analysis of Today's Usage of the Smartphone, Technische Universität Berlin, Dezember 2010
- Thomas Bläsing, An Android Applications Sandbox for Suspicious Software Detection, Technische Universität Berlin, 2010
- Tayfun Sözgen, Optimierung der Datenverwaltung zur effizienten Kommunikation und Speicherung in einem Daten-intensivem Intrusion Detection System, Technische Universität Berlin, 2007

Supervised Bachelor Theses

- Lars Borchert, Analyse und Nutzung heutiger Smartphones, Technische Universität Berlin, November 2010
- Jonathan Ziller, Methods of Artificial Intelligence for Context-based Anomaly Detection, Technische Universität Berlin, August 2010

Supervized Seminar Papers

- Karsten Raddatz, New Authentication Approaches, winter term 2009/2010
- Markus Herpich, Access Control, winter term 2009/2010
- Steffen Bingel, Security Levels, winter term 2009/2010
- Gilbert Assaf, Antivirus Software, winter term 2009/2010
- Leo Bronstein, Web service Vulnerabilities through JavaScript, summer term 2009

- Iris Breddin, Smartphone Malware Evolution, summer term 2009
- Lars Borchert, Smartphone Nutzung, summer term 2009
- Aldin Sljivar, Smartphone Honeypots, summer term 2009
- Sascha Narr, Smartphone Security, summer term 2007
- Thebin Lee, Windows Mobile, summer term 2007

Chapter 2

Smartphones - Ubiquitous Computing Devices

Smartphone is a commonly used term for describing current comprehensive mobile phones where no global industry definition exists. The smartphone follows the vision of Mark Weiser [237] in providing ubiquitous computing to its users and, therefore, can be seen as a milestone in computing history. This thesis, in particular, refers to the security of smartphones. For understanding the importance of this field, a general understanding of this term will be discussed first.

Since no global industry definition on the term *smartphone* exists, various published definitions are checked in Section 2.1 where a condensed one is presented finally. In Section 2.2, hard- and software characteristics of smartphones are shown where an essential discussion on the differences between smartphones and *classic* computers is presented. Since smartphones gain more and more popularity, benefits in terms of possibilities how to use them is presented in Section 2.3. In Section 2.4, security aspects of smartphones are presented. The evolution of smartphones can be found in Appendix B.

2.1 Definition

Since there is no common industry definition or understanding of the term smartphone, we will discuss the various opinions on that term in this section

for finally giving our own condensed definition. Giving an own definition is necessary since the presented opinions often exclude devices from competitors. In our definition we will try to cover the main characteristics of smartphones as a basis for the usage of this term throughout this work.

Best [29] collected several descriptions on smartphones, including the definition of Gartner: "A large-screen, data-centric, hand-held device designed to offer complete phone functions whilst simultaneously functioning as a personal digital assistant (PDA)". A further description from Jason Langride from Microsoft UK that Jo Best collected was: "For us, smart phones combine traditional communication devices and provide rich applications and rich data applications". David Wood from Symbian Ltd. says: "Smart phones differ from ordinary mobile phones in two fundamental ways: how they are built and what they can do. The way they're built - using open systems to take advantage of the skills, energy and innovation of numerous companies from a vast range of industries - means that smart phones extend the phenomenal track record of mobile phones by improving constantly and rapidly, year by year".

Fulton [95] refers to a report from the market research company ABI describing what a smartphone is *not*. In this report, the early Apple iPhone is categorized as "feature phone" and not a smartphone since it lacked the possibility of installing native third-party applications. In the field of telecommunication the term "feature phone" is used to describe classical mobile phones that were extended by several features, e.g. calendar, calculator, or applications running through a middleware like Java ME. According to Fulton, ABI uses the following definition for smartphones: "a cellular handset using an open, commercial operating system that supports third-party applications".

One essential differentiation between a smartphone and sophisticated feature phone is: a smartphone uses an operating system that supports *native* third-party applications. Nowadays, the term *commercial* in not applicable any more, since open source operating system emerged, e.g. Android[1] and Openmoko[2]. Therefore, we will omit such terms relying on the assumption that these operating systems are evolving through time providing updates to vendors and users while not being only a proprietary firmware as known from feature phones. Additionally, the term *native* clarifies that

[1] `http://android.com`, visited 15.3.2010.
[2] `http://www.openmoko.org/`, visited 15.3.2010.

running applications through a separate middleware, like Java ME, will not make a smartphone. Another essential point being included in most statements presented here is that a smartphone is a mobile phone, sized hand-held. This means it is easily usable by holding it in hands and prevents sub-notebooks with phone functionality being categorized as smartphones. Regarding the statements on smartphones presented here, one common characteristic was omitted by all: the connectivity. Most smartphones provide several interfaces allowing wired and wireless communication reflecting its actual purpose being a communication device.

Summing up the discussed points in section, we make the following definition which is applicable throughout this work:

Definition 1 *A Smartphone is a mobile hand-held phone that uses an operating system supporting native third-party applications and includes multiple communication interfaces for providing connectivity.*

2.2 Characteristics

Following Def. 1, we can state which criteria mobile phones have to fulfill in order to be categorized as smartphones. Besides this definition, it is also important to see the difference between smartphones and *classic* computers. If there would not exist any differences, research on smartphones would get obsolete since mature results from *classic* computers could be applied to the field of smartphones. Additionally, for getting better insights into smartphone platforms, hard- and software characteristics will be explained.

2.2.1 Differences between Computers and Smartphones

Smartphones represent compact mobile *computer* hand-helds showing similarities to *classic* computers, like PCs or laptops, on hard- and software level. Both platforms use an operating system that allows the installation of third party software. In some cases, computer operating system components are reused on smartphone side, e.g. Linux parts in Android, Mac OS X parts in the iPhone, and Windows parts in Windows Mobile. Additionally, common file formats, like MP3, Video, or PDF files, allow cross platform usage of data. On hardware side, further similarities can be seen:

both platforms use a central processing unit (CPU), memory, persistent storage, and in- and output devices.

In turn, the major differences between computers and smartphones can be seen in the compact and mobile nature of smartphones. Due to their hand-held size, smartphone architectures are planned on a very limited space meaning mostly highly integrated circuits are used for their production. The limited size also restricts the size of the battery, probably one of the most essential components in a smartphone providing energy for all consumers, e.g. the CPU or display.

Although processing capabilities of smartphones increase steadily[3], and might also even up with computers in future, the capacity of the battery determines whether corresponding components can be integrated or not. The CPU, for example, will normally increase power consumption when working on a higher clock speed and communication technologies can vary in terms of energy usage[4].

Another important point is that unlike laptops, smartphones are intended to run permanently making their owner able to be called at any time and any place. Due to this intention, maximizing the time that a smartphone can be used without recharging it, is an important goal. This goal is seen as essential requirement not only for smartphone manufacturers but also for network providers and application developers. Comparing this with an laptop or computer application, it is currently not imaginable that developers will consider energy constraints in their software.

Furthermore, smartphone provide various techniques for wireless communication, e.g. UMTS, Wi-Fi, IrDA, Bluetooth, GSM, and GPRS. Computers like laptops can also support these technologies but need to be upgraded in most cases. Smartphones provide these technologies out of the box.

Summing up the key differences between *classic* computers including PCs and laptops, it can be stated that smartphones differ from these systems due to (I) their highly integrated hand-held size, (II) their optimization towards battery life time[5], and (III) their out of the box support of various communication technologies.

[3]See birth of Moore's Law from [147].

[4]3G (UMTS) normally consumes more power than Wireless LAN which can be seen on the power usage statistics of the Google Nexus One smartphone [102].

[5]Optimization in terms of how long a battery will last without being recharged.

2.2.2 Hardware Characteristics

A general description on smartphones can be seen on Figure 2.1 which shows an example architecture of a smartphone basing on [161]. Most of the highlighted components are realized through integrated circuits combining various functionalities. In this section, a subset of these components is introduced for describing key hardware characteristics of smartphones.

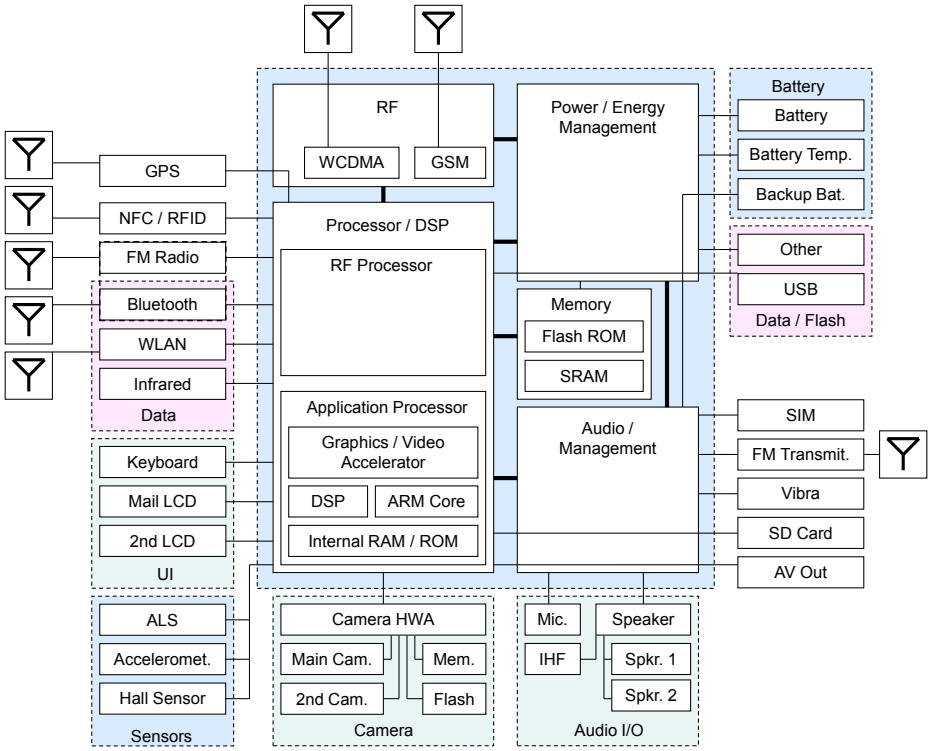


Figure 2.1: Sample smartphone architecture diagram based on [161].

Several smartphones and other hand-held devices base on an ARM architecture, e.g. the Apple iPhone, the Motorola Droid, but also the Nintendo DS[6]. Figure 2.1 is an example for this. The core component, situated in the center of Figure 2.1, manages all parts of the smartphone, especially the input and output data. Processors handle application-related instructions but also radio frequency (RF) related instructions for establishing connections to phone networks. Attached components handle battery,

[6] `http://en.wikipedia.org/wiki/ARM_architecture`, visited 25.02.2010.

data, external memory, user interface, sensors, cameras, audio, and wired or wireless connections.

The most essential functionality of a smartphone remains to be making phone calls. Therefore, smartphones include a microphone and speakers for providing the ability to talk to others when not using a headset. Using the subscriber identity module (SIM) card[7], users are allowed access to mobile phone networks through assessment and validation of the international mobile subscriber identity (IMSI) in a home location registry (HLR). SIM cards can be locked through a personal identification number (PIN) where it is important to say that entering this PIN only grants access to the network on most phones, the device is not protected through this measure. Replacing the SIM card from a stolen phone will lead to full access to device, data, and on-device services in most cases.

Table 2.1: Comparison of resistive and capacitive touch screens [133]

Capability	**Resistive**	**Capacitive**
Visibility indoors	Typically very good	Typically very good
Visibility sunlight	Typically poor, the extra layer reflects too much ambient light	Typically very good
Accuracy	At least pixel-wise	Theoretically accuracy within few pixel, but practically limited by finger size
Costs	"cheap"	up 10-50% more expensive than resistive screen
Robustness	Pressure-based approach requires soft top layer which is vulnarable to damage	surface can be made of resistant glass
Working temperature	-15°C to 55°C	0°C to 35°C

Beside these basic capabilites given through the described components, smartphones can provide much more functionality. A smartphone is a mobile device that mostly unifies components and resulting functionalities of a

[7]As shown on center right side.

cellular phone, a PDA, an audio player, a digital camera and camcorder, a Global Positioning System (GPS) receiver, and a PC. Smartphones formerly often used PC-like *QWERTY* keyboards in order to increase typing speed and sometimes PDA-like pen displays for improved data and command handling. Mechanisms were developed that additionally improve text input, like "Text on 9 keys" (T9) which represents predictive text technology.

Nowadays, more and more touch screen LCDs are built into smartphones basing on a capacitive panel. These panels base on the ability of humans to influence electrostatic fields, which is measurable in change of capacitance [203]. Alternatively, resistive panels can be used which base on physical pressure creating an electrical connection between two separated layers. Both, capacitive and resistive panels have their advantages and drawbacks, shown on Table 2.1 which bases on [133].

External devices can be attached to most smartphones e.g. for using additional storage space or being used as storage itself via USB. Additionally, memory cards like the Secure Digital (SD) card can be inserted into a smartphone, on some device even while the system is running (hot swap), e.g. on Nokia N93. Current high capacity cards can hold up to 128 GB. Inserted SD card can be read and written through the phone itself or through a computer attached via USB.

In many cases, smartphones provide access to a main camera on the back of the phone and sometimes to a secondary front camera for video calls.

Smartphones use different techniques for creating wireless connections for communication purpose:

- GSM represents the second generation (2G) of mobile end-to-end communication, mainly used for voice calls and services like SMS[8].
- GPRS in combination with 2G is often described as 2.5G, as it provides voice and packet data.
- W-CDMA[9] was designed as replacement of GSM and is used in the FOMA system in Japan and UMTS[10], being able to transport data at higher speed than GSM.

[8]Short Message Service (SMS).
[9]Wideband Code Division Multiple Access (W-CDMA).
[10]Universal Mobile Telecommunications System (UMTS).

- Near Field Communication (NFC) is modern technique getting deployed on smartphones. A key field of application is mobile payment.

Additionally, the devices provide Bluetooth, Wireless LAN (WLAN), or IrDA[11] support for shorter range wireless connectivity. Using one of these connections, a user is able to make phone calls, use an Internet browser, play multi-player games, or read emails.

Smartphone use various sensors for giving the operating system, applications, and user's information on their device and environment. The Ambient Light Sensor (ALS) can be used for controlling the brightness of the screen. The hall sensor is used for determining the position of the phone while the accelerometer is used to identify movements. These both sensors enables corresponding smartphones to detect motions which can be used in various ways, e.g. to remote control a toy car through moving the phone.

2.2.3 Software characteristics

In this section, the operating systems with the biggest share among smartphone platforms will be introduced as well as corresponding application development. This will raise awareness for threatened platforms and devices.

Operating Systems

Most mobile devices use proprietary OSs, which has the disadvantage that only few or even no additional software is available. On most smartphones this disadvantage does not exist as they mostly use one of the following standardized operating systems that allow installation of native software. Providing the ability to install additional applications allows users to customize a device according to their software needs. Following Canalys [37], the main competitors in this field are: Symbian OS from Symbian Ltd. [138] 46.2%, Research In Motion (RIM) [96] with its Blackberry hand-helds 20.6%, Apple [100] 17.8%, Microsoft [51] 8.8%, and Google Android [9] 3.5%, where the percentages show the worldwide share on the smart phone market, respectively. Other hold only 3.2%. Comparing the shares with growth rates,

[11] Infrared Data Association (IrDA).

Android and iPhone will gain more while Symbian and Windows Mobile will lose importance.

Symbian Symbian Limited is a software producing company located in London, UK. Formerly, it is was owned by several companies, like Ericsson, Nokia, Panasonic, Samsung, Siemens and Sony Ericsson. In December 2008, Nokia took over Symbian Ltd. for transferring it to the *Symbian Foundation*. According to [138], the aim of the Symbian Foundation is "To bring to life a shared vision to create the most proven, open and complete mobile software platform - and to make it available for free." [138]. Current smartphone manufacturers that license Symbian OS are Samsung and Sony Ericsson.

Symbian OS uses three security methods: *capabilities*, *installation file signing*, and *data-caging*. Capabilities limit access to sensitive APIs. There are three levels of limitation where on the highest level full device and network access is granted to the corresponding application. These limitation levels are defined by certificates that are used to sign Symbian OS Installation System (.SIS) files. Without a valid signing, it is not possible to install application on Symbian OS devices[12]. Data-caging extends this approach as it limits access to the file system. Depending on the limitation coming from the certificate, application can only write to certain areas, like the application folder, user data folder, or system folder.

Symbian OS holds the greatest share on the world wide smartphone market with 46.2% which can be seen as its biggest advantage in comparison to the competitors. One drawback about the current Symbian system is that it is facing major changes which is shown in [139]. This can lead to a complete new architecture forcing developers to program their applications again from scratch. Due to its high share, research work in Section 4.3 is realized on Symbian OS.

BlackBerry (RIM) Research In Motion (RIM) provides proprietary operating systems for its BlackBerry devices. BlackBerry devices got known for their ability to send and receive emails (push email) attracting business managers and other persons relying on email messages. Today, various people use BlackBerries resulting in 20.6% worldwide market share and therefore being number two in the smartphone world. Applications

[12]Applicable to Symbian S60 3rd.

for Blackberry devices can be combinations of native user interfaces and java-side application logic. Hence, security mechanisms base on Java-level permission that can be customized to developers expectations. Most BlackBerry devices use a QWERTY-style keyboard which can speed up email writing.

A central advantage of the BlackBerry platform is its push email technology allowing instant transmission of emails to the devices. A disadvantage of the BlackBerry platform is that it does not support full native applications decreasing the possibilities for developers to interact with the system. Although the BlackBerry platform is very interesting, no research work was performed on this system.

Apple iPhone The Apple iPhone is a very interesting device that can be classified as smartphone since the day Apple released a SDK for it in early 2008 [100]. The device runs a modified version of Mac OS X, called iOS, and includes several applications, e.g. the Safari browser, a music player, and digital camera. Third-party software can be developed with the SDK through subscribing to the Apple iPhone developer program.

Although the device did not introduce purely new technology on release, the combination of functionality and design had a significant impact on the smartphone market. Since release in June 2007, the iPhone gained a 17.8% share on the smartphone market. Additionally, the iPhone can be seen as the main driver of smartphone Internet usage. The iPhone has a 51% share in submitted Internet requests, making it market leader according to [97]. In terms of smartphone applications, Apple is also market leader. Apple announced in September 2009 that it's application store[13] just surpassed the number of two billion downloads, providing more than 85,000 applications for more than 50 million iPhone and iPod touch customers [99].

According to the presented numbers, the iPhone is leader in several smartphone-related fields, e.g. smartphone Internet usage, which can be seen as its biggest advantage. Despite this leading role, iPhone development is restricted to non-critical libraries making it hard to create system-level applications, which most security application are. Therefore, no research work of this thesis was performed on the Apple iPhone.

[13] Apple online application store is called App Store.

Windows Mobile The Windows Mobile operating system is based on Windows CE and was developed for mobile devices like PocketPCs, PDAs, smartphones, and embedded systems (e.g. smart fridges [51]). The current version Windows Mobile 7 is also called Windows Phone 7. Windows Mobile security employs three major approaches: *security roles*, *security policies*, and *application signing*. Security roles define users or groups having pre-set rights on a device. The most privileged role allows changing security policies, which are rules permitting certain actions on the device, e.g. installing and running unsigned applications. Application signing principles of Windows Mobile are very similar to the ones of Symbian OS. Basically, Windows Mobile software should be signed in order to permit access to sensitive APIs[14].

Windows Mobile holds a 8.8% percent share on the smartphone market where its major advantage can be seen in its interoperability with other Microsoft operating system, e.g. Windows 7. A major disadvantage of windows mobile systems was their former insecurity, which was also stated by the UK Communications-Electronics Security Group (CESG) [52]: "The current CESG policy/ guidance states that Windows Mobile version 6.1 is not deemed suitable to access, store or process RESTRICTED (IL3) data."

In this thesis, very early research work was realized on Windows Mobile. In Section 4.3, a monitoring client for anomaly detection was implemented that collected information on the hosting Windows Mobile system.

Google Android Google Android [9] is a software stack that includes an operating system, middleware and basic applications. The first Android device was released in October 2008 being named T-Mobile G1 while being produced by HTC [49].

The Android system is built upon the Linux 2.6 Kernel and supports most of its functionalities. Android treats every application equal meaning both, that a developer is able to replace every single Android program and an Android application can be run on any Android device only being limited by the provided functionalities[15]. Google Android security mechanisms are based on those of a Linux system. Access control, e.g. user and group IDs, is managed where every installed application gets its own user ID with its specific permissions. These permissions allow finer-grained access

[14]Security policies can make application signing unimportant.

[15]Example functionalities are used in Internet tablets and navigation systems.

adjustment for processes using certain functionalities, e.g. sending SMS messages or dialing a phone call. As it is an open platform and its possible large market share, one can expect that Android should be in the focus of most malware developers.

Since the first devices were released, Android gained a market share of 3.5% on the smartphone market while it is also deployed to other systems, e.g. netbooks [101], tablets [122], or electronic paper devices (e-ink) [121].

Smartphone Software Development

Developing, building, and testing smartphone software requires tools which are often included in a Software Development Kit (SDK) or Integrated Development Environment (IDE). Since applications are a central aspect of smartphones, these software *packages* will be briefly introduced in this section.

The SDK A SDK is a collection of software that gives a software developer the ability to create and deploy software for a certain framework, platform, operating system, programming language or hardware. Most SDKs are freely available on the web pages of the corresponding manufacturers. Example SDKs can be found in Android or iPhone SDK [9, 100]. Most SDKs are delivered with a software emulator.

The Software Emulator A software emulator gives a developer the ability to run and test software on his computer though it is developed for other systems or platforms, e.g. Symbian OS devices. This can reduce costs, since software prototypes can already be programed and run without buying a real device. On the other hand, the emulator often does not support all functionalities[16] of a real device. This can lead to serious problems, if not-supported functionalities have a severe impact on the program stability.

In general, unlike a simulator which reproduces program behavior, an emulation attempts to reproduce the same states real devices would enter at corresponding points. Regarding current SDKs, most of the so called software emulators only *simulate* connections, interfaces, and functionalities through mapping e.g. simulator Bluetooth port to PCs serial port.

[16] e.g. connectivity like GSM or UMTS

Therefore, the use of the term *simulator* would be more appropriate than the term emulator which is currently commonly used.

The IDE The Integrated Development Environment is very similar to SDKs, it often combines tools to be able to write, compile, build and debug software. The main difference is that today's IDEs integrate most tools into one single tool that has graphical user interface (GUI). Then, most relevant actions can be performed through the user interface, which often speeds up development. Examples for IDEs for mobile devices are MS Visual Studio, Metroworks Codewarrior, Nokia Carbide, and Eclipse.

2.3 Usage

Increasingly, smartphones have become the platform of choices for both business and consumer. The class and types of applications being deployed in the current generation of smartphones provide a compelling argument for the need of increased security. For example, most banking institutions worldwide have deployed smartphone applications that have access to a customer account, and its sensitive data. Hence, understanding this emerging trend is critical to our understanding of the compelling reasons for securing the smartphone platform and corresponding infrastructure. This topic will be covered next.

AdMob Inc., a subsidiary of Google Inc. that is specialized on mobile advertisement, stated in its monthly mobile metrics report for February [98] that smartphones surpassed feature phones in terms of Internet traffic share in October 2009. Considering the smaller amount of smartphones in comparison to the number of feature phones, this fact underlines the meaning of smartphone browsers and the possibility to use various communication technologies for interconnection. Therefore, in this section, we will start describing smartphone usage in the year 2005/06 when our research in this field started. Additionally, we will highlight changes from the past surveys to our current numbers.

2.3.1 Smartphone Usage in the Years 2005/2006

In this section, we refer to a survey [213] and a monitoring experiment [230] which investigated the actual usage of smartphones.

Table 2.2: TNS GTI Top-10 applications/services 2005

No.	Application	Usage
1.	SMS	83%
2.	Games	61%
3.	Camera	49%
4.	MMS Picture	46%
5.	PDA Functions	36%
6.	Internet	31%
7.	WAP	30%
8.	Bluetooth	28%
9.	Email	27%
10.	Video Camera	27%

In Global Technology Insight 2005 [213], TNS Technology identified the most used applications in 2005. The results base on data coming from 6807 people using a mobile phone (6517 persons), PDA or laptop and accessed the Internet at least once a week. The study partly focused on the adaptation of technology applications on mobile devices, which we used to excerpt the top ten applications. Table 4.6 shows the extracted Top-10. It is interesting to see that the most used applications from this survey were messaging, games, and the camera while presented results are slightly biased through participation of a small number of laptop users.

Verkasalo et al. [230] monitored 562 smartphone users over a period of six months where only devices were taken into account that at least provided data from 21 active days. Table 2.3 shows the results for the overall Top ten applications while Table 2.4 shows the top ten third-party applications.

2.3.2 Smartphone Usage in the Year 2010

For verifying the somehow outdated numbers from 2006, we conducted a survey on smartphone usage including 146 participants, mostly from Germany. This survey is part of a German diploma thesis by Tobias Himmelbach where the work still needs to be published.

Table 2.3: Top-10 applications by launches 2006

No.	Application	Usage
1.	Logs	100%
2.	Messenger	100%
3.	Phonebook	100%
4.	Calendar	93%
5.	Browser	91%
6.	Profile application	90%
7.	Clock application	89%
8.	Camcorder	88%
9.	Calculator	81%
10.	Application Manager	74%

Table 2.4: Top-10 3rd party application by launches 2006

No.	Application	Usage
1.	File explorer	29%
2.	Browser	25%
3.	Picture application	17%
4.	Messenger	17%
5.	Text processing application	16%
6.	Game	15%
7.	Game	13%
8.	Picture Viewer	13%
9.	Bloging tool	12%
10.	Picture Editor	12%

In this survey, interesting points could be observed regarding the usage of smartphones. Although not considered in [213], we asked the participants whether they actually use their smartphone for making phone calls or not. Interestingly, only 97% responded to this question positively while first assumptions were that the missing 3% might be a statistical error ratio. But when conducting an internal survey for a major German telecommunication service provider with more than 500 participants, we got the exact same number. While no clear reason can be given for this, some participants might use their smartphone only for navigation or as music player.

Comparing the numbers from the year 2006 with the ones of 2010 which are also shown on Table 2.5, one can see that sending text messages kept

one of the most used services of smartphones. *Internet* usage obviously increased moving from sixth place to third place. *Clock* usage was not considered directly in [213] but still is one of the most used functions on a smartphone. *Camera* usage increased from 49% in the year 2006 to 81%. A very interesting change was the usage for navigational tasks (80%). Another interesting point is the increased usage of a smartphone for listening to *music*. This can be explained with the increasing capabilities and storage smartphones provide and being directly addressed in advertisement.

Table 2.5: Smartphone usage survey results from 06/2010 showing percentage of users using certain applications on a regular basis.

No.	**Application**	**Usage**
1.	Telephone	97%
2.	SMS	91%
3.	Internet	90%
4.	Email	87%
5.	Clock	87%
6.	Camera	81%
7.	Navigation	80%
8.	PDA	80%
9.	Music	71%
10.	Games	67%

2.4 Security

This thesis describes security techniques for detecting malicious applications on smartphones. Therefore, background information on relevant security principles is presented in this section. Moreover, security of smartphones is explained through listing prevalent threats and corresponding security measures to them. Readers familiar with basic security concepts and paradigms can jump over to Section 2.4.2.

2.4.1 Security Background

Smartphones represent compact mobile computer hand-helds. This allows us to apply known security definitions and principles to them. Starting with

the term "security" itself: one interpretation of this word is the condition

Table 2.6: Summary on security goals according to [31, 217, 164]

Confidentiality	Data that is transmitted or stored should only be revealed to an intended audience
Integrity	Modification should be possible to detect and the creator should be identifiable
Availability	Services should be available and function correctly

of being protected against danger or loss. The Department of Defense in the U.S.A. defines it as "a condition that results from the establishment and maintenance of protective measures that ensure a state of inviolability from hostile acts or influences" [163]. Bejtlich [26] states that security is the process of maintaining an acceptable level of perceived risk where the security process revolves around four steps: assessment, protection, detection, and response. Bishop [31] introduces the term "security goal" to be able to describe objectives that have to be achieved in order to state a computer system or network is secure. These goals are *confidentiality*, *integrity*, and *availability* where a summary on these terms is given on Table 2.6.

Confidentiality According to the United States Code (U.S.C.) [217], *confidentiality* refers to preserving authorized restrictions on information access and disclosure, including means for protecting personal privacy and proprietary information. Generally speaking, *confidentiality* refers to limiting access to data and information to authorized persons. In case of computer systems, authentication methods, like user name and password or biometric fingerprint recognition, can authorize access to a system. The American NIST[17] [164] states that a loss of *confidentiality* is the unauthorized disclosure of information.

An example for keeping *confidentiality* on a certain file is to control access to it through user file system rights. A certain user can be assigned sole ownership and right to read, write, and execute the file. An example for losing *confidentiality* is, if an attacker is able to escalate his system rights to root level.

[17]National Institute of Standards and Technology (NIST).

Integrity Following the U.S. Code [217], *integrity* refers to guarding against improper information modification or destruction, and includes ensuring information non-repudiation and authenticity. Bishop [31] states that *integrity* includes data *integrity* and origin *integrity*. Data *integrity* assures that the data is free of modifications or corruptions. Origin *integrity* guarantees that the source of data and information is marked correctly. Bishop [31] additionally explains that *integrity* methods fall into two classes: prevention mechanisms and detection mechanisms. Prevention mechanisms aim for maintaining *integrity* while detection mechanisms try to identify possible alteration of data and information. NIST [164] states that a loss of integrity is the unauthorized modification or destruction of information.

An example for proving existing data *integrity* might be realized by checking a created collision-free hash code on a certain file. An example for loss of data integrity can be seen in a list of grades from a teacher that was modified by a student for his own benefit. A very simple example of lost origin integrity[18] can be seen in the assessment of most scientific works, e.g. bachelor, master, and Ph.D. theses. Each work has to be free of plagiarism else origin *integrity* would be broken.

Availability U.S.C. [217] defines *availability* as ensuring timely and reliable access to and use of information. Hence, *availability* describes whether a resource or information can be used in a timely fashion or not. NIST [164] describes that a loss of availability is the disruption of access to or use of information or an information system.

Denial of Service (DoS) attacks are a common example for disrupting online services, as seen in August 2009 when twitter services[19] were note reachable for hours [240]. Besides obvious DoS attacks, *availability* can also be harmed by unintended action, e.g. when the cleaning sir/lady removes a power plug in server center for plugging in the vacuum cleaner.

The goals presented on Table 2.6 can be harmed through the following threats [31]: *eavesdropping*, *modification*, *masquerading*, *repudiation*, *denial of receipt*, *delay*, and *denial of service* where a summary on these terms is shown on Table 2.7. A threat is a potential violation of security meaning

[18] This example also applies to data integrity.

[19] `https://www.twitter.com`, visited 15.3.2009.

that the violation does not actually need to occur but need to be protected against [31]. Actions that lead to a violation are called *attacks*; those who perform them are called *attackers* [31].

Table 2.7: Security threats according to [31]

Eavesdropping or Snooping	An entity reads information that it is not intended to read
Modification or Alteration	Data is being altered or destroyed
Masquerading or Spoofing	An entity claims to be another
Repudiation	An entity falsely denies participation in an act
Denial of receipt	An entity falsely claim not to have received a delivery object
Delay	The delivery on an object is delayed
Denial of service	Any action that aims to reduce the availability and / or correct functioning of services or systems

Eavesdropping *Eavesdropping* describes the unauthorized interception of information and is also called snooping. Examples for *eavesdropping* are reading post cards that are not addressed to you or monitoring (wireless) network traffic[20], e.g. for capturing user-names and passwords. In all cases, *eavesdropping* is passive. Measures to maintain *confidentiality* can counter this threat [31].

Modification *Modification* describes the unauthorized change of information and is also known as alteration [31]. Since *integrity* measures address the threat of modification the same examples apply here: a student that breaks into the central computer of his school in order to alter a list of grades represents the modification threat. Another real life example for modification is whenever people exchange price tags in shops in order to pay less than actually needed.

Masquerading The threat of *masquerading*, which is also called spoofing, is given whenever an entity claims to be another entity. A very simple example is the usage of eavesdropped account login credentials. A common real life example is the usage of faked identity cards by

[20]This method is also called passive wiretapping.

under age persons in order to buy alcohol or to enter a discotheque. *Masquerading* is addressed by methods that maintain *integrity* [31].

Repudiation The threat that an entity falsely denies participation in an act is called *repudiation of origin*. An example for this, which is also given in [31], is if a customer orders an expensive product and denies having ordered it when it gets delivered. Integrity mechanisms cope with this threat [31].

Denial of receipt If an entity claims that it did not receive an information although it did, this is described as *denial of receipt* [31]. Using a similar example as before: if the customer receives an expensive product but denies this by asking the vendor whether it was already shipped or not, this can be seen as *denial of receipt*. In case of computer systems, methods to ensure *integrity* and *availability* counter this threat.

Delay *Delay* is a threat that includes all actions that lead to a *delay* of delivery of an object. An attacker can, e.g. *delay* the forwarding of an email that warns employees of a company not to use a certain service since it was misused for phishing purpose right until the people use the service. *Availability* methods target this threat [31].

Denial of service *Denial of service* is a threat that bases on preventing objects to be used at a certain or any time. *Denial of service* attacks can be realized through exploiting communication protocol flaws that lead to states not allowing the system to answer (e.g. timeouts). This threat is of special interest whenever companies or institutions are relying on responsiveness of their services, e.g. in case of online shops or online trading. But not only commercial services are critical; if an attacker succeeds in performing a *denial of service* attack on the communication system of the police, it will be hard to coordinate the police cars and troops. The *denial of service* threat can be countered with measures ensuring *availability* [31].

2.4.2 Security of Smartphones

As shown in Section 2.4.1, several threats to computers are known which also apply to smartphones. In this section, a more detailed description on these threats will be given while suitable security mechanisms that can

counter them are presented, too. What will not be given is a complete taxonomy on smartphones threats and attacks. Various researchers tried to find a tautological taxonomy for computer threats in past two decades where none was presented yet up to our knowledge. Example approaches can be found in [204, 136, 87].

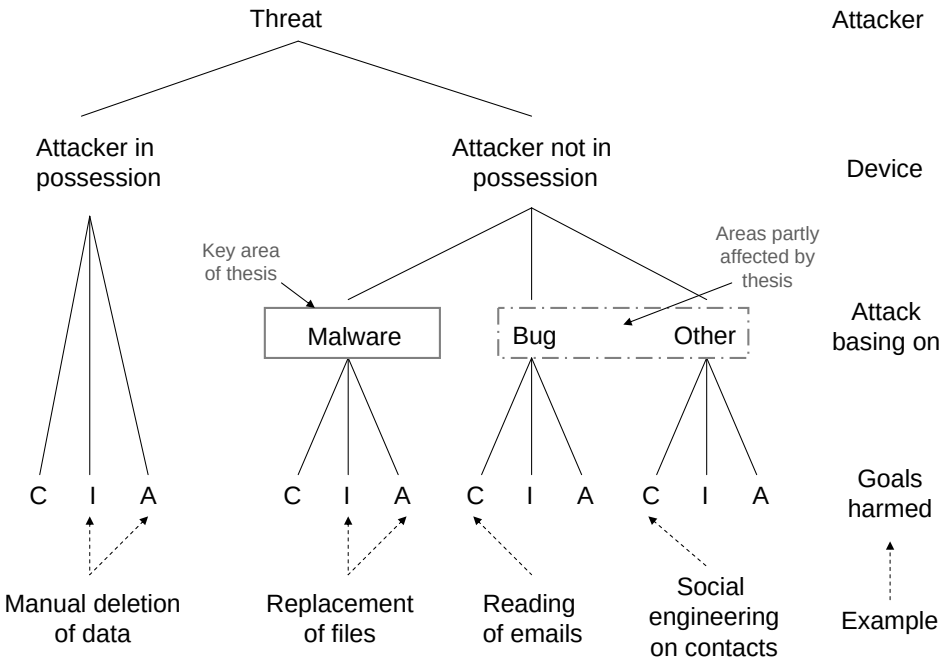


Figure 2.2: Major threats to smartphones are shown on this figure. The goals being potentially harmed refer to *(C)onfidentiality*, *(I)ntegrity*, and *(A)vailability*. Additionally, the main area of this thesis is highlighted while affected areas are also shown. In this figure, the term *bug* also refers to flaws that can be exploited.

Instead, a general overview on the field of smartphones threats, which can be seen on Figure 2.2, and corresponding attacks will be shown in the next sections. Figure 2.2 makes one major distinction: the attacker is in possession of the device or he is not. This distinction emphasizes the daily threat of mobile hand-held devices of potentially getting lost or stolen in comparison to stationary computers. This work mainly targets the case that the attacker is not in possession of the device. All approaches presented in

this work aim for detecting malicious applications running on a smartphone. Since one of the presented approaches bases on behavior-based detection other areas than malware detection are also touched.

Threats to Smartphones

In this section, well-known threats and attacks against smartphones will be presented. As stated in Def. 1, smartphones have a standardized operating system with available SDK and various interfaces. For being able to present threats to smartphones in a structured way, we will use Figure 2.3 that shows a simplified view on the most common smartphone interfaces which can be accessed through libraries included in the SDKs. Corresponding to these interfaces, relevant threats will be described.

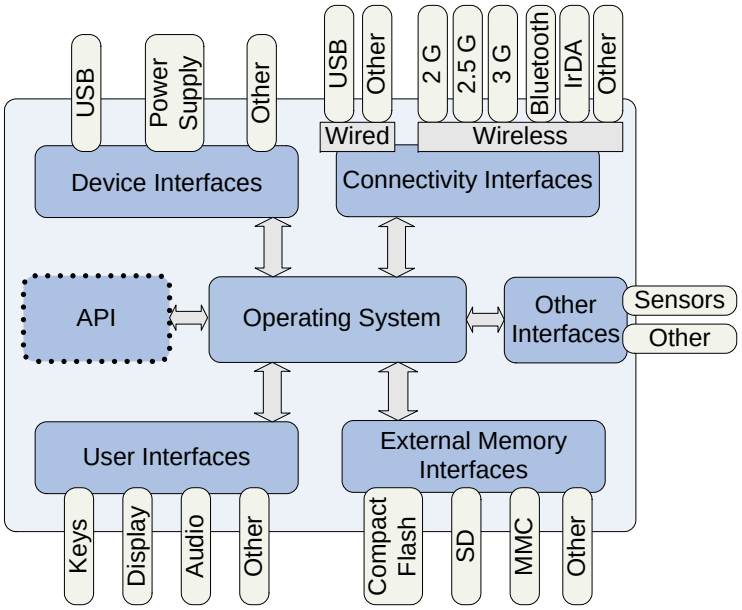


Figure 2.3: On this figure, a simplified view on the most common smartphone interfaces is presented. These interfaces will be used in order to structure the presented attacks.

The presented interfaces are sorted by their parent interfaces, which are *Device Interfaces*, *Connectivity Interfaces*, *User Interfaces*, and *External Memory Interfaces*. Attacks on other interfaces, like the ones from the sensors, are imaginable, e.g. an attacker might influence the magnetic field

in order to manipulate the compass or might cause signal interference for disturbing GPS, but will not be discussed in detail due to lacking publication of incidents. Each of the other interfaces faces various threats, where the threats presented next are separated as shown in Figure 2.2. At the end of this section, a summary on the threats is provided in Table 2.9.

Attacker is in Possession of the Device When having a smartphone in the hands of an attacker, e.g. device was stolen, kept unattended, or lost, another distinction has to be made in order to be able to describe the resulting threats: either the device operating system is accessible by the attacker implying an attack through the *User Interfaces* or it is not accessible. Access to the device operating system can be prevented by setting up mandatory user authentication, which is possible for most current smartphone operating systems[21], e.g. Android, iPhone, or Windows Mobile. In case of iPhone and Windows Mobile, a Personal Identification Number (PIN) or more complex pass-codes can be set.

PIN authentication is a well-known method on mobile phones to protect access to the SIM card where most users are probably unaware that in most cases the four digits PIN does not restrict access the device. This PIN is used to verify the International Mobile Subscriber Identity (IMSI) in the Home Location Registry of the corresponding network provider. If verification is successful, access is granted to the network. The problem about this method is that it has the purpose of controlling access to the network and not to the device or data. Replacing the SIM card normally leads to full access to the device.

Therefore, setting up proper authentication is very important for protecting against attacks through *User Interfaces* targeting the device and data. If the device has a set authentication method[22], attackers can try to use methods known from the field of computer forensics. The first approach might be to remove a memory card from the *External Memory Interfaces*, e.g. Secure Digital (SD) or Multimedia Card (MMC). If data was stored unencrypted, as assumable in most cases, all data can be read and modified possibly harming the security goals *confidentiality* and *integrity*. Since the device is in attacker's hand, *availability* is also harmed. Besides the threat of (private) information being read from a removed memory cards, these

[21] Authentication methods came up with newer smartphone OS, like Android. Before, most smartphones were lacking this feature.

[22] In this case it is assumed that this is not breakable by the attacker.

cards can also be used in order to distribute malware, e.g. the Windows Mobile malware called WinCE.Cxover.A [137]. Smartphone malware will be handled more detailed in Chapter 3.

Additional steps can be taken using tools, like SIMIS 2 [73] that targets Universal Subscriber Identification Module (USIM) cards[23] or Oxygen Forensic Suite [46] for investigating smartphones basing on all major operating systems. These tools are able to retrieve the IMSI, IMEI, contacts, messages, emails, attachments, web browser cache, and even deleted files without the need for special hardware; the software and a connector cable are sufficient in most cases implying an attack through the *Device Interfaces*. Since most of these tools are specialized on data extraction, the *integrity* goal is not harmed. Further information and recommendation on handling of cell phone forensics can be found in a special report of the National Institute of Standards and Technology written by Jansen et al. [108].

Besides the threat of forensic tools being connected to the device, the *Device Interfaces* obviously can be attacked by destroying the interface mechanically or by giving not specified inputs, like a too high voltage, to them.

Attacker is not in Possession of the Device When the attacker is not in possession of the device, he is limited to attacks on and through interfaces that are accessible to him. Normally, these will be the wired and wireless *Connectivity Interfaces* that include short and longer range wireless communication interfaces. These interfaces can be attacked directly[24] or they can be used for transmitting malicious data, e.g. malware or phishing emails.

As an example, Bluetooth was the source of many attacks in the past. Popular attacks were called *BlueSnarfing*, *BlueBugging*, *BlueJacking* and *BlueSmacking*. *BlueSnarfing* refers to the exploitation of immature Bluetooth implementation on some cellular- and smartphones. Using the Obex Push Profile (OPP) that normally provides an exchange mechanism for vCards[25] attackers were able to secretly download known files [225], like the corresponding file for the phone book[26] or calendar[27]. The *BlueBugging* attack bases on sending Hayes AT commands [48] to infiltrated phones.

[23]This tool also targets older Subscriber Identification Module (SIM) modules.
[24]In most cases the protocol itself or its implementation is attacked.
[25]These are electronic business cards.
[26]telecom/pb.vcf
[27]telecom/cal.vcf

By doing so, the attacker is able to perform various actions, e.g. initiation calls, sending SMS messages, or reading the dial history [223]. *BlueJacking* itself is not an attack; it just describes the usage of Bluetooth in order to transmit messages to devices in range. Messages might contain a hoax or offensive content and therefore still can be seen as threat to unknowing users. *BlueSmacking* refers to a denial of service attack similar to the classic "ping of death" attack [224]. In this attack, the targeted device will be sent packets with predefined length that are normally used in order to determine the round trip time (RTT). By doing so, the attacked device can be forced into a state solely trying to answer the sent packets ending in a Denial of Service. Besides attacks on and through the Bluetooth protocol, Bluetooth was also used for transmitting the first malware[28] for smartphones called Caribe [169].

But not only Bluetooth can be used in order to attack a smartphone, other communication protocols are also threatened. Engel [62] describes the "curse of silence" attack that can result in a device not being able to receive SMS/MMS messages any more until it is factory reseted. The attack bases on a SMS message sent as email through modified header containing a receiver email address longer than 32 characters. Several devices from running Symbian OS 2nd and 3rd version are affected by this attack.

Frick and Bott[29] [77] filed a patent in the year 2000 describing the so called IMSI Catcher which is a virtual base transceiver station (VBTS). Due to a feature of the GSM protocol, mobile nodes connect to the base transceiver station with the perceived highest transmission power. If a new node appears using a different area code to the former station, the mobile device tries to connect to this new node by sending its IMEI and IMSI. Another feature of the GSM protocol allows the BTS to decide on the encryption to be used for communication between the BTS and a corresponding node. In case of the VBTS, encryption will be switched off which allows attackers to capture all communication including phone calls.

If not using an IMSI catcher, attackers aiming for capturing communication have to try to break the encryption. A successful approach in finding secret key of the GSM A5/1 algorithm is presented by Biryukov et al. [30]. Approaches on breaking the algorithms A5/1, A5/2, and A5/3 are presented by Barkan et al. [20].

[28] The term malicious software (malware) will be introduced in the next section

[29] Both were employed by Rohde & Schwarz

Table 2.8: Results of the heap spray attack on mobile platform browsers

PLATFORM	OS	BROWSER	RESULT
iPhone	Mac OS X 2.2.1 (5h11)	Safari	System freezes. Fix requires restart of device.
iPod Touch	Mac OS X 2.2.1 (5H11)	Safari	System freezes. Fix requires restart of device.
HTC G1 "Dream" (retail)	Google Android RC33	Google Browser	Browser crashes. All applications killed. Application stack restarts.
HTC Android Dev Phone 1 (developer)	Google Android dream_devphone -userdebug 1.0 UNLOCKED	Google Browser	Browser crashes. All applications killed. Application stack restarts.
HTC Compact IV	Windows Mobile 6.1	Internet Explorer	Throws warning. Script can be stopped manually.

Besides attacks on the communication of smartphones, application weaknesses can also be used in order to attack a device. A still unpublished work of us bases on a memory vulnerability. The technique used to take advantage of the vulnerability is known as *heap spraying*, executed within the browser domain. To the knowledge of the authors, this technique was introduced by persons using the pseudonym *Blazde* and *SkyLined* for attacking Microsoft's Internet Explorer in 2004. Following [174], heap spraying targets vulnerable programs that jump into invalid memory spaces within the valid heap space. An essential requirement to perform *heap spraying* is the ability to control the application's heap. If an attacker managed to gain control, he will be able to insert large amounts of so called *No-Operation slides*, all ending with the final shell code to be executed, into the target heap. A common example application, being vulnerable to *heap spraying*, is the browser. In the past, Microsoft's Internet Explorer was the primary target for this type of attack, but other browser, as well, show similar vulnerabilities, e.g. Webkit-based browsers of smartphones and other mobile

devices[30]. In most cases, JavaScript is used for injecting the shell code into the heap where additional descriptions are given in [54, 185, 178, 207]. Table 2.8 depicts affected devices and OS versions in our tests using a modified web page to inject data into heap.

As stated formerly, one the biggest threats to smartphones is malicious software harming the security of the device and user. In most cases, smartphone malware bases on the *Application Programming Interface (API)* using existing functions in order to perform malicious actions. However, in some cases smartphone malware uses an application weakness, like the first iPhone worm called Ikee [84]. When having a certain SSH application installed on a "jail-broken"[31] device, the standard password is set "alpine" resulting in a serious vulnerability if not changed. The worm Ikee used this vulnerability in order to log into devices using the "alpine" password changing the background image to a photo of Rick Astley[32].

Summary on Smartphone Threats Table 2.9 shows a summary on the presented attacks with corresponding threats to smartphones. Comparing the numbers of check-marks in each row, malware, manual input by an attacker, and Bluebugging can have the worst outcome. Of course, not all threats have the same impact but the ones listed here include all possible threats. Therefore, they can be seen as the most severe ones.

Prevalent Security Mechanisms in Smartphones

For protecting a smartphone against attacks, several security mechanisms can be found built into the corresponding operating system.

For preventing unsolicited access through the *User Interfaces*, authentication methods can be used to control access to a device. Despite the probable assumption of many users, the prevalent "one-shot" authentication basing on a four digit PIN on start-up of the device does not restrict access to the device; it is used to authenticate the user to the mobile phone network. Therefore, most mobile phones do not actively use or even support user authentication meaning if the device is lost or stolen attackers get full access to it. Vendors of current smartphone platforms realized this problem

[30] Android, iPhone, and iPod affected.
[31] A modified firmware is installed giving the user more control over his device.
[32] Rick Astley is a music artist.

Table 2.9: Smartphone interfaces and their threats

	Masquerading	Eavesdropping	Modification	Repudiation	Forgery	Sabotage
Device Interfaces						
Manual actions by attacker		✓				✓
Forensic tools		✓				
Connectivity Interfaces						
BlueSnarfing		✓				
BlueBugging	✓	✓	✓	✓	✓	✓
BlueJacking						
BlueSmacking				✓		✓
Curse of Silence				✓		✓
IMSI Catcher		✓		✓		✓
User Interfaces						
Manual actions by attacker	✓	✓	✓	✓	✓	✓
External Memory Interfaces						
Manual actions by attacker	✓	✓			✓	✓
Application Programming Interface						
Heap spraying			✓			✓
Malware	✓	✓	✓	✓	✓	✓

and included authentication functionality to their devices. As an example, Android includes the possibility to set a visual pattern for authenticating to the system. This pattern has to be drawn on a grid consisting of nine points which can be seen on Figure 2.4.

In case of iPhone and Windows Mobile, a four digit PIN can be set[33]. Interestingly, all mass market devices we saw did not include any other authentication method that was built-in and shipped by the vendor. Anyhow, by installing additional application it is possible to add authentication methods to a device. The problem is that in most cases these methods

[33]More complex pass-code including regular characters can also be set.

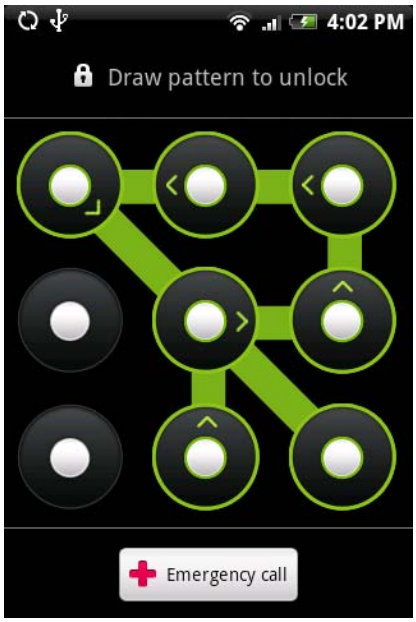


Figure 2.4: Android authentication using visual pattern

only protect a limited space[34] [145] and not the device itself. Nevertheless, such application together with encryption methods can be very useful when protecting memory cards in the *External Memory Interfaces*. If these cards are not protected, full access to them is very probable due to their FAT file system[35] which does not support proper access control.

Protection of the *Device Interfaces* is up to the end user since these are only threatened when an attacker has physical access to them. The opposite can be found when considering the *Connectivity Interfaces*. In most cases, the user will not have control over protecting the communication of the device. Protection of the communication is part of the communication protocol where the former examples [224, 223, 225, 77, 30, 20] show that the possibility of successful attacks should not be ignored.

When looking at security mechanisms protecting against malware and misuse of the *Application Programming Interface (API)*, several approaches can be found built into current smartphone operating systems. As shown earlier, Symbian uses a certification system in order to prevent third-party application to access restricted function calls and file system areas. For

[34]This can be a certain application or container file.
[35]Fat file system is used for interoperability reasons.

deleting system files, for example, you will need a certificate that only Symbian or phone manufacturers will get. Android uses UNIX-like user IDs to assign specific permission to applications. Additional Java-level permission can be set to restrict access to certain packages that allow usage of sensitive or critical functions, e.g. calling capabilities. A comprehensive guide to Android security mechanism can be found in [61]. Similar to Android, the Apple iPhone uses a sandbox system for its application to prevent application to access other application data. Additionally, system files, resources, and the kernel are shielded from user application space. Like Symbian, Android, and Windows Mobile applications, Apple iPhone applications need to be signed when released. The corresponding certificate that is issued by Apple can prevent unsolicited alteration of the application while making the origin[36] traceable. Windows Mobile secures its devices through security roles, security policies, and application signing.

But whenever these protection methods are bypassed by malware, e.g. through an exploit in a library, no comprehensive measures are available to secure smartphones. Most available anti-virus products for smartphones still base on signature-based approaches[37] leaving devices infected by new or unknown malware exposed to the malicious functionalities. This situation can be seen as the main motivation of this thesis for finding light-weight approaches capable of detecting new and unknown malware on smartphones.

2.5 Related Research

In this section, related work in the fields *smartphones*, *smartphone security*, and *user awareness* is presented.

2.5.1 Smartphones

As stated in Section 1.1, the smartphone follows the early vision of Mark Weiser [237] but does not reach it completely: "Ubiquitous computing enhances computer use by making many computers available throughout the physical environment, but making them effectively invisible to the user".

[36] In this case the developer is meant.

[37] Although not proven, it is very likely that more comprehensive methods were not deployed yet due to limited processing capabilities.

Especially the invisibility is not achieved yet due to the requirement of a screen as visual output device which is currently bound to the device itself. Nevertheless, it is imaginable that current screens might be replaced in near future letting Weiser's vision become truth for smartphones.

Abowd [2] underlines the meaning of smartphones for the world of ubiquitous computing. He states that for the time of publishing his article in 2005, the killer applications of ubiquitous computing were person-to-person voice communication and text messaging (SMS).

Charlesworth discusses the ascent of smartphones in his article [39] while he also predicts their descent as soon as they get broken down into separate components being integrated into clothing which again follows the vision of Weiser.

Reynolds [182] talks about the implications and consequences of the increasing amount of smartphones connecting to the Internet while considering data security issues when such devices are stolen. He also discusses the lacking revolution of human-computer interfaces on smartphones compared to the corresponding revolution of computing itself.

Guo *et al.* [85] describes possible attacks for smart-phones, which range from privacy violation and identity theft to emergency call center DDoS attacks and national crises, concerning their interoperability between the telecommunication networks and the Internet. They also describe defense solution space including smart-phone hardening approaches, Internet-side defense, telecommunication-side defense, and coordination mechanisms that may be needed between the Internet and telecommunication networks.

Several fields of application were presented and proposed by other authors: Ravi et al. [179] describe a protocol for the usage of smartphones for accessing ubiquitous services, e.g. door opener service. Beale [24] shows example applications supporting social interaction with smartphones, e.g. a local dating service and file sharing. Cheok [42] uses smartphones for realizing a modern real-life version of a capture-the-flag game. In this game, two teams try to capture the flag of the other team while trying to protect its own. The Flag was realized a small and hand-held box running Linux while allowing Bluetooth connections. Yu et al. [242] describe the usage of smartphones for supporting context-aware media recommendations. Their approach uses local characteristics and preferences of the media player as well as the locations, activity, and time in order to recommend media, like images, video, or text. Raento et al. [177] also work in the field of context-

based computing where they provide a platform for context-aware mobile applications.

Furthermore, smartphones find application in the field of medical helpers as described by Leijdekkers [131]. In his work a personal heart monitoring system using smartphones is described being capable of detecting life threatening states. Additionally, interesting areas of application were pointed out by Roussos et al. [186]:

- Mobile Phones as information service endpoint, e.g. applied as navigational assistance or location based services.
- Mobile Phones as remote controllers for different devices, like television or Hi-Fi station.
- Mobile Phones as pervasive network hubs to provide wide area connectivity, e.g. for wearable systems that need to communicate in order to transmit health-related data.
- Mobile Phones as ID tokens in order to store information used to verify the user and information.

Using smartphones as input devices opened a new research field in Human Computer Interaction (HCI). Ballagas et al. [19] describe smartphones as ubiquitous input devices. Furthermore, they state that the smartphone might even become the default psychical interface for ubiquitous computing application while providing the basis for new interaction paradigms. Anquetil et al. [15] show the possibility of integrating online handwriting recognition on a smartphone[38].

As a final reference, Amft et al. [11] is given. Amft et al. describe the evolution "from backpacks to smartphones". In this work, the past, present, and future of wearable computers is shown where the smartphone is told to be the future central on-body platform for general purpose computing tasks. Several other peripheral components, like wearable computing devices and sensors, will be able to connect to the central and invisible smartphone in order to exchange information or computational tasks. In the opinion of the author of this thesis, this outlook seems realistic while fulfilling the vision of Weiser completely.

[38]This approach already succeeded in the year 2002.

2.5.2 Related Research in the Field of Smartphone Security

In Section 2.4.2, several threats and attacks on smartphones have been presented. In this section, we will introduce related research papers investigating the possibility of new attacks as well as the feasibility of new approaches to protect the devices.

Racic et al. [176] demonstrate an attack, which can drain the mobile devices battery power up to 22 times faster. This attack targets a unique resource bottleneck in mobile devices (the battery power) by exploiting an insecure cellular data service (MMS) and the insecure interaction between cellular data networks and the Internet. The attack consists of two stages. In the first stage, the attacker generates an victim list by exploiting the MMS notification system. In the second stage, UDP packets are sent to the victims in order to drain the battery.

Jesse D'Aguanno[39] gives detailed information on how to attack RIM Blackberry supporting networks[40].

The continuous work of Mulliner et al. 2006 [153, 156, 152, 155] has led to essential work concerning Windows Mobile and Symbian OS security.

Enck et al. [60] present a rule-based system in order to indicate malicious potential of Android applications. Therefore, they collected the top 311 applications from android market and checked them for occurrences of certain permission set in a configuration file of each. This check showed that five of these applications implemented dangerous functionalities. Another five also showed dangerous permissions but these could be argued through provided functionality of those applications.

Ongtang et al. [166] propose a policy enforcer for Android that hooks into the system in order to control application at installation and at runtime.

In [221], Traynor et al. present how to exploit open functionalities in SMS-capable cellular networks. Using a single modem, an attacker is able to deny voice service in scale of major cities, like Washington DC. Using a bot net can result in whole countries being attacked. In [222, 220]

[39] He presented as speaker with the pseudonym "x30n".

[40] proof-of-concept at `http://www.praetoriang.net/presentations/blackjack.html`

the same authors propose mitigation strategies for this attack using queue management and resource provisioning.

Hwang et al. [92] state that embedded devices, like cellular phones, smart cards or embedded network sensors are mostly portable, communicate wireless and are battery powered or at least energy-limited. The design of security for embedded systems differs from traditional security design, as different characteristics can be found for each kind. There are two main groups of characteristics that differentiate the security architecture from Embedded System from that of workstations and servers: resource limitations and physical accessibility. In their work, Hwang et al. claim that embedded security cannot be solved at single security abstraction layer and therefore present security measures for all abstraction layers.

2.5.3 The Role of the User in Security

In the Ph.D. thesis of Michael Becher[41] from University of Mannheim readers get pointed to the direction of security awareness and the influence of the user on the system security. Several interesting articles were published that are not directly related to smartphone security or malware detection but still can open a different view on computer and smartphone system security.

Bruce Schneier wrote several books on computer security while claiming at the beginning that cryptography can solve security problems. In the book [201], he revokes his statement saying that "weak points have nothing to do with mathematics. They were in the hardware, the software, the networks, and the people. Beautiful pieces of mathematics were made irrelevant through bad programming, a lousy operating system, or someone's bad password choice.".

Furnell et al. [78] present results of a survey of over 340 participants that aimed for determining their understanding of security features within Windows XP and three popular applications. The major conclusion is that there is a need for increased usability since a large amount of respondents had problems with even standard security features.

A similar result is presented by Whitten et al. [239]. In their work they describe a test where the participants were given 90 minutes in which to

[41] His thesis should get published soon.

sign and encrypt a message using PGP 5.0. Since the majority was not able to do so successfully, they conclude that PGP 5.0 was not usable enough although it had rich user interface.

Tössy et al. [219] recommend to train the smartphone user not to install application received via Bluetooth or MMS while Görling [81] states that several security researchers claim that educating the user is the best approach to computer security. In his work, Görling questions these statements. He additionally says that "the user will circumvent a security model where the security features clash with the tasks the user is trying to carry out". Therefore, security must not be added, it must be integrated in an earl stage.

2.6 Summary and Conclusion

In this chapter we started with showing the evolution of relevant technologies in Section B that most likely led to today's smartphones. Several inventions, e.g. wireless transmission and mobile computing, played an important role where smartphone contain and use a various amount of technologies for providing services to users.

After discussing different view points on the characteristics of smartphones in Section 2.1, we gave our own definition in Def. 1 for having a common understanding when using this term throughout this work. Essential characteristics of smartphones include supporting native application development as well as numerous communication interfaces on a hand-held sized device.

In Section 2.2, we presented hard- and software characteristics of current smartphones while using the Google Nexus One as example. The Nexus One runs at 1 GHz and has 512 MB Ram installed which offers users a wide range of application possibilities. Additionally, we highlighted the difference between smartphones and *classic* computers which is basically the mobile nature and the compact size of a smartphone.

For showing the evolution of smartphone usage, we compared published surveys on smartphone usage with new ones we conducted in May 2010 in Section 2.3. On interesting but obvious change was the increase of Internet usage on devices. But in general, smartphone usage did not change that much over time: a smartphone still mainly serves as voice-centric commu-

nication device. Despite this observation, we believe that the smartphone will follow the vision of Mark Weiser.

Chapter 3

Malicious Software for Smartphones

Smartphones get increasingly popular which also attracted malware writers beginning from June 2004. From this point on, malware count increased steadily while main target remained Symbian OS[1]. After the introduction of application signing, the amount of emerging malware decreased while only few news could be read on this topic for a long time. After researcher called Collin Mulliner presented a way to break Symbian OS security, this changed and new malware emerged.

In this chapter, we present the evolution of smartphone malwares until the end of 2010. We describe key aspects of such malwares and list affected platforms. Additionally, general malware countermeasures and detection are presented. These are not specifically designed for smartphone platforms but applicability to this domain is researched and for some methods already proven.

The chapter is structured as follows. In Section 3.1, we introduce information on malware basics. In Section 3.2, related research in the field of smartphone malware is presented. Smartphone malware evolution is revisited in Section 3.3. In Section 3.4, general countermeasures and detection approaches are presented that are currently used in order to handle malware. A summary on this chapter is given in Section 3.5.

[1] `http://www.symbian.org/index.php`

3.1 Introduction to Malware Basics

In this section, we discuss the danger of malware for smartphones. Therefore, we give a brief introduction to *common* malware principles.

Malware is a portmanteau of the two words *mal*icious and soft*ware*, which clearly indicates that malware is a computer program with malicious intentions. In order to understand what these malicious intentions actually are, we introduce the terms: *infection vector* and *infection payload*. The *infection vector* describes which techniques are used to distribute the malicious application. Several known approaches are: e.g. file injection, file transport, exploit[2], or boot sector corruption. The *infection payload* represents the actual content that is used to harm the victims' machine. Several known possibilities for payloads are[3]: deleting files, denying service, or logging keystrokes.

Table 3.1: Characteristics of viruses, worms, and Trojan horses.

	Virus	**Worm**	**Trojan Horse**
Appearance	needs a hosting medium	independent program	malicious functionalities disguised
User Interaction	usually needed	usually not needed	usually needed
Vector	such as file injection or boot sector	such as exploit	such as email atachment or download
Payload	such as system modification	such as malware drop	such as backdoors

There are three common categories of malicious software: *virus*, *worm*, and *Trojan horses*. A *virus* mostly comes in a hosting medium that can be, e.g. an executable file or a floppy disk. If the user executes this file, the virus processes its' malicious commands which can be almost everything the OS allows. A *worm* can often spread without user interaction. Once started, it searches for infectable victims in range. If a victim is found, it normally uses an exploit to attach itself to the victim and then repeats this behavior.

[2]uses some kind of hardware, software, service, or protocol weakness

[3]We suggest [211] for further readings.

Sometimes worms drop other malware that can be back-doors that allow remote access. *Bot* programs installed this way can make the victim to a remote triggered Denial of Service (DoS) attacker. A *Trojan horse* is a program that is disguised, e.g. as a popular application, in order to pursue a user to execute or install it. This is done by choosing a well-known name from a popular game and placing the malware for download on a web page or file sharing tool. In Table 4.6, additional popular application categories are listed that help to convince the victim to execute the malware.

Although it is not possible to categorize every malware clearly, most of them can be usually assigned to one of the mentioned categories in [211]. On Table 3.1, a short overview on the malware characteristics is given.

Malware can be propagated using several techniques and communication interfaces, ranging from an exploit to using social engineering. Regarding smartphones, the most used infection mediums are Bluetooth, Internet, MMS, Memory Card, and USB as illustrated on Figure 3.1.

Figure 3.1: Smartphone malware propagation.

3.2 Related Work

Smartphone security and malware detection is a rather new field of research where topics of publications are scattered within this domain. Starting with the first wave of Symbian OS malwares, several authors pointed to the "new"

threat targeting smartphones, e.g. Dagon et al. [53], Jamaluddin et al. [107], Piercy [169], Niemelä [159], Leavitt [126], and Hypponen [93].

Overviews on smartphone malware appearance were given by Töyssy et al. [219], Gostev [82, 83], Fleizach et al. [72], Lawton [124], Schmidt et al. [190], and Shih et al. [205] while most of them end in 2005 or 2006. In this work, we update these overviews by extending the list of appearances to the end of 2008 while practically adding a new entry for the beginning of 2009.

Android and iPhone malware propagation is still a basically not investigated field of research. Since both platforms mainly use an online store for distributing software, new infection vectors have to be found for predicting or simulating malware propagation. Valuable input is given by Mickens et al. [143], Bulygin [35], and Wang et al. [234], who give interesting insights into propagation models and estimations.

The possibility of attacking smartphones was investigated by several researchers. Among them, the continuous work of Mulliner et al. 2006 [153, 156, 152, 155] has to be noted since essential work concerning Windows Mobile and Symbian OS was presented. Review of techniques was also published by Racic et al. [176] who used MMS in order to deplete the battery of mobile phones. Becher et al. [25] presented a promising approach for creating a worm for Windows Mobile. Unfortunately, they were lacking the appropriate exploit for making a fully working malware. Jesse D'Aguanno[4] gives detailed information on how to attack RIM Blackberry supporting networks[5].

Fleizach et al. [72] introduce a simulator for mobile malware propagation. Many assumptions are made and mobility in the sense of users changing cells is not covered. Nevertheless, valuable information on possible malware propagation speed and on phone-book entry distribution is provided.

Jamaluddin et al. [107] evaluate the possibility that mobile phones, like smartphones, will have to face the same sort of malware that PCs already have to defend against. Therefore, they introduce existing and future threats regarding these devices, and show the ease with which certain types of malwares can be implemented.

[4]presented as speaker with the pseudonym "x30n"

[5]`http://www.praetoriang.net/presentations/blackjack.html`

Martin et al. [156] describe three main methods to perform attacks preventing CPUs of mobile devices to go into sleep mode. For countering such attacks, they propose a power-secure architecture using authentication and energy signatures.

3.3 Smartphone Malware Evolution

Initial work on showing smartphone malware evolution was published by Gostev [82] from Kaspersky Lab. Key results for the time span from June 2004 until August 2006 were that smartphone got an increasingly popular target for attackers where mainly Symbian OS malwares appeared. Our work extends former descriptions where we were able to compare public and internal data on malware appearance. Interestingly, we noticed a great discrepancy between published malware appearance and corresponding available descriptions on their behaviors. Especially in the last two years, descriptions on the behaviors got scarce without obvious reason.

3.3.1 Smartphone Malware from 2004 to 2008

For statistical purpose, we gathered all published malware *descriptions*[6] from various web pages (e.g. from F-Secure, Kaspersky, McAfee, Symantec, Sophos, and similar) for identifying key aspects of mobile malwares. One obvious aspect is their appearance in time. Figure 3.2 shows mobile malware evolution from January 2004 to December 2008 based on published mobile malwares with available behavior description.

We found 288 smartphone malwares until the end of 2008 where peeks in new appearing malware can be found at the end of 2005 and in the middle of 2006. It is imaginable that these peeks were caused by the introduction of a certificate-based signing system for Symbian OS applications[7]. Malware writers might have feared a decreasing number of possible victims motivating them to increase their efforts on malware creation. In the signing process, a trusted Symbian partner checks the complete source code and binaries for meeting certain criteria, such as being free from memory leaks and abusive methods. If the check is successful, the application gets signed

[6] malwares lacking descriptions were ignored

[7] `http://www.symbiansigned.com`

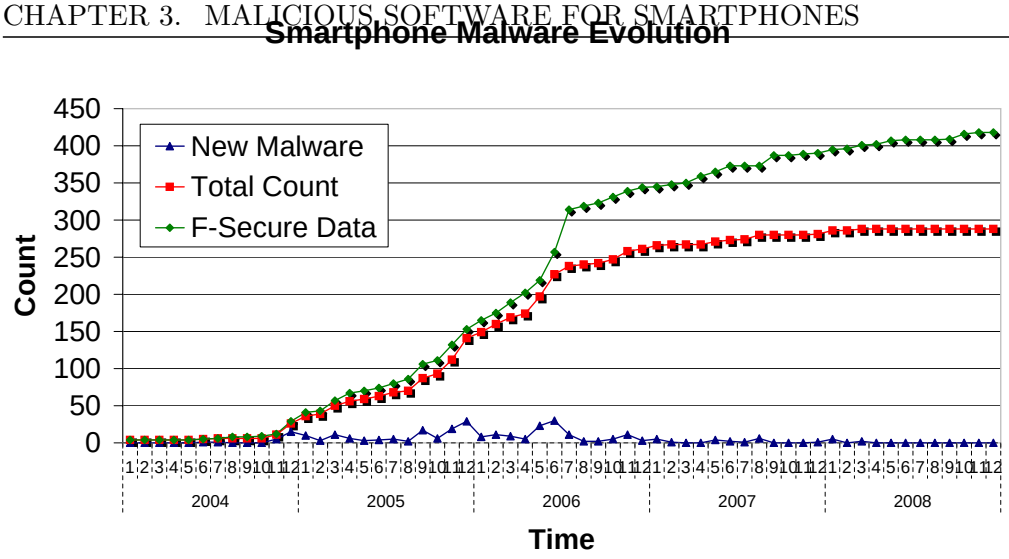


Figure 3.2: Mobile malware evolution basing on published malware including descriptions on their behavior. F-Secure data was added for comparison showing a big discrepancy between internal anti-virus vendor and published data.

with a certificate and stays clearly identifiable through a given unique ID. Additionally, signing restricts access to sensitive function calls from certain APIs, e.g. network control, preventing abusive usage. Application signing gets mandatory for the current Symbian S60 *3rd* Version which is installed on most Nokia smartphones from the end of 2005.

For comparison, we requested the corresponding numbers from F-Secure Research in Helsinki[8]. Comparing the numbers from Figure 3.2, you can see that F-Secure counted 418 malwares, 130 more than we found, showing that there are several malwares without publicly available descriptions. Additionally, following the F-Secure numbers, in the middle of 2006 more than 100 new malwares appeared[9].

Based on published malware descriptions, we listed the malware effects[10] which can be seen on Figure 3.3. Please note that the categories are not disjunctive, therefore the count of malware having certain effects exceeds our total count of 288 malwares. Several different malicious behaviors were recognized while more than half of the malwares manipulated files disabling application or device. Another interesting point is that 50 malwares

[8]We want to kindly thank Jarno Niemela for providing us these information

[9]Of course, it is also possible that F-Secure updated its database at that time.

[10]Malware effects are commonly known as infection payload.

did not have a malicious behavior except their propagating functionality.

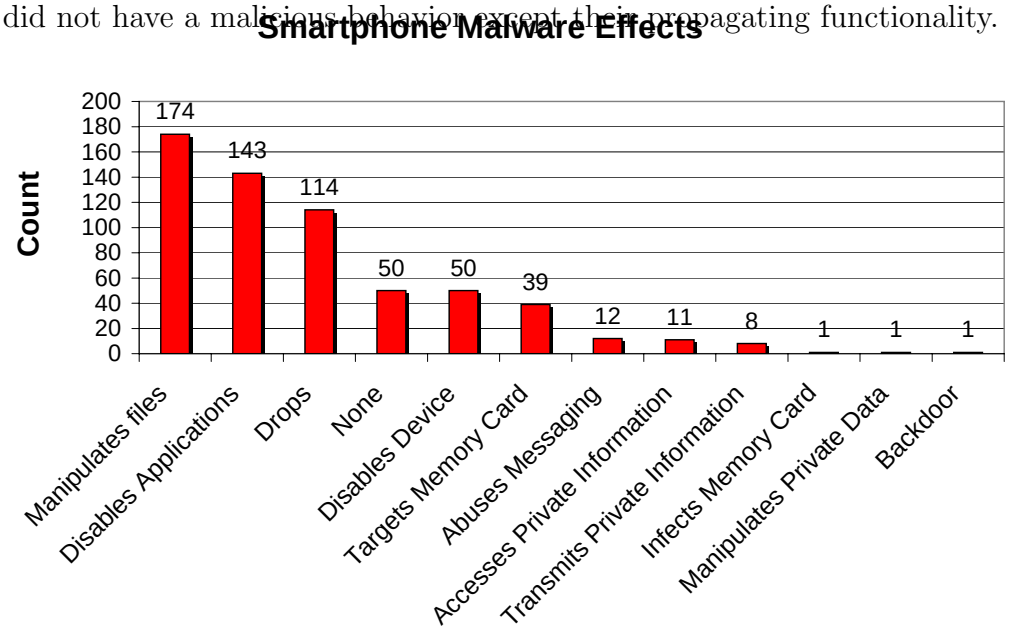


Figure 3.3: Smartphone malware impact is depicted on this figure. Manipulating files, including replacing and deletion, was the most common payload of smartphone malware until 2008.

Smartphone malware uses various channels for infecting new devices. Additionally, several other technologies, like Bluetooth, MMS or memory card, were used for propagating these malwares as illustrated in Figure 3.4 and Figure 3.1. What most malwares, especially for Symbian OS, have in common is that they require an installation file for propagation.

All malwares basing on (Symbian OS) installation files explicitly need user interaction for installing on the system. Therefore, most smartphone malwares are categorized as "Trojan horses" (84%), c.f. Figure 3.5. Even worms (15%) need user interactions in order to get installed. Hence, propagation schemes cannot be compared with Windows worms using system vulnerabilities.

Interestingly, most of the malwares target Symbian OS (283 malwares) where only 4 Windows Mobile and 2 Java ME malwares were recognized. The payload of the Windows Mobile and Java ME malwares included remote access, file deletion, and abuse of the SMS in order to charge high service usage rates.

Coming back to Figure 3.2, malware appearance decreased starting

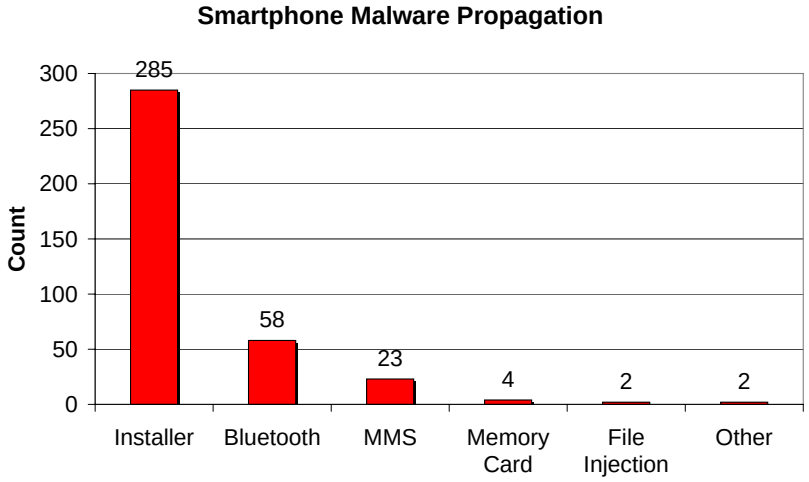
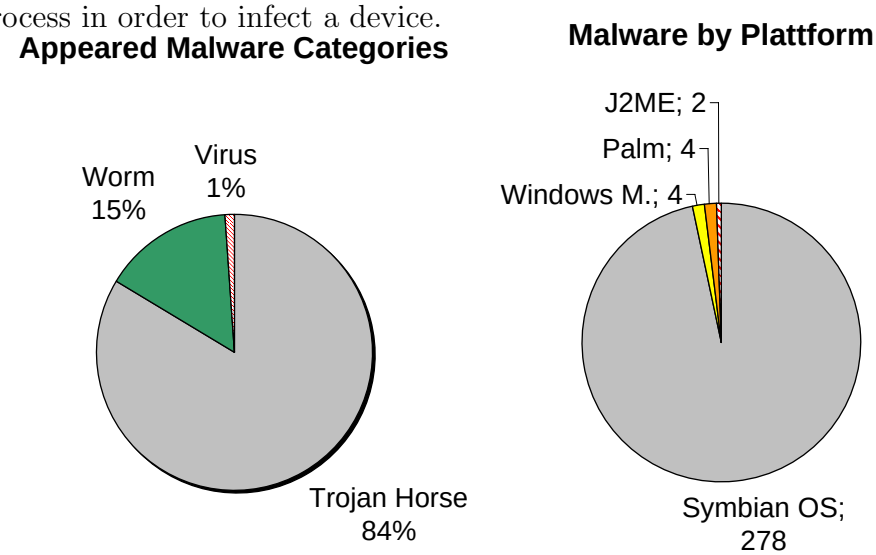


Figure 3.4: This Chart shows smartphone malware propagation channels. Almost all smartphone malwares required manual start of the installation process in order to infect a device.



(A) (B)

Figure 3.5: A: 84% of smartphone malwares are Trojan horses; B: 79% of all smartphone malwares target Symbian-based phones.

from the middle of the year 2006. Until end of 2008, only about 100 new malwares appeared while between the same time span from end of 2004 to the middle of 2006 about 300 emerged. The reason for this can be seen in the certification system of Symbian OS 3rd where devices running this operating system version gained more and more market share at this time.

This OS version is not vulnerable to the former malwares. Only news's that one spyware got certified for Symbian OS 3rd, called FlexiSpy [210], this recalled the existing threat to this new age of smartphone OS.

In addition to the malwares "in the wild" several research activities aimed for bypassing the security systems of smartphones. One of the latest works of Collin Mulliner resulted in the ability to bypass the security mechanisms of Symbian OS 3rd which he presented on Black Hat Conference 2008 in Japan [154]. Shortly after, in February 2009, the first malware targeting Symbian OS 3rd appeared using a valid certificate[11].

3.3.2 Smartphone Malware from 2009 to 2010

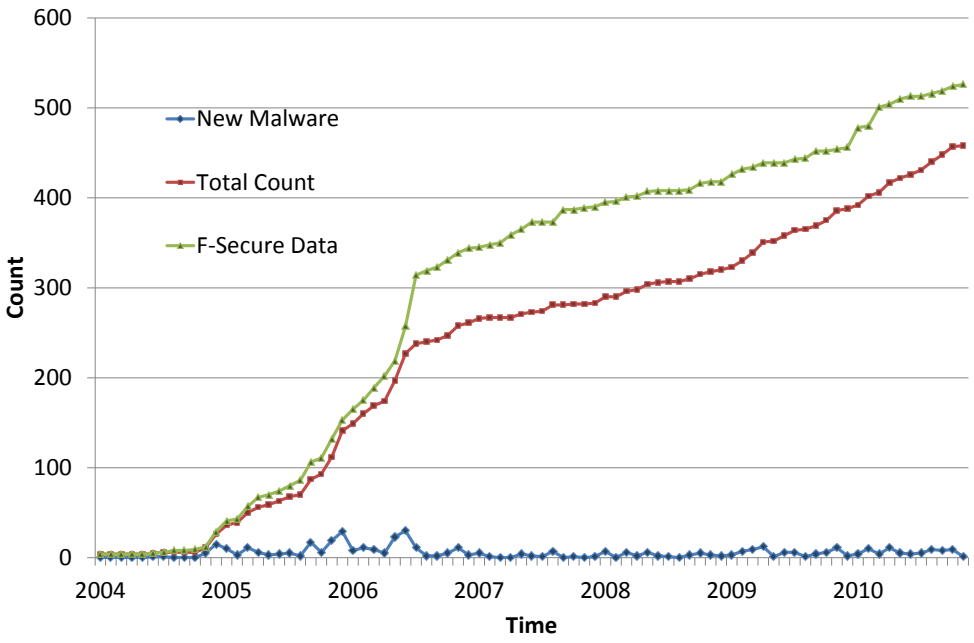


Figure 3.6: This chart shows updated graphs on smartphone malware appearance by November 2010. Again, F-Secure data is added for comparison.

Figure 3.6 illustrates the malware growth by November 2010. Again, our data is compared with updated F-Secure data[12] where following observations and experiences were made. One was that other anti-virus vendor

[11] `http://www.f-secure.com/weblog/archives/00001609.html`
[12] Again, we want to thank Jarno Niemela from F-Secure for his help.

lists seemed to get updated more steadily. In January and March 2010, F-Secure added a lot of malware to their databases which the others had added earlier. Additionally, listed smartphone malwares on most anti-virus vendor pages lacked a detailed description on the payloads. Therefore, no explicit numbers can be presented on the potential impact of most of the malwares for the years 2009 and 2010.

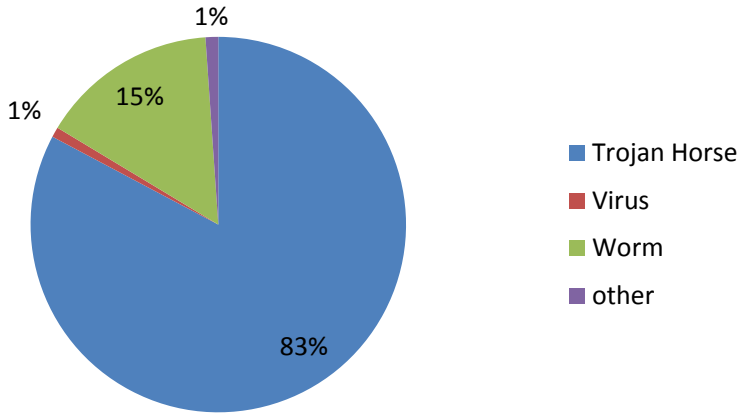


Figure 3.7: Emerged malware categories by 2010

In comparison to the earlier phases of smartphone malware, the categories almost did not change which is shown on Figure 3.7.

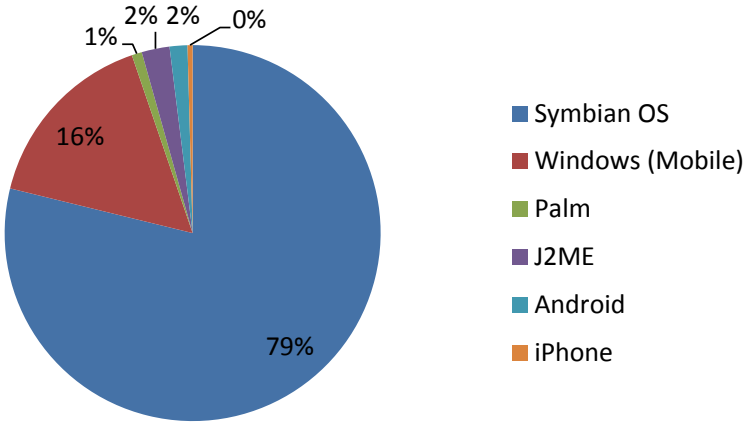


Figure 3.8: Malware per platform by 2010

What did change was the amount of malwares for the different plat-

forms. Windows-based smartphones cover 16% of all malwares. Android and iPhone faced malwares; only one appeared for each platform but several variants were generated. The iPhone malware affects "jailbroken" devices having a SSH-client installed. Android got target of a malware named Fakeplayer which pretends being a media player but instead sends SMS messages to premium services.

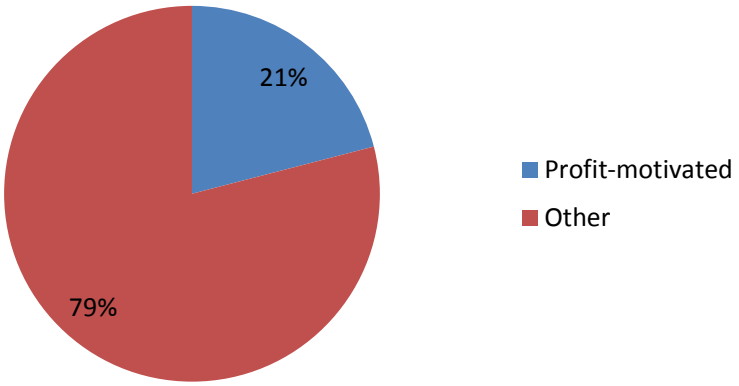


Figure 3.9: Share of profit-oriented smartphone malware

Interestingly, abusing SMS for gaining profit got a major motivation for writing smartphone malware as illustrated in Figure 3.9. Especially Russia was center for such kind of attacks. Most of these malwares send text messages to Russian premium services causing high cost at victim side. The increase of this abusive malwares is depicted in Figure 3.10. By the time of writing this thesis, only preliminary numbers until the beginning of November were available. But in general, an increasing interest can be observed and since this is the only monetary model that seems to work, a growing number of such malwares has to be expected.

3.4 Malware Detection Approaches and Countermeasures

Countermeasures, which help to secure a system, can be usually taken by installing certain hard- or software. Three main systems for computers can be identified: firewalls, anti virus software and intrusion detection systems

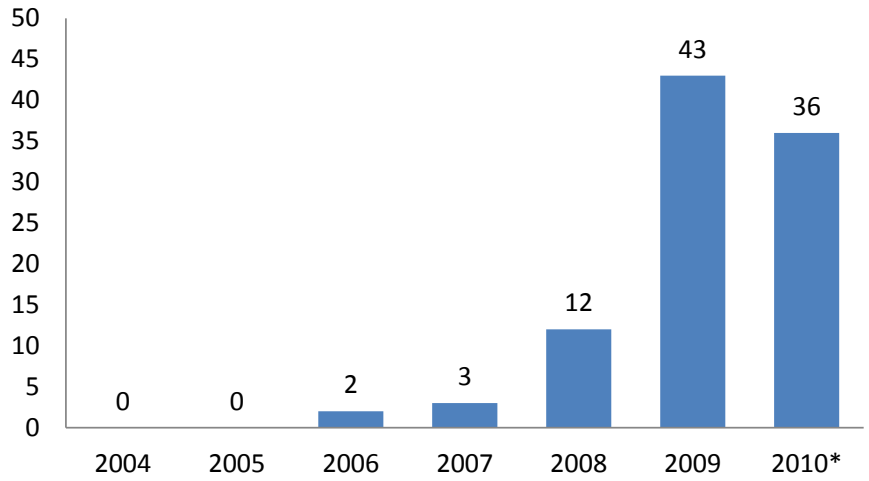


Figure 3.10: This graph shows the amount of malwares abusing premium messaging services by the end of October 2010.

(IDS). Readers familiar with IT security tools, intrusion detection, and malware detection principles can jump over to Chapter 4.

Firewalls [135] are so called "white list"-based systems, which means that there is a special list of rules explicitly allowing certain ports to communicate with internal or external peers[13]. If malicious software is able to masquerade as trusted software using a trusted port, a basic firewall will allow all communication activities. Anti virus scanners use "black lists" in order to detect certain threats included in the black list.

A virus scanner [212] can block viruses, worms, and Trojan horses with real time monitoring or manual scanning. Malware is detected by scanning for and finding a certain string or pattern, also called signature. Therefore, the malware has to be known by the scanner. Virus scanners normally include a specific disinfection routines corresponding to the detected signatures.

Intrusion Detection Systems (IDS) [112] formerly were systems that monitored network traffic. Logged traffic was used by network administrators in order to detect abnormal behavior. Countermeasures like closing ports or locking systems could be taken by the administrators. IDS evolved into intrusion prevention systems (IPS) which are able to detect certain

[13]e.g. TCP/UDP

abnormal behaviors and take preventive measures automatically. Basing on abnormal behaviors, intrusion detection and prevention systems (IDPS) are basically able to detect malware activity while they lack the removal routines known from virus scanners.

Virus scanners and Intrusion Detection Systems present the basis of our approaches. Architectures that are presented in this work will use similar principles as shown in the following sections, where virus scanners and IDS will be described in Section 3.4.1 and Section 3.4.2, respectively. While firewalls focus on restricting network traffic, virus scanners and Intrusion Detection Systems try to detect malicious software and activities using *static* and *dynamic analysis*. The major difference between *static* and *dynamic analysis* is that *dynamic analysis* refers to data acquired on runtime while *static analysis* does not. *Static analysis* can solely rely on data extracted from binaries in a static manner. Methods being applied to the acquired data for detecting malware can basically be the same to for both variants. Corresponding explanations and definitions related to detection results are given in Section 3.4.4.

3.4.1 Virus Scanners

In this section, we describe widely deployed virus scanners and corresponding approaches to detect and counter malware. Following Szor [212], several approaches and optimizations are used where an excerpt of these is displayed next.

String scanning: This is the simplest approach to detecting computer malware. It uses a sequence of bytes that should only exist in malware and not in benign programs. An example given by Szor [212] is the byte sequence:

```
0400 B801 020E 07BB 0002 33C9 8BD1 419C
```

This sequence represents hexadecimal application code which basically represents assembler instructions in turn. Additionally, this sequence also represents a signature that can be shared in order to detect malicious code which might be found in several different files. A problem about this static sequence is that a simple change in the order of the instructions leads to a mismatch resulting in a missed detection. This

simple reordering can be seen as the first approaches of malware writers to *obfuscate* their malicious code in order to mitigate detection. In order to tackle problems with static sequences, *wildcards* and *mismatch scanning* were introduced. *Wildcards* allowed skipping bytes or byte ranges while *mismatch scanning* defined a fixed number of hexadecimal digits that could take any value. The latter approach was especially useful in generating more generic signatures that were able to detect whole families of malware.

Further optimizations were applied in order to increase detection speed. *Hashing* is a common way to speed up search algorithms. Since classic virus detection is a search operation, detection speed can greatly benefit from hashing. *Top-and-Tail* and *Entry-point scanning* are further methods to increase detection speed. In *Top-and-Tail* scanning limits the search area to the top or tail of a file. This was useful since the majority of early malware prefixed, appended, or replaced host files. An interesting but unfortunately outdated approach is called *hyperfast disk access*. This approach bypasses the operating system-level APIs in order to access hard disks directly through the BIOS. Using these methods, a ten-times-faster file I/O could be achieved.

Smart scanning: This approach was used in order to prevent malwares to mitigate detection through inserting no-operation (NOP) slides into their code. Anti-virus scanners using *smart scanning* skipped NOP slides and did not use them for their signatures.

Skeleton detection: This approach was invented by Eugene Kaspersky. *Skeleton detection* refers to a line-by-line checking of macros in order to identify the skeleton of essential and important macro statements. Unimportant statements and white spaces are dropped resulting in a better detection even for variants.

Algorithmic scanning: Following Szor [212], the term *algorithmic scanning* is a bit misleading since it basically describes the manual creation of a virus signature whenever a generic approach fails. Early implementation consisted of hard-coded functions integrated into the anti-virus search engines. Due to frequent updates of the virus scanners, these hard-coded detection routines often led to problems with the stability. A solution to this was the introduction of virus scanning languages which allow seek and read operations in their simplest form.

Since *algorithmic scanning* is more expensive in terms of computational burden, this detection approach relies on measures decreasing the amount of data to be checked. In case of anti-virus search engines, a common measure is called *filtering*. A filter can be anything that is virus-specific, e.g. information where to search, function names, and similar. It bases on the finding that malwares typically infects only subsets of objects. The problem about *algorithmic scanning* can be seen in the inability to work on most polymorphic[14] and encrypted malwares.

X-RAY scanning: Detecting malware with encrypted components is another important challenge that has to be addressed by detection engines. Attacking the encryption of a malware is called *X-RAY scanning*. *X-RAY scanning* basically takes care of encryption by searching for encrypted malware parts on selected areas, e.g. top, tail, or near of entry-points.

Code emulation: Code emulation bases on a virtual environment that is created in order to observe the behavior of a suspicious application. This application is installed into this encapsulated environment for preventing it to infect components of the real system. Such an environment is also called a *sandbox*. Bishop [31] defines a *sandbox* as follows: "A sandbox is an environment in which the actions of a process are restricted to a security policy." *Sandboxes* can typically be found in anti-virus software but basically every virtualized environment can be regarded as one. Examples for this are Android applications which run in a separate Java-based sandbox increasing runtime security of unknown applications.

Metamorphic malware detection: This term describes various approaches that try to detect malicious applications through other indicators than, e.g. code strings. One example for this is the *geometric detection* that monitors file system alterations of software for detecting malicious behavior and files.

Heuristic Detection: Another common approach is called *heuristic detection* which basically bases on machine learning algorithms. With these, anti-virus software can detect a new malware through comparing it with behavior of similar malwares or through detecting anoma-

[14] a certain malware having different appearances

lous actions that indicate malicious intentions. Although having the advantage of being able to detect unknown malwares, heuristic detection is prone to *false-positives*[15].

3.4.2 Intrusion Detection Systems

IDS are also capable of detecting malware, therefore, we will introduce these systems in this section. Scarfone et al. [189] define intrusion detection as a process of monitoring predefined systems[16], like networks and hosts, for gaining data describing the state of the monitored systems. This data is analyzed for finding incidents that help to indicate malicious events. These incidents may be caused by malicious activity initiated by malware. Non-malicious actions, like typing wrong IP-addresses that point to servers where the user is missing access authorization, can also be recognized as incidents. Kruegel et al. [120] use a similar definition for intrusion detection with following motivating points:

1. Surveys have shown that most computer systems are flawed by vulnerabilities regardless of manufacturer or purpose [123]. Users and administrators are generally very slow in applying fixes to vulnerable systems [181]. As a consequence, many experts believe that computer systems will never be absolutely secure [27].

2. Deployed security mechanism, e.g., authentication and access control, may be disabled as a consequence of misconfiguration or malicious actions.

3. Users of the system may abuse their privileges and perform damaging activities.

4. Even if an attack is not successful, in most cases it is useful to be aware of the compromise attempts (for learning purpose).

Intrusion detection systems are software tools that automatically gather data, analyze it and identify such incidents. These systems evolved to intrusion prevention systems (IPS) including additional prevention capabilities.

[15] related terms and definitions are given in Section 3.4.4

[16] we will stick to the host-based approach in this work since it gets more detailed information on the monitored system

Preventive measures can be disconnecting or locking hosts that produce incidents [189]. For consistency reasons, we will use the term Intrusion Detection and Prevention System (IDPS) wherever applies. An IDPS has three key tasks:

1. Monitor system characteristics, e.g. application, network, or operating system behavior.

2. Analyze the monitored data for detecting incidents, e.g. security policy breaches or application misbehavior.

3. Initiate measures basing on the detection results, e.g. generate report, lock systems, or disconnect unauthorized entities.

IDPS basically use one or a combination of the following approaches for detecting incidents [120]:

- Misuse-based system

- Anomaly-based system

Misuse-based system Misuse-based detection uses a knowledge base of predefined patterns (signatures) that can be matched with the monitored data. Signatures can have various forms, e.g. strings, execution stacks, or binary information. Whenever such a recorded event matches a signature, the IDPS initiates a predefined measure on the detected malicious activity. Misuse-based detection has the advantage of producing only few false-positives which means detecting normal action to be malicious. But using signatures comes with drawbacks. Since the knowledge bases is the only source for identifying malicious events, new and unknown threats can not be detected in most cases. Furthermore, making the signature very broad increases the rate of falsely classified normal events. Making the signature too specific will allow attackers to easily modify their software for not getting detected [80, 168, 238].

Anomaly-based system Scarfone et al. [189] state that an anomaly-based system bases on the assumption that all anomalous activities are malicious. Therefore, the system creates a model of normality basing on

normal activity which then enables the system to indicate anomalous activity. Anomaly-based systems have the advantage of being basically able to detect even unknown threats. The downside of these systems is that they tend to produce a lot of false positives which results in a bad accuracy. Examples for this approach are [140, 90, 13, 171].

Host-based Intrusion Detection Systems

A host-based intrusion detection system monitors system characteristics of a single host for identifying suspicious activity [189]. These characteristics can contain network, system, and application information that help to distinguish between normal and abnormal behavior on the monitored device. The techniques used by the various systems depend on the focus of these. They can rely on a single method or cover a broad range of functions. Common functions are:

Kernel monitoring includes tracing system calls for identifying call sequences matching malicious activity. Additionally, API and library calls can be monitored for detecting intrusive and malicious behavior.

Network monitoring includes capturing data traffic on transport layer or higher. Beside network traffic, network configuration is also of interest, e.g. network interfaces might be set to promiscuous mode without user interaction.

File system monitoring can be done by performing various measures. File integrity checking cryptographic checksums can be done for verifying files to be in unchanged state. File attribute checking can indicate ongoing attacks which may include changing ownership of important files. File access checking can indicate malicious activity on critical files. All these measures can only indicate already ongoing attacks since they are called periodically.

Log file monitoring bases on analyzing application output files. Applications can log incidents, commands or application usage for indicating their status and problems. Filtering these logs can help to identify malicious activity.

Sandboxing is a technique that uses virtual environments for running suspicious programs before including them to the real system. It can prevent malicious programs, e.g. Trojan horses, to compromise a target

system since malicious calls are realized within this virtual environment.

Vigna et al. [231] state that host-based intrusion detection systems have both advantages and disadvantages when compared to network-based intrusion detection systems. HIDS are able to access semantically rich OS-based information whereas NIDS only see packet streams that might be fragmented or even encrypted. Furthermore, the amount of data that has to be handled is usually more limited. Another advantage is that HIDS tend to be harder to bypass since several different characteristics of the system are monitored. This makes it more difficult for attacks to evade detection in all areas. Finally, it is easier for HIDS to counter attacks since the source of malicious activity can be addressed directly.

The main disadvantage that is pointed out by Vigna et al. [231] is that once a process gains administrative rights it is able to manipulate the complete system including the HIDS. Therefore, it is essential to prevent this event. Another major problem is that HIDS might influence the system performance which also affects application usage. A NIDS is able to monitor several hosts without affecting their performance substantially. Furthermore, monitoring several systems simultaneously enables the system to detected wide-spanned attacks that might not be recognized on a single system. It also decreases maintenance costs since only one system has to be set up and controlled. A HIDS requires installation on every system that is planned to be monitored.

Audit Data Gathering / Monitoring Lee et al. [128] state that their research aims to develop a systematic framework to semi-automate the process of building intrusion detection models. A basic premise in their work is that abusive behavior can be distinguished from normal behavior by mining audit data from the corresponding system. Their IDS/framework bases on three algorithms/methods (classification, association rule, and frequent episode) and additional programs, which help to build detection models. The key architecture seems to consist of sensors, detectors, and model generators. Similar approaches are described in [208, 129].

State of the Art and Taxonomies of IDSs Several publications can be found to describe current and ongoing research, as well as commercially available products in the field of IDSs. Stefan Axelsson [17, 18] presents a

survey on several research publications on IDSs. In this survey he proposes "A Taxonomy of Intrusion Detection Principles".

Allen et al. [8] present state of the art and best practices in IDSs. They present example projects from research and industry and point out organizational issues when deploying and using IDSs in companies.

3.4.3 Static Analysis versus Dynamic Analysis

As explained earlier, virus scanners and Intrusion Detection Systems try to detect malicious software and activities using *static* and *dynamic analysis*. The major difference between *static* and *dynamic analysis* is how the data is acquired. Methods that are used to analyze the monitored data can be the same for both approaches.

Static Analysis

Static analysis represents an approach of checking source code or compiled code of applications before it gets executed. Chess and McGraw [43] state that *static analysis* promises to identify common coding problems automatically. While manual code checking is also a form of *static analysis*, software tools are used in most cases in order to perform the checks[17]. Chess and McGraw additionally claim that good static checkers can help to spot and eradicate common security bugs.

Static analysis can use simple pattern search operation or slightly more complex machine learning approaches in order to detect flaws and weaknesses in the code of software. A simple search might aim for finding insecure function calls in `C` programs. A more complex approach might be the usage of statistical methods in order to determine occurrences of certain calls. Either way, static analysis can be an appropriate code checking tool that can help to improve quality or security of software. The major drawback, which is also stated by Chess and McGraw [43], is that a static tool's output still needs human evaluation. This means that if a tests concludes with "no errors found" there still is no guarantee that the code is free of flaws or other issues. The results depend on the up-to-dateness of the corresponding detection rules and methods.

[17] e.g. ITS4, Flawfinder, or RATS

Related work can be found in Wagner and Dean [233] and in several other publications [134, 28, 44, 71, 134, 59, 149, 235, 233, 233, 194, 196].

Dynamic Analysis

One of the first articles on dynamic analysis for security purpose was published by Forrest et al. [74]. They used a machine learning approach using system call histories to learn normal system behavior. Abnormal behavior in terms of call sequences was used to indicate malicious activities. Szor [211] states that *dynamic analysis* techniques focus on black-box testing. Black-box testing is the process of executing a malware[18] in order to monitor its behavior. Typical aspects that are monitored are: network port usage, transmission of network packets, system call sequences, processes, and file system and registry[19] changes.

Work related to dynamic analysis can be found in [23, 45, 134, 23, 198, 131, 26, 5, 10, 14, 144].

3.4.4 Related Definitions and Terms

When talking about detection of incidents, following terms are commonly used in order to describe the performance of detection approaches. Taking all possible and actually happened *incidents* into account, the set of incidents can be separated by *real negatives* and *real positives*. *Real negatives* are represented by all occurred incidents that were intended to happen while *real positives* are the set of incidents involved into malicious activity. These terms are not often used[20] in conjunction with scientific articles but they are still needed in order to evaluate every detection approach.

Four classes are commonly used in order to describe detection results of corresponding approaches. One can interpret the terminology in following generic way:

```
[evaluation of the detection result] [detection result]
```

True negatives (TN) describe the class of events that were detected being benign where the detection result was actually true/right. *False neg-*

[18] should be done in a virtual machine
[19] on Windows operating systems
[20] since they are obvious the research community

atives (FN) describe the class of events that were detected being benign although they were actually malicious. This can be seen as the worst case of every detection mechanism since it gives the users a false sense of security which is also emphasized by Chess and McGraw [43]. Therefore, most developers and researchers modify their detection approaches to minimize false negatives. This comes at a cost which is often an increase of *false positive (FP)*. *False positives* are incidents that were detected being malicious where they actually were not. As an example: a mobile intrusion detection system monitoring the behavior of a smartphone might respond to an event where 100 short messages (SMS) were sent to contacts included in the phone book. This incident might have been caused by a malware trying to trap these 100 contacts or it might have been an intended messaging on New Years Eve. In the latter, it is a false positive where the first case is a *true positive (TP)* which are basically incidents such protective system should detect. On Table 3.2, the four detection classes are aligned to the real world classes.

Table 3.2: Detection results and reality

Reality	real negatives		real positives	
Detected	negative	positive	negative	positive
Result	true negative	false positive	false negative	true positive

The terms *accuracy*, *TP rate*, *false discovery rate*, *precision*, and *FP rate* are widely used in articles describing detection approaches. *Accuracy* describes the rate of correctly detected incidents where the higher accuracy values are, the better.

$$accuracy = \frac{TP + TN}{TN + FP + FN + TP} \tag{3.1}$$

The *TP rate* which is also called *quality*, *hit rate*, and *recall* refers to the rate of correctly detected incidents out of all detected incidents. The higher this value is, the better the detection should work.

$$\text{TP rate} = \frac{TP}{FN + TP} \tag{3.2}$$

The *false discovery rate* describes the rate of falsely detected incidents out of all detected incidents where the smaller the value, the better the system

can be.

$$\text{false discovery rate} \quad = \quad \frac{FP}{FP + TP} \tag{3.3}$$

Precision is also called *positive predictive value (PPV)* and refers to the rate of correctly detected incidents out of all positively detected events.

$$\text{precision} \quad = \quad \frac{TP}{FP + TP} \tag{3.4}$$

The *FP rate* which is also called *false alarm rate* or *fall-out* shows the rate of falsely classified benign incidents out of all benign incidents. The smaller this value is the better the detection might work.

$$\text{false alarm rate} \quad = \quad \frac{FP}{TN + FP} \tag{3.5}$$

All of these terms try to indicate the capabilities of certain detection approaches but good values do not necessarily mean the detection works well. One might consider the case of a detection system that was tested with 1000 events where one of these was malicious. If the approach would be capable of detecting this single incident, the accuracy and TP rate of this system would be 1 although only one case was tested and available. Therefore, every test run should include as much benign and malicious data as possible. Else, the results can obviously only give indications on the detection capabilities but no considerable numbers.

3.5 Summary and Conclusion

In this chapter, we introduced the basics of malicious software in Section 3.1 where we started with general descriptions which are applicable to most computer systems. In Section 3.2, we presented related work in the field of smartphone malware which leads to a detailed description on the evolution of these malicious applications in Section 3.3. The amount of smartphone malware increased from year 2004 on and by the end of 2010, more than 500 malwares will have appeared. While most of the smartphone malwares are Trojan horses manipulating the system, the share of profit-oriented malware increases steadily. By the end of 2010 more than 20% of all malwares will try

to fool users in order to earn money through premium services that are used secretly in background. Since this seems to be the first profitable payload, an increase of such malware targeting smartphones can be expected.

In this work, we present various new approaches in order to detect smartphone malware. Therefore, we also give general descriptions on detection principles in Section 3.4 which includes descriptions on *static* and *dynamic analysis*. The main parts of this thesis will then be divided by approaches basing on either *static* or *dynamic analysis*.

Chapter 4

Malware Detection through Dynamic Analysis

As seen in Section 3.4.2, dynamic analysis can be part of virus scanners and intrusion detection systems that protect host computers or networks. The key point about dynamic analysis is that data is acquired at runtime in comparison to static analysis which does not require executing binaries for investigating them. This can have the advantage that incidents are detected in real time enabling the system to start appropriate countermeasures in time. Furthermore, single events might seem unsuspicious, but in a sequence of events they might indicate a malicious process with the goal of harming the system. This might also be detected by a system using dynamic analysis in order to understand observables running a monitored system. Another advantage of dynamic analysis is that when used in a debugger, malware can be analyzed step by step which can lead to the generation of a signature usable by anti-virus software. Signatures are typically a very efficient way of detecting *known* malware but new and *unknown* malware normally cannot be detected which is the biggest drawback of this approach.

The contribution of the chapter is three-fold. First, results on monitoring Windows Mobile and Symbian OS devices for anomaly detection are presented. Second, an architecture that enables monitoring and detection of anomalies on Linux-based Android devices is shown. Third, results on applying dynamic analysis of smartphone binaries in a cloud-based system are given.

4.1 Introduction

In this section, focus is set on smartphones being monitored at runtime in order to acquire data indicating malicious activity. Several approaches and architectures will be presented in this section that enable dynamic analysis of smartphone executables.

In Section 4.3, an approach to monitoring smartphones in order to extract data (features) that can be used for *remote* anomaly detection is introduced. Anomaly detection does not require signatures in order to work which allows the detection of new and unknown malware. It can also be used in conjunction with signature-based schemes to decrease the response time whenever new malware emerges. For this purpose, the anomaly detection algorithms have to learn the normal behavior of an user and device in order to be able to distinguish between normal and abnormal, possibly malicious actions. The extracted features are sent as vector to a remote system taking the responsibility for extended security measures away from the probably unaware user. These vectors are processed by methods and algorithms from the field of artificial intelligence, like Artificial Immune Systems (AIS) [75] or Self-Organizing Maps (SOM) [7], in order to detect abnormal behavior.

In Section 4.4, a general monitoring and detection architecture which is also used in approaches being tested by us on the Android platform is presented. In this section detailed descriptions on how all components of the system interact are given. Additionally, insights into our detection components and processes are presented.

In Section 4.5, work-in-progress in employing cloud services for detecting malicious applications on smartphones is described. System calls being made by binaries using the `strace` command on smartphones are monitored. We believe malware detection can follow the cloud-computing paradigm by processing these calls thoroughly using different techniques. For decreasing computational burden on the smartphones, no complex call sequences are processed. Data structures are limited to simple trees pointing to system calls that have been made including the frequency of calls. Various classification algorithms including Support Vector Machines and Tree Kernels on benign and malicious binaries are evaluated. Preliminary results appear to be promising and need more investigation.

4.2 Related Work

Several promising approaches for anomaly detection on stationary and mobile systems, such as smartphones, have been presented:

Forrest *et al.* [74] propose a method for anomaly detection where normality is defined by short-range correlations in a process' system calls. Their experiments (involving "sendmail", "ftpd" and "lpr") show that short sequences of system calls in running processes generate a stable signature for normal behavior. The signature has low variance over a wide range of normal operating conditions and is specific to each different kind of process, providing clear separation between different kinds of programs. Furthermore, it has a high probability of being perturbed when abnormal behavior, such as malicious activities, occur.

Davis et al. [105] propose a host based intrusion detection engine (HIDE) which has the goal to alert an user when a suspected attack is underway before irreparable damage is caused. Therefore, HIDE first monitors anomalous behavior of the battery when it fails to go into suspension or sleep mode. Then HIDE sends an alarm message to the user and the nearest proxy server and starts to write logs that contain the causes of the higher power consumption. In the next step, HIDE suggests mitigation measures. The authors run preliminary tests on their system using "ping" command in order to generate abnormal battery depletion activities (ABDA) which is detectable through their engine. Further sophisticated test are required for proving the functionality of their system.

Cheng et al. [41] developed a system that uses system status and log file monitoring in order to detect malware infections. Therefore, a monitoring client is installed on a Windows Mobile 5 device that is able to determine its own phone number, the date, the cell id, the SMS and calling logs. Statistical and abnormality-based analysis for processing the monitored data is used. For privacy and authentication, ticketing and encryption is introduced. Whenever an infection is detected, the system alerts the corresponding device as well as all devices with contact to the infected one. They evaluate their approach with a simulation basing on SMS traces coming from a cellular-network provider from India.

Buennenmeyer et al. [34] present a similar approach to Davis et al. [105] which is monitoring current changes on a smartphone in order to detect anomalies. The changes can be caused by malwares or external attackers,

e.g. flooding or network probing. The monitored data is sent to a remote server that creates profiles of each monitored device and hence is able to detect anomalies. They evaluate the power consumption of the monitoring and state that the client uses less than 2% of battery resources compared with the corresponding baseline battery lifetime.

Miettinen et al. [144] designed an intrusion detection framework, which uses host-based and network-based intrusion detection. If an anomaly is detected on a mobile device, the device sends an intrusion alert to a back-end server. This server is able to collect further information from network-based sensors in order to create network related intrusion alarms when necessary. They use a correlation engine in order to correlate the device and network intrusion alarms.

Samfat and Molva [187] presented a distributed intrusion detection system for cellular networks that tries to detect abusive behavior like masquerading or eavesdropping in future inter-networks. They use learning algorithms in order to obtain user profiles, which in turn are used as signatures to detect abnormal behavior. Example features are *call length* or *cell information*. They use network-based intrusion detection and do not try to detect on-device malware.

Bose *et al.* [33] propose behavioral detection framework to detect mobile malware, instead of common signature-based solution currently available for use in mobile devices. They represent malware behaviors based on a key observation that the logical ordering of application actions over time often reveals the malicious intent even when each action alone may appear harmless. Also, they propose a two-stage mapping technique that constructs malicious behavior signatures at run-time from the monitored system events and API calls while studying 25 distinct families of mobile malware in Symbian OS. They discriminate the malicious behavior of malware from the normal behavior of applications by training a classifier based on Support Vector Machines. Detection rates from simulated and real malware samples are stated to be better than 96%.

Kang *et al.* [111] present an approach considering system call sequences as a classification problem on a bag of system calls. In this bag, the frequency of the system calls is stored where the call ordered is dismissed. Experimental results on public data sets show that that the frequency information is effective enough to discriminate between normal and abnormal sequences.

Lee *et al.* [127] present an approach to use data mining techniques to discover patterns of system features that describe application and user behavior. Therefore, they use system call traces from the *sendmail* command, as well as *tcpdump* data in order to detect suspicious behavior.

Chaturvedi *et al.* [40] present an approach for capturing data-flow behaviors on top of system call traces. Therefore, they provide a formal definition of data-flow behaviors and present algorithms for building such models. These models enable them to detect even sophisticated attacks that normal system call traces-based approaches mostly do not detect. The drawback of their approach is that the needed training data was sized from 1.4 to 4 million system calls for each trace (100 to 300 MB). Additionally, the overhead of collecting these large-sized traces were not presented.

Nash *et al.* [158] designed an IDS against an unique form of DoS attack known as a battery exhaustion attack by taking into account the performance, energy, and memory constraints of mobile devices. Their IDS uses several parameters, such as CPU load and disk accesses, to estimate the power consumption per process using a linear regression model, to identify processes that are potentially battery exhaustion attacks.

Kim *et al.* [114] propose a power-aware malware-detection framework that monitors, detects, and analyzes previously unknown energy-depletion threats. The framework includes a power monitor which collects power samples and builds a power consumption history from the collected samples, and a data analyzer which generates a power signature from the constructed history. Similarities between power signatures are measured by the χ^2-distance, in order to reduce both false-positive and false-negative detection rates.

In Schmidt et al. [197, 198] the authors demonstrate how to monitor a smartphone running Symbian OS in order to extract features that describe the state of the device and can be used for anomaly detection. These features are sent to a remote server, because running complex IDS on this kind of mobile device still is not feasible, due to capability and hardware limitations. They give examples on how to compute some of the features and introduce the top ten applications used by mobile phone users based on a study in 2005. The usage of these applications is recorded and visualized and for a comparison, data results of the monitoring of a simple malware are given.

Another battery-based IDS is presented by Jacoby et al. [106]. It mea-

sures the device's power consumption, which is correlated with the application activity on the device by running a rule-based host intrusion detection engine.

Wang et al. [235] observe the usage of DLL and API methods for training support vector machines (SVM) for behavior detection. Behavior-based approaches normally suffer from a high false positive rate while needing a significant amount of processing power, storage, and memory. The authors claim an accuracy of 99% but do not present the corresponding detection rate which would help to rate the quality of the results.

Bayer et al. [23] present a tool named TTanalyze. This tool is able to monitor Windows applications dynamically by monitoring usage of system and API calls. The focus of this work does not lie in the detection of malware; the aim is to understand the malware behavior for decreasing the window of vulnerability.

Kirda *et al.* [115] present a spyware detection technique that overcomes some of the limitations of existing signature-based approaches. Their technique is based on an abstract characterization of the behavior of a popular class of spyware programs and applies a composition of static and dynamic analysis to binary objects to determine if a component monitors users' actions and reports its findings to an external entity. Their characterization is resilient to obfuscation; independent of the particular binary image thus can be used to identify previously unseen spyware programs. Because the interaction with the operating system is necessary for a spyware component, they analyze those Windows API calls that a component can use to leak information from the current process, especially the ones that are performed in response to events. For that reason, they use dynamic analysis to monitor interaction of the component with the browser and record all of the browser functions that are invoked in response to events, in order to determine the code regions that are responsible for handling events. Then, they use static analysis to examine these regions for the occurrence of system calls relevant to the creation of threads or timers to assume that any imported API function, which exists in that region, can be invoked in response to an event. Finally, they have automatically generated an API call blacklist from extracted data using frequency analysis.

In the opinion of the author, the approach of Miettinen et al. [144] is the most promising. It regards the resource constrains of smartphones but still allows complex analysis of indicated intrusions. With the increasing

capabilities of smartphones, more and more functionality can be moved from the server to the mobile devices. This approach may be supported by the system from Samfat and Molva [187] which would add intrusion detection capabilities to the mobile phone network.

4.3 Monitoring Smartphones for Anomaly Detection

In this section, it will be demonstrated how to monitor a smartphone running Symbian or Windows Mobile operating system in order to extract features that describe the state of the device and can be used for anomaly detection. These features are sent to a remote server because running a complex intrusion detection system (IDS) on this kind of mobile device still is not feasible due to capability and hardware limitations. Examples are given on how to compute some of the features and introduce the top ten applications used by mobile phone users based on a study in 2005. The usage of these applications is recorded by a monitoring client and visualized. Additionally, monitoring results of public and self-written malwares are shown. For improving monitoring client performance, principal component analysis (PCA) is used which lead to a decrease of 80% of the amount of monitored features. Additionally, the performance results of two approaches are shown, basing on an Artificial Immune Systems and a Self-Organizing Map, respectively.

This section is structured as follows. In Section 4.3.1, we give a brief overview on the framework we use for processing the monitoring data. In Section 4.3.2, we show how to build a monitoring client for smartphones and give explicit examples on values that can be extracted from Symbian OS and Windows Mobile devices. In order to be able to learn what is normal on smartphones, we map actions excerpted from a study on mobile phone usage to different use cases and specify testing scenarios for these. Examples of these together with the corresponding monitoring results are given in Section 4.3.3. In Section 4.3.4, we present the results of principal component analysis for reducing the amount of features.

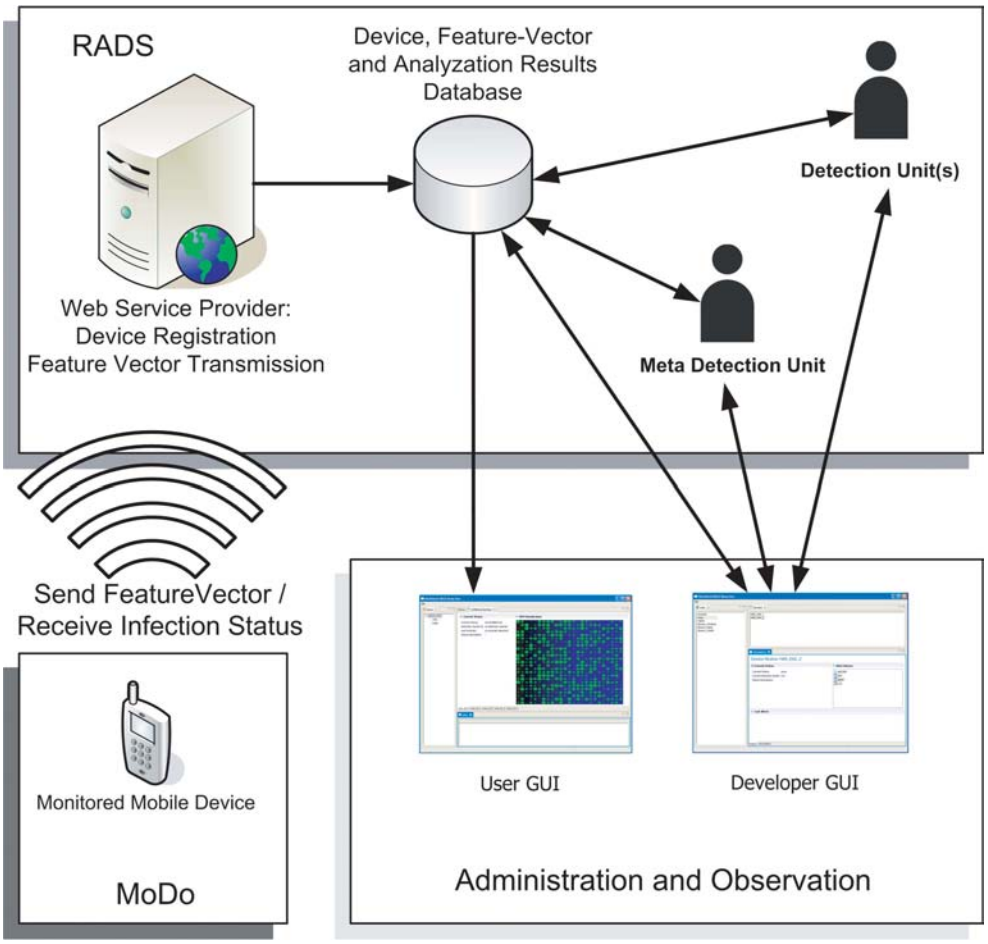


Figure 4.1: The remote monitoring framework consists of three main components: I) a monitoring client is installed on user side that is capable of extracting system status information. II) The remote server receives and analyses the data for identifying anomalies. Detection units are managed by meta detection units being able to assign different kinds of detection units to suitable kind of data. III) Users and developers can use an administration and observation interface for interacting with the system.

4.3.1 The Monitoring Framework

For understanding the purpose of our work we present the corresponding framework on Figure 4.1 in which our monitoring clients are included. We have to note that the focus of this work is describing the development of a monitoring client for smartphones so we will not discuss design or imple-

mentation issues concerning the global framework.

In general, our framework looks similar to the high-level system descriptions of Miettinen et al. [144]. The main difference to their approach is that do not use a correlation engine for analyzing the data. Instead, we use components which will referred to as detection and meta detection units in this section. Additionally, our system we will provide detailed information on our system in comparison to the generic system recommendations made in [144].

The framework consists of three components: smartphone monitoring client, remote anomaly detection system (RADS), and visualization. The client will be described in Section 4.3.2. The RADS provides a web service for communicating with the client. The received monitored features are stored into a database which is accessible by detection units. These detection units analyze the data for finding anomalies which might indicate malware activity. The detection units are implementations of machine learning algorithms, e.g. AIS [75] or SOM [7], that are able to handle multivariate data in order to produce detection results. The *meta* detection units work similar to the ordinary units except that they analyze results for weighing results from different detection algorithms. Since different algorithms perform different on certain device usage data, using meta detection units might improve overall detection results. The visualization indicates the device status, incidents, and detection results.

4.3.2 The Monitoring Client

Intrusion detection can be separated into two fields: signature-based misuse detection and anomaly detection. As the devices are monitored for anomaly detection, it is important to monitor device data that enables differentiation between normality and anomalies. Eugene Spafford et al. [208] points out that host-based approaches, direct data collection techniques, and internal sensors are preferable to network-based approaches, indirect data collection techniques, and external sensors. This was taken into account when designing our monitoring client.

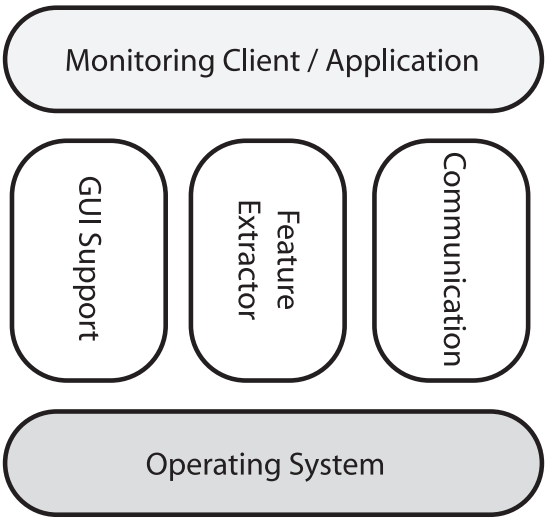


Figure 4.2: The generic client structure is divided into three center components that realize communication, extraction of system information, and GUI support.

Generic Client Design

We propose a generic architecture in Figure 4.2 including three main components for the monitoring client: User Interface, Communication Module, and Feature Extractor.

The User Interface enables client configuration, like changing server IP address or port. It can be used to visualize the state of the monitoring client, e.g. sending or buffering, and can indicate anomaly detection results.

The Communication Module is responsible for managing connection states and sending or buffering the monitored features which is shown on Figure 4.3. If the client cannot connect due to signal loss, it starts buffering until a connection can be established. If a connection is not possible before the buffer is filled, it adds the last extracted vector and removes the first.

The Feature Extractor has several different components for gathering and computing features. Features describe the state of the monitored device. They represent various measurements and observations of resources and

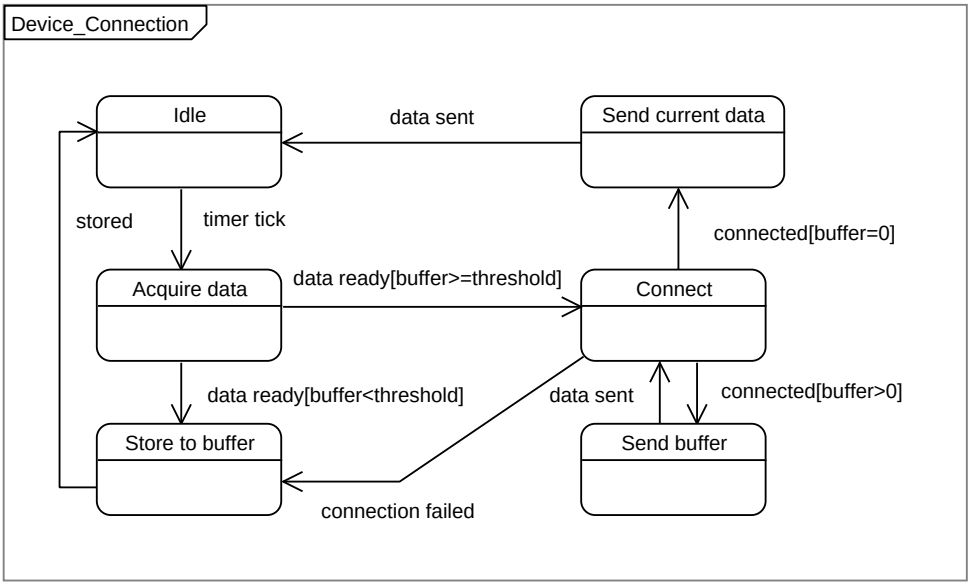


Figure 4.3: The possible connection states of the monitoring client

other hard- and software components. If no direct interfaces are provided by the operating system, features are extracted by using algorithms or methods which provide approximated results. This is done with care, as additional encumbering of the already limited device possibly distorts the monitoring results.

Securing the Monitoring Client As the communication or even the monitoring client itself can be targeted by malware, it is important to secure the functionality of the client. Using encryption for the communication channel should be a proper way to secure data transmissions. Application data does not need to be secured beyond existing application security measures since all data we extract from the system can be extracted by anyone else. Securing the application itself is more complicated. The Symbian OS API provides a method for setting processes to different *critical* levels. On highest level, if the monitoring client process is killed, the device reboots and restarts the process. This functionality was not added yet, as it obviously could lead to denial of service attacks on the device, but at least it would guarantee either that the monitoring agent is running or that the user brings the device to a specialist for fixing the problem. Another possibility

is checking the running applications which are clearly identifiable through an unique Symbian application ID. As soon as an unknown application is started, it could be compared with an application white list that includes all allowed programs. If an unknown process is started the system can kill it or alert the user and the system. Different operating systems might support similar functionalities where it is probable that not all clients will be able to implement and run the same security measures.

The Symbian Client

Figure 4.4: Nokia E61 and HTC TyTN B smartphones running the monitoring client.

The monitoring client was developed in Symbian C++ version S60 3rd with "Nokia Carbide.vs" and consists of the three proposed components. *The User Interface* can be used to change server port and address, to start, stop, or move the client into the background. Further user information can be inserted in order to control access to the remote server. For reasons of program stability and to prevent interferences, the GUI is running in a thread separate from the other components. Further work may even remove the user interface to a separate application, since there is no need to tie up GUI resources for an application running in the background. *The Communication Module* uses SOAP[1] Web services on top of TCP/IP in order to communicate with the server. As we found out, sending data — or

[1]This was formerly known as Simple Object Access Protocol.

even just remaining in ready-to-send mode — is rather expensive in terms of battery power. To prevent the rapid depletion of the power source, all data is stored locally and sent in bulk after reaching a certain threshold level. *The Feature Extractor* is triggered to fetch new monitored data every 20 seconds which then is sent to the server using the appropriate service.

Table 4.1: Excerpt of the extracted features

Name	**Complexity**	**Description**
RAM FREE	simple	Indicates the amount of free RAM in Kbyte
USER INACTIVITY	simple	Indicates, if the user was active in the last ten seconds
PROCESS COUNT	medium	Indicates the amount of running processes
CPU USAGE	complex	Represents the CPU Usage in percent
SMS SENT COUNT	complex	Represents the amount of SMS messages in the sent directory

The Symbian Features Symbian OS[2] provides some programming interfaces for extracting features, e.g. fetching the amount of free RAM or the user inactivity time are simple API calls. But not all areas are covered by API calls, especially reading network traffic packets cannot be done by average developers as the application programming interfaces are restricted. Some other features need complex method constructs in order to be extracted. We distinguish between three different method complexities: *simple*, *medium* and *complex*. Features that can be called through Symbian C++ interfaces taking only one or few lines of code are categorized as *simple*. Features that need several classes or algorithms to be computed are marked as *complex*. Everything in between is marked as *medium*. Some of the features can be used to identify and manage observed users or devices, e.g. IMEI[3] and IMSI[4]. The IMEI and IMSI are unique numbers that clearly identify mobile devices or mobile network users. In the following, we describe how to compute some of the features shown on Table 4.1 with pseudo

[2]This was tested on Version 9.1 S60 3rd.
[3]International Mobile Equipment Identity (IMEI).
[4]International Mobile Subscriber Identity (IMSI).

code. Some of these will be used to visualize user activity in Section 4.3.3. In order to present, how Symbian C++ programming looks like, we will show the real call for getting the available RAM:

RAM FREE is a feature that can be easily extracted. All applications need more or less RAM in order to work, so every running program/malware should have impact on this value.

```
User::LeaveIfError(HAL::Get(
        HALData::EMemoryRAMFree, iFreeRamSize));
```

USER INACTIVITY indicates if a button was pressed within the last 10 seconds. If so, a "0" is returned else a "1". This feature uses a function that returns the absolute user inactivity time in seconds. This value is very interesting for giving hints on activities that are not directly caused by the user and happen automatically and/or periodically in the background.

Table 4.2: Pseudo code for indicating user activity

```
GET UserInactivityTime

IF UserInactivityTime ≥ 10 seconds
      RETURN User is inactive
ELSE
      RETURN User is active
```

The PROCESS COUNT can be easily computed through a while loop that is checking the existence of processes. Each started application should increase the process count at least by one, and so should malware.

The CPU USAGE cannot be read through a given Symbian OS interface. While searching for an approximation we found a method described by Marcus Gröber[5] that manually checks whether the CPU is busy or not. This is done by requesting a timer event with low priority 100 times a second. Another request with high priority checks every second how often the low priority request was actually called. The answer can be used to approximate the usage of the CPU since the more the CPU is busy the less

[5] http://www.mgroeber.de/epoc.htm

Table 4.3: Pseudo code for getting the process count

```
WHILE there are more processes
        INCREASE counter
        FETCH information from process object
        STORE process information

RETURN the counter
```

the low priority request will be called. The following code fragment shows the main calls and functions of this method.

Table 4.4: Pseudo code for approximating the CPU usage

```
CREATE new request Active Object low priority
CREATE new check Active Object high priority

SEND low priority time request
        to CPU 100 times/second
CHECK number of accepted requests/second
        Return approximated CPU usage/second
```

SMS SENT COUNT, like every feature relating to messaging (SMS, MMS, and email), needs some more complex functions to be computed. But once implemented, most of the messaging-related features can be extracted using the same classes. Together with USER INACTIVITY this feature can help to indicate malware sending messages to cost-intensive premium services.

Table 4.5: Pseudo code for getting the amount of SMS messages sent

```
CREATE messaging session
CONNECT messaging session to sent folder
        SELECT SMS sent folder
                RETURN amount of SMS messages
```

The Windows Mobile Client

The Windows Mobile monitoring client was developed in `C#` for Version `6.0` using Microsoft Visual Studio 2008 in the final stage. Similar to the Symbian OS client the Windows Mobile client bases on our generic client design. The extracted features are the same as on Symbian OS. They only differ in the way they are called through the given Windows Mobile APIs. In general, developing the monitoring client on Windows Mobile was easier than on Symbian OS since provided APIs are well documented and running the needed development and build environment does not need that much effort as with Symbian OS. Various code examples can be found online which are easy to integrate. Figure 4.4 shows the HTC TyTN B running the monitoring client.

A First Android Architecture Draft

The Open Handset Alliance Project "Android"[6] aims for developing the first complete, free, and open mobile platform. There is an ongoing debate whether open source software is *more* secure than closed source software where the most important pros and cons can be found in Lawton [125]. Since the discussion reflects various different opinions which are all argued well, we will omit to continue the debate in this work. We only state that open source software has the potential to be more secure where it depends on the quality of the code reviewers whether it is or not.

Android bases on Linux kernel 2.6 and uses Java on top of Linux for providing a development and runtime environment for 3rd party applications. The Android Java environment and byte code is not compatible to Java SE or ME and uses a proprietary virtual machine called Dalvik VM. This is optimized for mobile usage and provides one virtual machine instance for each running Java application. This enables Linux to handle the instances separately where every application is restricted by Linux file system rights. Since the openness of Android allows modification of almost every software component and Linux was used as core OS, this platform provides a good foundation for building a monitoring agent benefiting from several years of Linux security research. A first simple architecture basing on our generic approach can be seen on Figure 4.5. Since accessing security relevant system characteristics might be problematic using JNI native calls from Java

[6] `http://code.google.com/android/`, visited 15.3.2009.

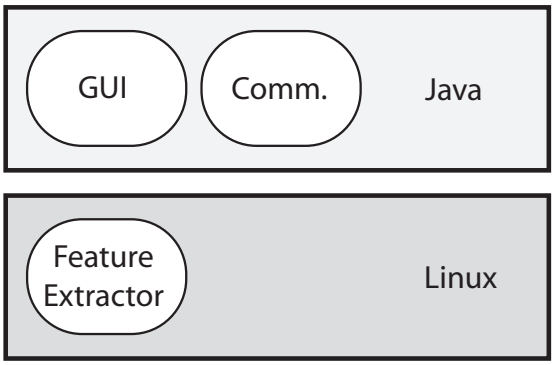


Figure 4.5: This simple monitoring architecture considers limited device capabilities present in smartphones from the year 2006.

applications, the extractor was placed on the Linux OS layer. Application control (GUI) and communication are placed on the Java layer since the Android Java environment provides various libraries for implementing these.

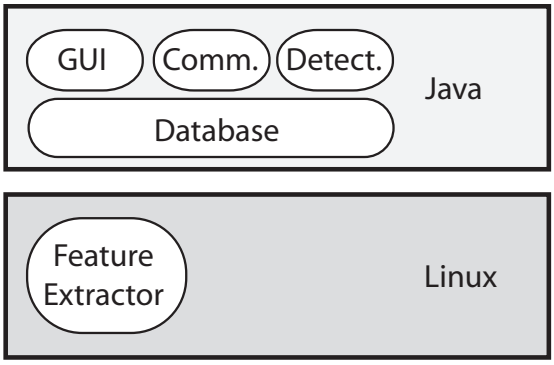


Figure 4.6: This improved architecture for future smartphones includes the idea to move storage, processing, and detection capabilities to the phone itself.

This simple architecture can be extended following the assumption that the capabilities of smartphones increase steadily. This means that early client-server design decisions that moved most of the data analysis processing to the server may be changed. Relocating some of the server functionality to the client side will result in the reduction of communication latencies. Additionally, having a light-weight detection on the client might lead to a

dramatic decrease of communication data as the client could only send data referring to already detected anomalies. Furthermore, Android provides access to local database which can be used to store the monitored data. On request of the server, database excerpts can be sent to the remote detection side for further analysis. The improved architecture can be found on Figure 4.6.

The monitored feature set of android devices will be slightly different from the one of Symbian OS and Windows Mobile. Since Linux intrusion detection research is mature, results from various systems can be taken into consideration, e.g. [8] and [17]. A key issue will be to identify and merge the most promising features known from file system, log file, connection, and kernel monitoring systems.

4.3.3 Experiments

As our goal is to provide data that enables differentiation between normal and malicious device usage we need to know first what actually normal usage looks like. TNS Technology released a booklet [213] sourced from the TNS Technology's Global Technology Insight (GTI) 2005 where typical user actions on mobile phones are described. We excerpted actions that we performed on Nokia E61 and 7610 smartphones in order to monitor normality. The corresponding software behaviors, visualized as data results, can be found in Section 4.3.3.

Table 4.6: The top ten applications being used according to TNS in 2005, as seen in Chapter 2

No.	**Application**	**Usage**
1.	SMS	83%
2.	Games	61%
3.	Camera	49%
4.	MMS Pictures	46%
5.	PDA Functions	36%
6.	Internet	31%
7.	WAP	30%[7]
8.	Bluetooth	28%
9.	Email	27%
10.	Video Camera	27%

TNS GTI 2005 Study

As introduced in Chapter 2, the GTI 2005 bases on data coming from 6807 people aged 16 to 49, in 15 different countries. These respondents used a mobile phone (6517 persons), PDA, or laptop and accessed the Internet at least once a week. The study partly focused on the adaption of applications on mobile devices which we used to excerpt the top ten actions that were introduced in that work. These top ten actions base on the percentage of mobile phone users who used the corresponding application and can be seen on Table 4.6.

Testing Specification

In order to perform the actions, we had to specify testing scenarios where we had to distinguish between different use cases, for example a smartphone user can send *and* receive a SMS message of various size with various recipients. We identified about 40 use cases and specified a testing protocol for each. An example protocol is given on Table 4.7.

Technical Set Up

One of the used devices is a Nokia E61 smartphone which is running Symbian OS 9.1 and has a *QWERTY* keyboard. It supports most of the conventional techniques and protocols used in current smartphones, for example WCDMA and WLAN. A 64Mb storage card is plugged which allows storage of various files, like videos which then can be viewed on the 320×240 pixel display [160]. The installed Symbian-*C++* monitoring client was triggered for sending a vector of features every 20 seconds to our remote server with a public IP-address and attached database. This is done using a web service via UMTS-connection. The feature vector that was sent has a size of less than 8 Kbyte and contains about 50 features.

Since malware is only available for older Symbian Versions, we additionally used a Nokia 7610 running Symbian S60 version 7.x in order to monitor Symbian malware available from P2P networks. The display has a resolution of 176x208 pixels where the device has a weight of 118g. It uses an ordinary cell phone keyboard and includes a MP3 player.

[7]This will be substituted with MP3 (19%) due to UMTS usage and increasing interest for MP3 capabilities on devices.

Table 4.7: The test specification for a multi-player game called Miniblaster

Preconditions: • Miniblaster is installed on two devices • Bluetooth is disabled • settings in Miniblaster: • music/sound enabled • Two minutes of non-device-usage
Testing: 1. Launch Miniblaster on both devices 2. Start hosting on one device 3. Join game on second device 4. Play two rounds 5. Host exits game with left selection key 6. Second device confirms note and exits 7. Two minutes of non-device-usage
Expected Results: • Fall of *FREE RAM* • Raise of *CPU USAGE* • Bluetooth gets enabled • Data transfer

On Windows Mobile side, we used a HTX TyTN B smartphone with similar capabilities and configurations as the Nokia E61.

Results

On Figure 4.7 to Figure 4.15, the usage of most of the top ten applications can be seen: in each case, activity leads to detectable changes in the system. In Figure 4.7 the usage of the Short Message Service is shown. It is separated into four parts: sending empty message, writing and sending a 150-character message, writing and sending a 300-character message, and writing and sending a 150-character message with multiple recipients. In Figure 4.8 the usage of three different kinds of games is recorded: a simple game called Miniblaster, a more complex game named Sky Force Reloaded, and

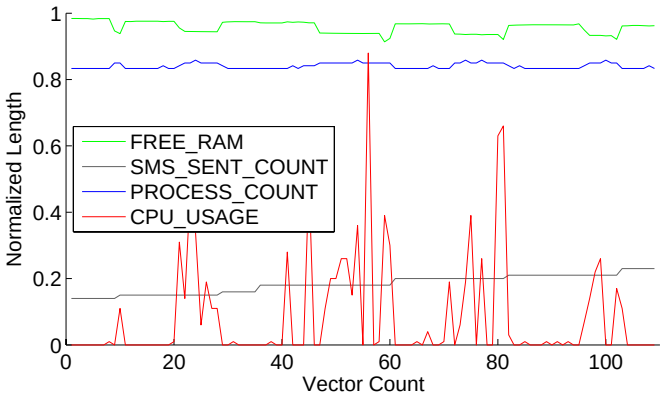


Figure 4.7: Monitoring results: When sending one or more text messages (grey bottom line), the system reacts to user activity. Especially the CPU indicates data processing triggered by the user.

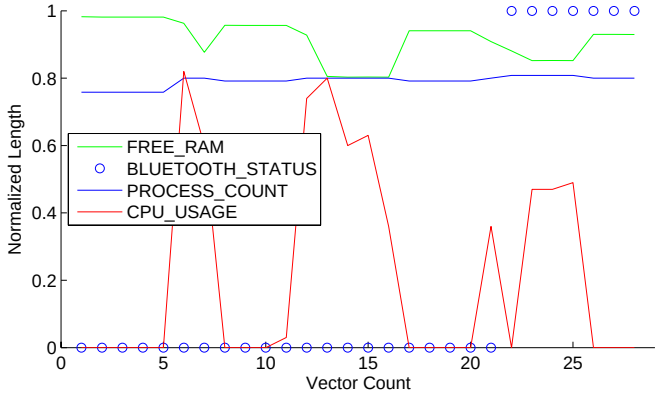


Figure 4.8: Monitoring results: Three different games were tested including a graphical game, a complex 3D game, and the first game in multi-player Bluetooth mode.

Miniblaster in multi player Bluetooth mode. Figure 4.9 visualizes sending an

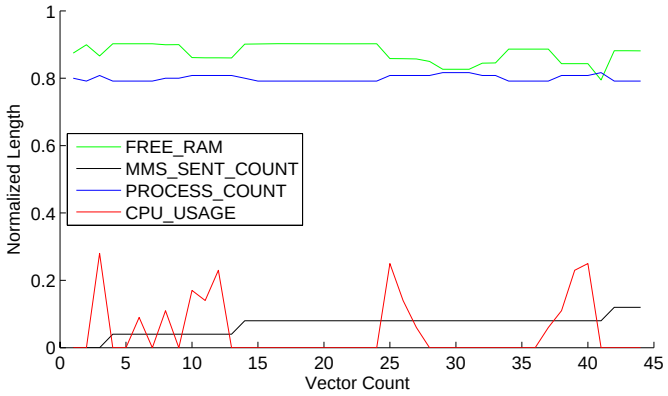


Figure 4.9: Monitoring results: The grey bottom line indicates the number of sent MMS messages. Depending on the content of the message, various processes are involved causing diverse monitoring results.

empty MMS message, writing and sending a 150-character MMS message, and writing and sending a MMS message with attached picture. Figure 4.10 represents the usage of PDA functionalities; in detail it is reading a .PDF file. Browsing the Internet can be seen on Figure 4.11 where different links were clicked and a picture was downloaded. Figure 4.12 refers to sending an image to a paired Bluetooth device. Figure 4.13 displays sending of various emails. On Figure 4.14, we used the browser to download a 8 Mbyte MP3 file which was played afterwards. Finally, Figure 4.15 represents the making of a new entry into the calendar.

What we can see although the number of vectors varies on the different figures is that each application affects the corresponding features in a different way, for example gaming produces much more CPU utilization than creating and sending MMS messages. This encourages the attempt to apply anomaly detection to the field of malware detection. A key issue that has to be solved will be the differentiation between software and malware with similar functionalities. The approach section follows the assumption that having only few features that are affected in a different ways might already enable detection algorithms to distinguish between malware and benign software.

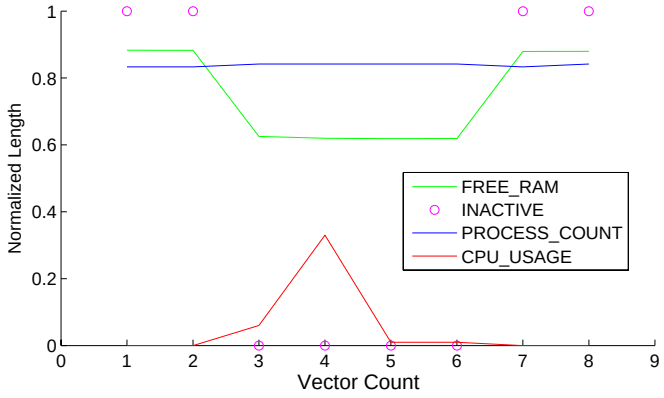


Figure 4.10: Monitoring results: This very simple example shows the possibilities a monitoring-based system can provide. In this case, we are opening .PDF file which leads to an increase of CPU usage and process count. In turn, the available memory decreases by the size of the PDF file plus the memory required for processing and visualization. When closed, the system returns to its initial state regarding the monitored features.

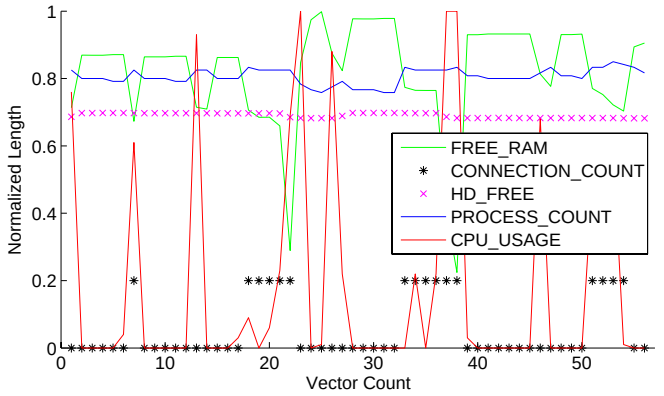


Figure 4.11: Monitoring results: In comparison to Figure 4.10, this chart has numerous entries. This is a good example why monitoring of devices can get very complex and anomaly detection using such data tends to generate false-positives.

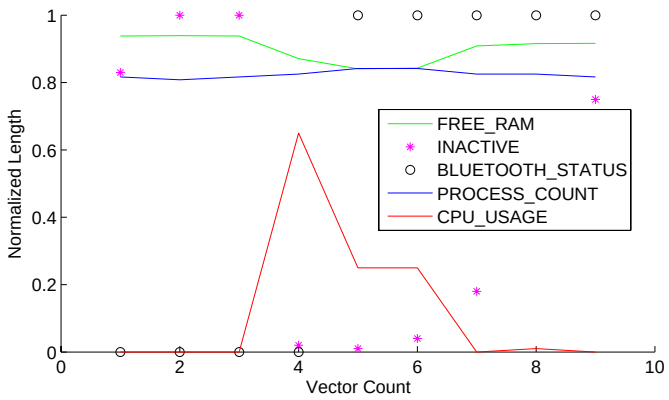


Figure 4.12: Monitoring results: Bluetooth data is transfered to a paired device.

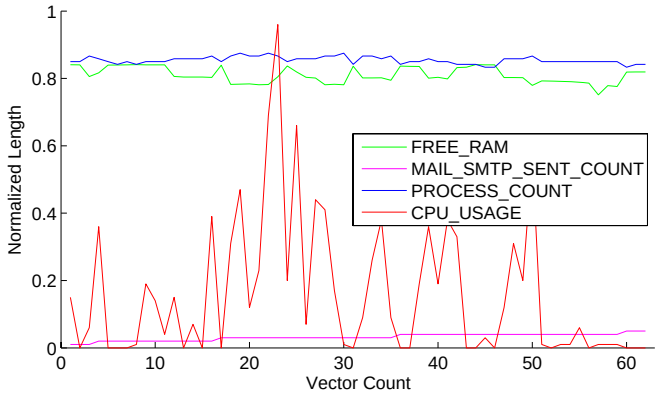


Figure 4.13: Monitoring results: Similar to sending SMS and MMS messages, sending emails affects the monitored smartphones. One observed difference is that writing and sending emails seems to be more resource exhaustive in terms of CPU and RAM usage.

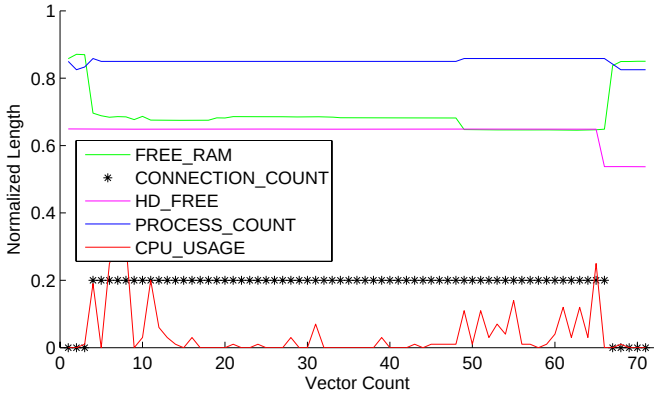


Figure 4.14: Monitoring results: Downloading and listening to MP3 file is another good example showing that every action taken has measurable impact on the system. After listening to the MP3, the system returns to its initial state except the amount of available hard disk space since the file was stored on the disk.

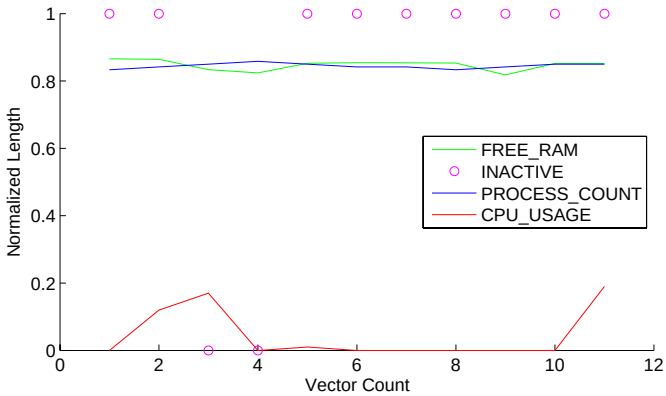


Figure 4.15: Monitoring results: When using calendar only minor changes can be observed. Distinguishing between services having a similar impact on the smartphone and mimicri-based malware can be seen as an open problem in this field.

Malware Monitoring Results We monitored several malwares on our Symbian devices. On the older Nokia 7610, we recorded malicious behavior of Blankfont.A, Hobbes.A, Cardblock.A, Mabtal.A, Fontal.A, and Dampig.A. Additionally, we created testing malware for the newer Symbian version S60 3rd. The monitoring results can be seen on Figure 4.16 to Figure 4.24.

On Figure 4.16, monitoring results of the Symbian OS malware *Blankfont.A*[8] can be seen. This malware replaces all icons and corresponding descriptions with blank field so that the usage of these gets more complicated. But except blank fields and icons, applications stay fully functional [65]. On Figure 4.17, the malware *Hobbes.A* is shown. *Hobbes.A* comes in a ma-

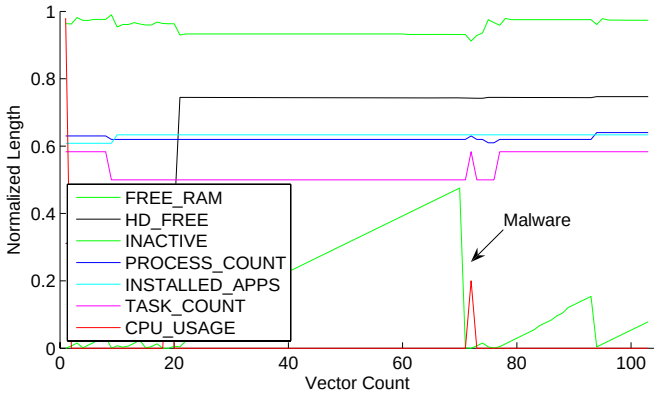


Figure 4.16: Malware monitoring: Blankfont.A replaces icons and descriptions of applications for irritating the user.

licious binary called `Symantec.sis`. This binary drops another binary into the system folder of Symbian os preventing the affected device to boot up properly. Post-infection booting will disable all smartphone functions except calling [69]. The malware *Cardblock.A* is shown on Figure 4.18. On execution, this malware sets a random password to the memory card. Additionally, it deletes system directories which destroys handle information for installed applications and for private data, like SMS a MMS messages or phone numbers [66]. On Figure 4.19, the malware *Mabtal.A* can be seen. *Mabtal.A* is a Trojan horse that drops other malwares onto the targeted system [70].

Figure 4.20 shows the monitoring results of the malware *Fontal.A*. This

[8] `http://www.f-secure.com/v-descs/blankfont_a.shtml`, visited 25.6.2009.

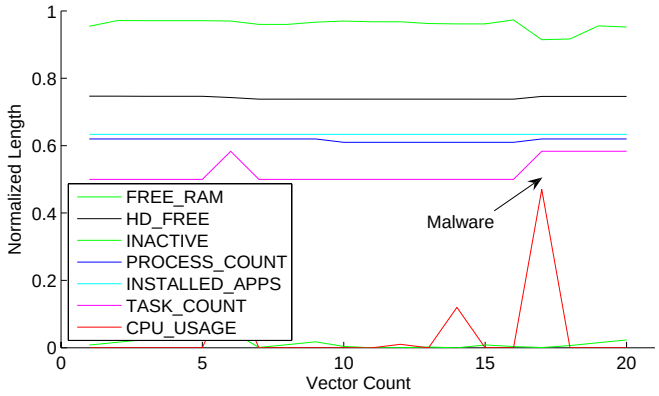


Figure 4.17: Malware monitoring: Hobbes.A drops a corrupted binary to the system preventing the system to be able to reboot.

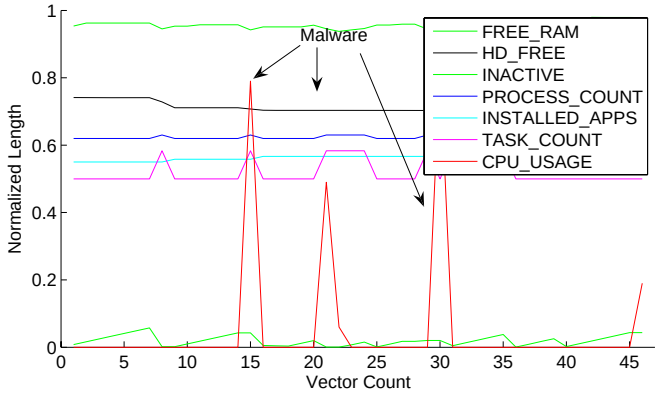


Figure 4.18: Malware monitoring: Cardblock.A sets a random password on the memory card and deletes system files.

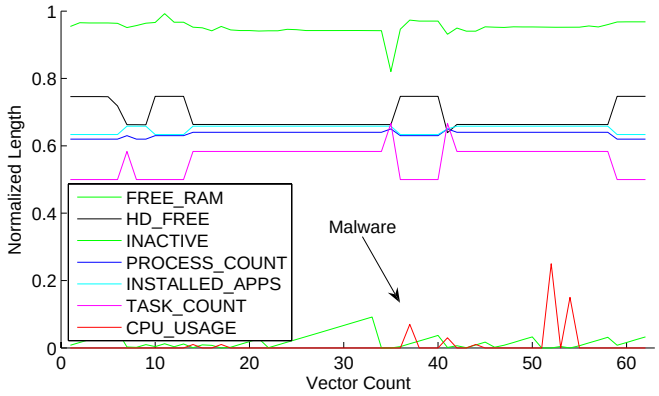


Figure 4.19: Malware monitoring: Mabtal.A drops malwares to the system.

malware copies a corrupted font file to the device while disabling the application manager. Disabling the application manager prevents the user from being able to install applications to the system. Additionally, this malware *bricks* the phone on reboot meaning that it cannot be used any more due to a dead lock in the booting process [68].

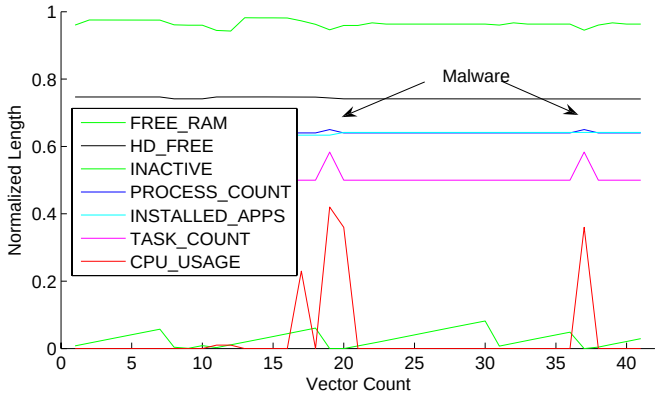


Figure 4.20: Malware monitoring: Fontal.A copies a corrupted font file to the system and disables the application manager.

On Figure 4.21, *Dampig.A* is monitored. *Dampig.A* disables applications and installs variants of the Cabir worm. Additionally, it disables the system file manager, messaging, phone book, and Bluetooth UI. Un-install information are corrupted for preventing the user to un-install the malware manually [67].

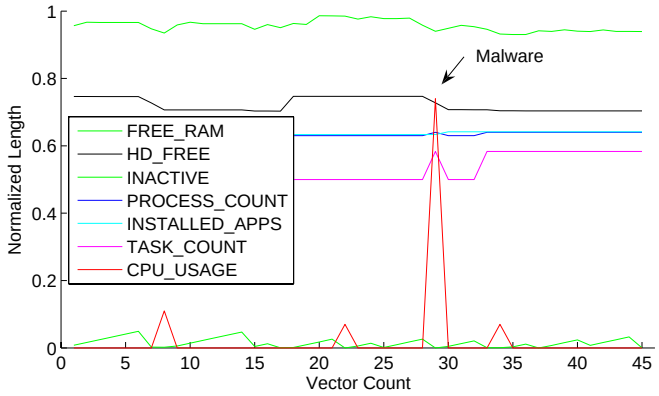


Figure 4.21: Malware monitoring: Dampig.A disables applications and system tools while dropping other malwares.

On Figure 4.22 the monitoring results of the malware described by [107] is shown. This simple and naive malware is capable of sending message to premium services. This malware was used in order to get a first impression on how the system can get affected due to malicious activity.

On Figure 4.23, a self-written testing malware was monitored. Extended from the basis of the Jamaluddin et al. [107] idea, we developed a Symbian OS malware capable of taking pictures and sending these via MMS to a predefined number in order to show that privacy-related attacks can be easily implemented. The corresponding picture-results can be seen on Figure 4.25 where we have to note that the users were not aware that a picture was taken[9]. Pressing on "2" triggered the malicious process.

Additionally, tests with manipulating the phone book via SMS commands succeeded too. Therefore we triggered a listener on the SMS inbox folder that only reacted on messages containing two leading "%" characters. Whenever such a message is received, the malware deletes the complete phone-book. Pseudo code for both malware can be found on Procedure 1 and Procedure 2 listings where monitoring results are shown on Figure 4.24.

In Figure 4.22, every time the SMS SENT COUNT increases an increase of processes and CPU busyness and a decrease of available RAM can be observed. At vector count 96 we determined that a Nokia E61 device can

[9]The users were informed afterwards. For privacy issues, pictures were chosen, that had an average quality.

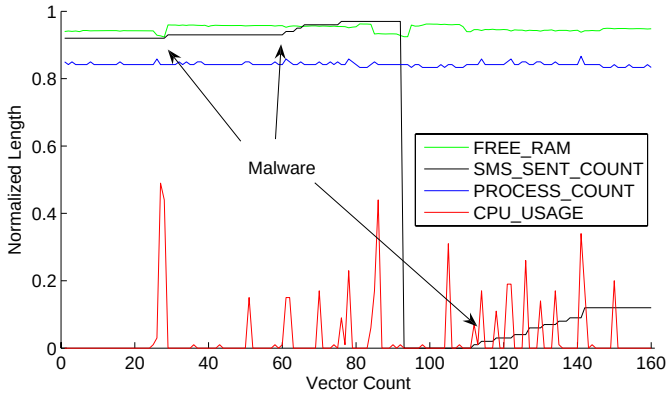


Figure 4.22: Malware monitoring: The Jamaluddin malware [107] shows which impact profit-motivated malware would have on a system. If badly written, user inactivity would be a good indicator to show that a malware is active. When hiding the malware activity within user activity, almost no difference can be observed between malware and user-intended messaging.

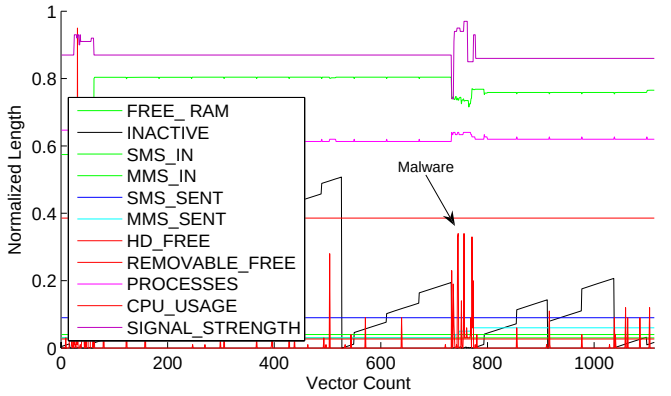


Figure 4.23: Malware monitoring: Interestingly, this testing malware realizing camera abuse showed us that high system activity caused by applications and malwares can also have impact on your radio transmission signal strength (top bar).

```
   Result: Takes and sends pictures
1  start KeyListener
2  if KeyEvent = "2" then
3  |  take picture;
4  |  add picture to MMS message;
5  |  send MMS message;
6  else
7  |  wait;
8  end
```

Procedure 1 Picture malware

```
   Result: Receives command SMS message and deletes contact list
1  start SMSListener
2  if IncomingSMS starts with "%%" then
3  |  prevent normal SMS handling;
4  |  delete contact list;
5  else
6  |  allow normal SMS handling;
7  end
```

Procedure 2 SMS malware

only hold 100 SMS messages which lead to the deletion of these. Additionally, we implemented two more testing malwares. The first malware takes a picture through the front camera, triggered by key strokes, and sends this via MMS to a predefined number. Example pictures can be seen on Figure 4.25. The second malware is remotely controlled by SMS messages. On receive of predefined content and strings, the malware deletes the complete phone-book.

4.3.4 Client-side Improvements

One objective was to find as many system characteristics as possible and necessary that can be usable for any remote anomaly detection system. After being able to retrieve 70 different features, describing the current state of a smartphone, the system characteristics were collected continuously over a long period of time. The resulting data was taken to evaluate common

[9]This class was removed since all values are already represented.

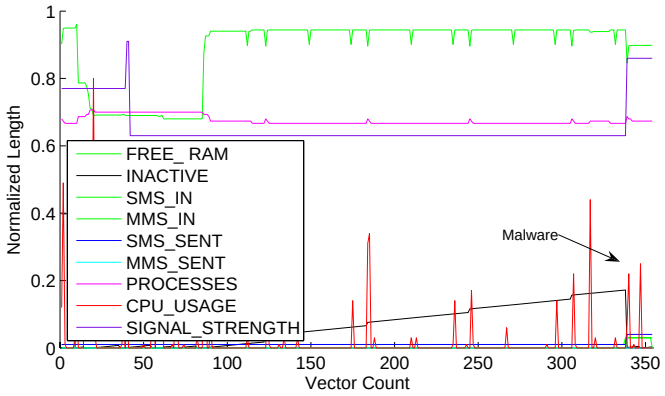


Figure 4.24: Malware monitoring: phone-book malware

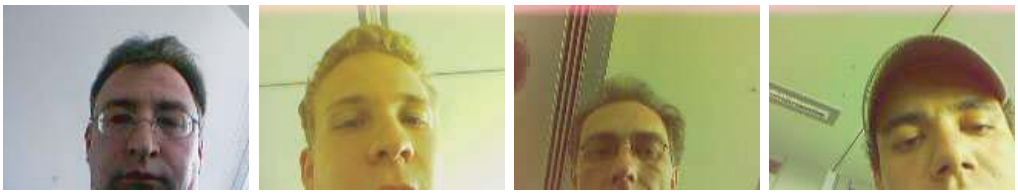


Figure 4.25: Pictures of smartphone users taken and sent by malware.

detection approaches. Furthermore, this data was analyzed in detail to detect conspicuous details helping us to reduce the number of observed features. These evaluations showed that 38 features can be ignored since they had no impact on application/malware detection.

The remaining 32 features were analyzed for finding redundancies that allow additional removal. This is necessary since processing large amounts of data causes high CPU usage and memory consumption which is a key issue for limited devices. Several methods for detecting redundant data are known from the field of machine learning. Because the Principal Component Analysis (PCA) has proven its usefulness [56], it was applied for this task. PCA is a method that is applied to multi-dimensional data in order to reduce the number of dimensions. This algorithm includes various mathematical steps starting with subtracting the respective mean from each existing dimension. Then the covariances are calculated and the corresponding eigenvectors and eigenvalues are determined. The features can be ranked by the calculated eigenvalue where the lower the eigenvalue is the less important it gets for the remote analysis.

Table 4.8: Principal component analysis results displaying Eigenvalue (EV) and Rank (R.)

EV	**R.**	**Feature 1**	**F 2**	**F 3**	**F 4**	**F 5**
0.6001	1	0.38 FREE_RAM	-0.377 DEBUG2	-0.377 TASK_CNT	-0.375 THR_CNT	-0.375 PROC_CNT
0.4414	2	0.544 BATTERY	+0.525 CON	+0.521 CON_DEL	-0.223 HD_FREE	+0.134 TASK_CNT
0.3519	3	-0.693 CELL_ID	-0.683 LOCATION	-0.17 USR_IDL	-0.095 HD_FREE	+0.084 DEBUG1
0.2749	4	-0.557 USR_IDL	+0.499 CPU_USG	-0.381 HD_FREE	-0.374 USR_IDL_B	+0.165 BATTERY
0.2029	5	-0.600 DEBUG1	+0.579 CPU_USG	+0.436 USR_IDL	-0.167 BATTERY	+0.154 HD_FREE
0.1413	6	0.678 DEBUG1	-0.526 USR_IDL_B	+0.373 USR_IDL	+0.29 CPU_USG	-0.119 BATTERY
0.0934	7	0.851 HD_FREE	-0.366 USR_IDL	+0.274 BATTERY	+0.191 CPU_USG	+0.08 DEBUG1
0.052	8	-0.733 USR_IDL_B	-0.517 CPU_USG	-0.383 DEBUG1	+0.164 HD_FREE	-0.064 THR_CNT
0.0159	9	-0.706 CELL_ID	+0.7 LOCATION	+0.062 USR_IDL	-0.045 USR_IDL_B	-0.043 DEBUG1

For automating these steps, several tools provide methods for performing such an analysis. For this work the Weka[11] tool was used to analyze a set of 3000 feature vectors which were recorded on a Nokia E61. This analysis identified nine relevant classes of features. These classes represent correlating features that have measurable impact on each other. Strong correlation means it is less important to look at all of them. Hence, we could reduce the number of features to one representative from each class which can be seen on Table 4.9. The detailed results can be found on Table 4.8.

Beside the identified features from the PCA we recommend to monitor additional features that are strongly related to smartphone malware. Having an eye on the number of installed applications can help to track down sources for anomalies. Whenever an anomaly appears as soon as an application is installed, it is probable that this anomaly was caused by the newly installed application. Since several malwares use Bluetooth and MMS in order to propagate, it makes sense watching the connections and incoming MMS messages. Additionally, monitoring outgoing messages can help to

[11] `http://www.cs.waikato.ac.nz/ml/weka/`, visited 25.6.2009.

Table 4.9: Ranked and recommended features are shown on this table. While automatic feature selection led to the first nine characteristics, the other five base on our personal experiences with smartphone malware and corresponding anomalies.

Rank	**Feature**	**Description**
1	FREE_RAM	Amount of available RAM
2	CON	Created TCP/IP connections
3	USR_IDL	User idle time in seconds
4	CPU_USG	CPU usage in percent
5	BATTERY	Battery charge level
6[10]	USR_IDL_B	Boolean user idle indicator
7	HD_FREE	Amount of available hard disk space
8	THR_CNT	Amount of running threads
9	CELL_ID	mobile phone network cell ID
10a	INST_APPS	Number of installed applications
11a	BT_CONN	Amount of opened Bluetooth connection
12a	SMS_SENT	Amount of sent SMS messages
13a	MMS_SENT	Amount of sent MMS messages
14a	MMS_RECV	Number of received MMS messages

track malicious programs sending local data to a Trojan horse master or to premium services in order to cause high costs. The additional features are marked with an "a" on Table 4.9 where a count of fourteen features was achieved in total.

The selected features were evaluated with a labeled monitoring data set in which the browser of the monitored device was started on few occasions. When having the objective to detect malware, detecting any running program is the first cornerstone. If it is not possible to detect anomalies caused by such a program, then detecting anomalies caused by malware is certainly not possible. Problems in detecting any program will otherwise result in a high false positive rate when detecting malware. The task of detecting any program is obviously not trivial, because other programs are running at the same time. For anomaly detection, the normal state was defined as the time before program to detect was started first. This normal data was used for training.

On the detection side we used algorithms basing on self-organizing maps (SOM) ([7], [116], and [183]), artificial immune system (AIS) ([141], [89],

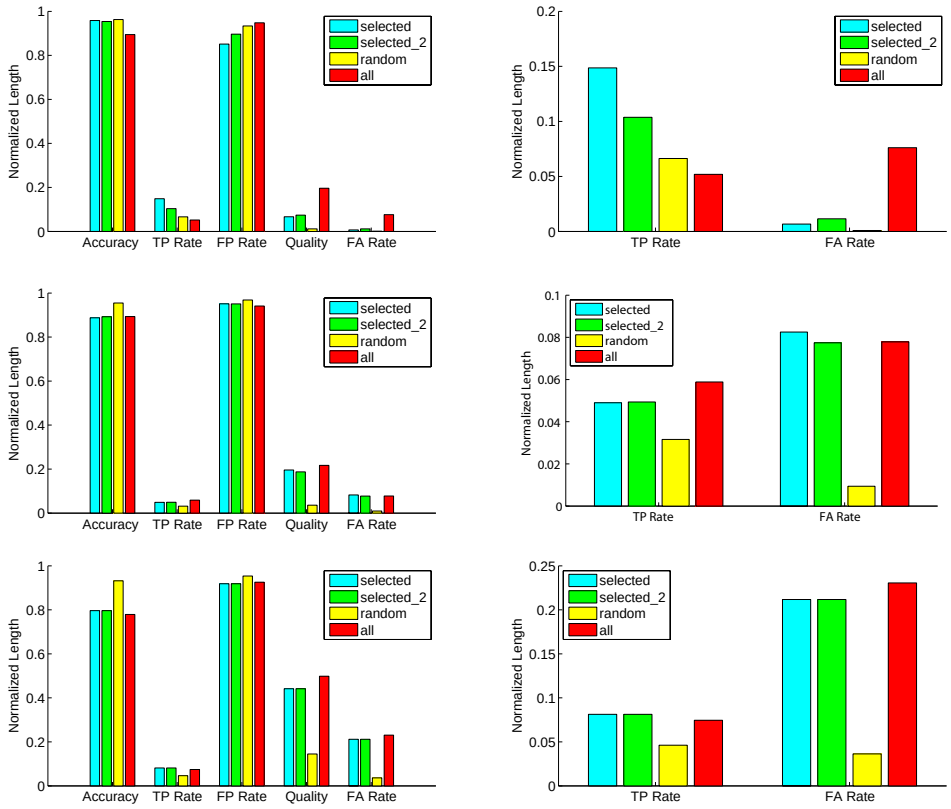


Figure 4.26: Detection results from top: Artificial Immune System with details on the right side, Self-organizing Map, and linear prediction.

and [79]), and an algorithm we called linear prediction in order to detect the browser activity. The linear prediction algorithm detects changes by checking four predecessors of a chosen feature. These four predecessors are used for estimating a probable successor. From the difference of this successor to the actual measured state, the anomaly value is concluded.

The accuracy, true positive rate, false positive rate, quality, and false alarms were evaluated. Especially the true positive and false alarm rate are of interest since they indicate how "good" the system performs; the true positive rate describes the rate of correctly identified incidents. The false alarm rate indicates the rate of falsely identified normal events. Figure 4.26 visualizes the results of four different feature sets. The first set was created by feature selection based on PCA (labeled as *selected*). The second set additionally includes our recommended features (labeled as *selected_2*).

For better assessing the impact of the feature selection two more sets were included. One set with all features (labeled as *all*) and one set of random features that were sized identically to the *selected* set.

It is obvious that different detection algorithms perform *different* but surprisingly the *selected* set caused a three times better detection of true positives than the complete set with the AIS algorithm. The recommended feature set resulted in a two times better true positive detection. Further investigations showed that the reason for this is that the more features are used in AIS the less precise its detection gets. Therefore, it can be stated that similar algorithms benefit from smaller feature sets where the results of the PCA work best. The SOM algorithm worked best with the complete feature set while while the SOM with our recommended set applied detected about one percent less true positives. Reducing the amount of features from 70 to 14 results in a save of 80% in terms of disk space. Additionally, computation and communication costs are reduced significantly which has a positive impact on the battery lifetime. Comparing these benefits with the loss of one percent in true positive detection, this is a deterioration that seems tolerable, especially in the field of mobile devices. The linear prediction algorithm works slightly better with the PCA and our recommended feature set than with the complete one. Therefore, similar simple approaches will benefit from a reduced set too.

4.4 An Architecture for Anomaly Detection on Android

In this section first results in creating an Intrusion Detection System for the Android platform are presented. Therefore, the corresponding architecture is shown in Section 4.4 and a more detailed description on the general detection system is given in Section 4.4.

Architecture

Figure 4.27 shows the architecture of the monitoring and detection client. The bottom-up view on it starts with the Linux operating system level generating signals received by the actual monitoring components. The Linux application level provides all the functionality needed for monitoring and

storing device and operating system information. On Java application level anomaly detection, detection collaboration, and detection response are realized where the corresponding states can be visualized in an user interface.

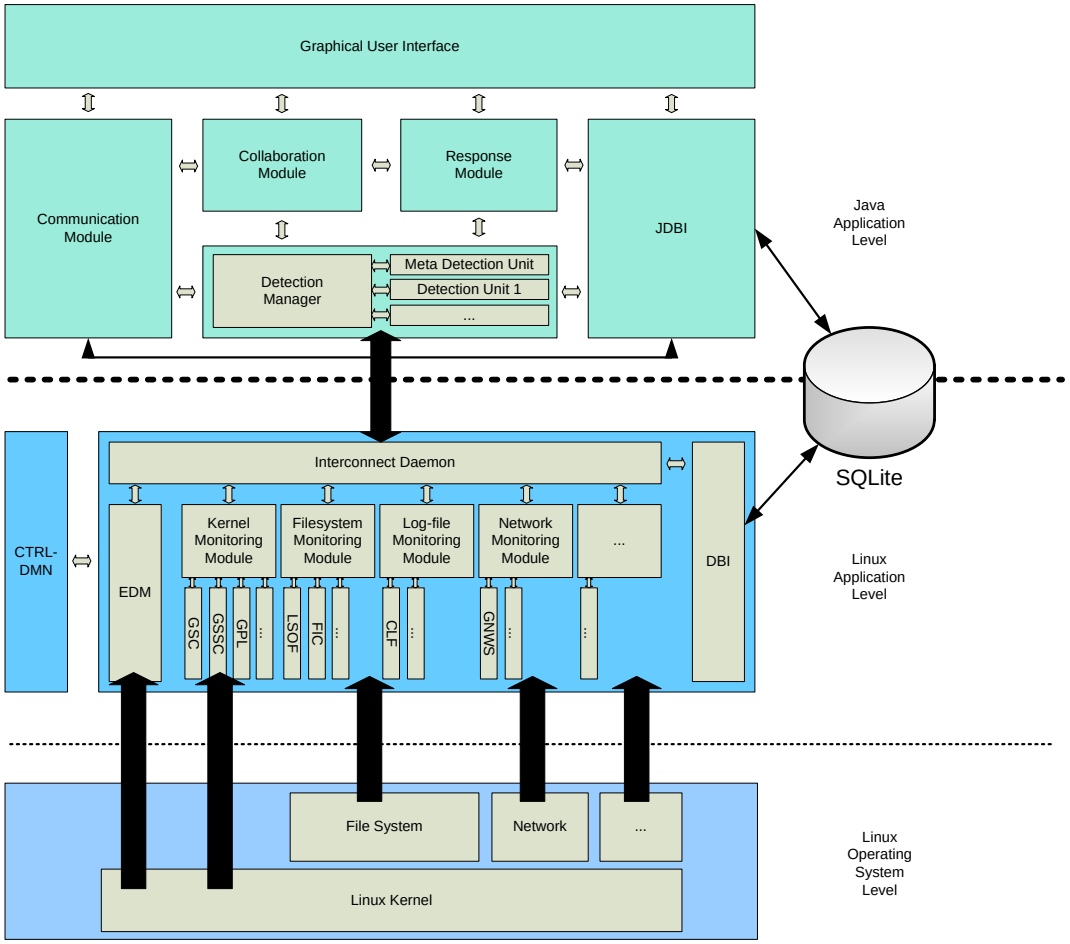


Figure 4.27: The monitoring and detection client architecture consist of a center part enabling monitoring of target devices and a top part that realizes detection, communication, and other relevant tasks.

Linux Operating System Level The Linux operating system level provides events that are recognized by the monitoring system. These events are initiated by kernel or file system changes.

Linux Application Level The monitoring architecture on Linux application layer consists of two programs: the monitoring application and the control daemon. The control daemon is responsible for checking the status and persistence of the monitoring application. The monitoring application extracts information (features) from the Linux kernel and file system. These features are used by the detection for creating a sense of normality. Therefore, the features contain information about the hardware and software states of the device. It has a generic and extensible design for modifying it to corresponding needs.

Interconnect Daemon This is the main module of the monitoring application. It is triggered and controlled by the event detection module for generating vectors containing features.

Event Detection Module (EDM) This is an essential component of the monitoring system. It recognizes changes in the kernel and file system and generates corresponding events, e.g. new process is started. Basing on these events, features are extracted that can vary in their content and size. Each feature is marked with a time stamp and event for later processing.

Kernel Monitoring Module This module extracts kernel-based features. Examples for this are process lists, system call traces, and symbol tables.

Filesystem Monitoring Module This module extracts and verifies information on files. Examples for this are a list of open files or an integrity check on predefined files.

Log-file Monitoring Module Since Android and many applications support logs, this module extracts information on changes and existence of these.

Network Monitoring Module This module can extract information on current network configurations, configuration changes, network status, and network traffic.

Database Interface (DBI) This interface provides access to the Android SQLite database from Linux application level. It is mainly used to store the feature vectors created by the event detection module.

Java Application Level The monitoring and detection architecture on Java application layer realizes several tasks for anomaly detection, detection collaboration, and detection response.

Detection Module The *detection module* runs light-weight detection algorithms based on feature vector excerpts from the database. It consists of a detection manager coordinating a variable amount of detection plug-ins. These plug-ins are instances of detection algorithm that on the one hand can analyze feature vectors and on the other hand can analyze results from different detection algorithms. Whenever cooperative detection algorithms are used, this module can additionally trigger the collaboration module.

Collaboration Module The *collaboration module* provides the means to enable detection as well as response in a collaborative manner as an API. Therefore, the collaboration module stores the node configuration of the device in a dedicated data model. Based on this model, *interests* for the collaboration can be defined that are matched against other node configurations. Thus, partners for the purpose of collaboration are found and communicated with via the *communication module*.

Response Module This module enables countermeasures to detected incidents.

Communication Module For exchanging feature vectors with the remote server or collaborative peers, this module provides suitable functions and network access.

Java Database Interface (JDBI) This interface provides access to the Android SQLite database from Java application level. It is mainly used to extract feature vectors and detection results recorded by the system.

Graphical User Interface This module visualizes current monitoring, detection, collaboration, and response status.

Detecting Anomalies

Approach An open system, like Android, requires protection against unwanted software and intrusion. In general, there are two techniques handling

this, namely misuse detection and anomaly detection. The former method is intended to recognize known signatures of malware and attacks, the latter to determine the degree of normality of some observables. Since there is no malware existent for the Android device, our focus is set on *anomaly detection*. Anomaly detection can be used to identify new and unknown attacks, which in turn can be used on- and off-line to generate signatures for fast detection in the future. Note that the detection architecture does not need to be changed for misuse detection.

The question arises what *normality* means. In our approach we distinguish an individual and a common sense of normality. Either are learned statistically and each device can check a system state according to both measures.

When constructing a detection mechanism for a mobile device such as Android, the computational costs has to be kept acceptable due to the limited resources and the need of energy saving. Thus, *battery efficiency* is a guide line for the architecture. Taking this into account, complex computational task and the storage of huge data sets is outsourced to an *external server* and the on-device detection algorithm is kept relatively simple. Since each detection requires energy, the system integrity should not be checked more often then really necessary, i.e. only on certain occasions. Hence, an *event-based* approach seems more reasonable than, e.g. a time-periodical one. Furthermore, neighbor devices are taken account in order to collaborate and exchange data in the existence of an ad-hoc network.

Detection Mechanism According to our approach five major tasks have to be handled:

1. Event detection, which is done by an *event sensor* (event detection module (EDM)).

2. System monitoring, to gain information about some system observables (features) when required. For each class of event there is an adequate monitoring module, recall Figure 4.27, the entirety of those we will call *system monitor*.

3. Detection, i.e. analyzing system features and assigning a status level, done by the detector, which consists of a *detection manager* and event-specific *detection units* and *meta detection units*.

4. Learning, which the external server is responsible for.

5. Collaboration, which is used in the absence of external server or for reducing the load from the external server.

Architecture Figure 4.28 outlines the architecture of the detection. The detection manager is a daemon, which can be implemented as an Android service. It is set on auto start and on the highest priority level. The system is prevented from stopping the detection manager via the method `setPersistent()`. In this way, it is assured that it runs permanently in the background. Normally, an activity should not be set persistent since then it blocks system capacities.

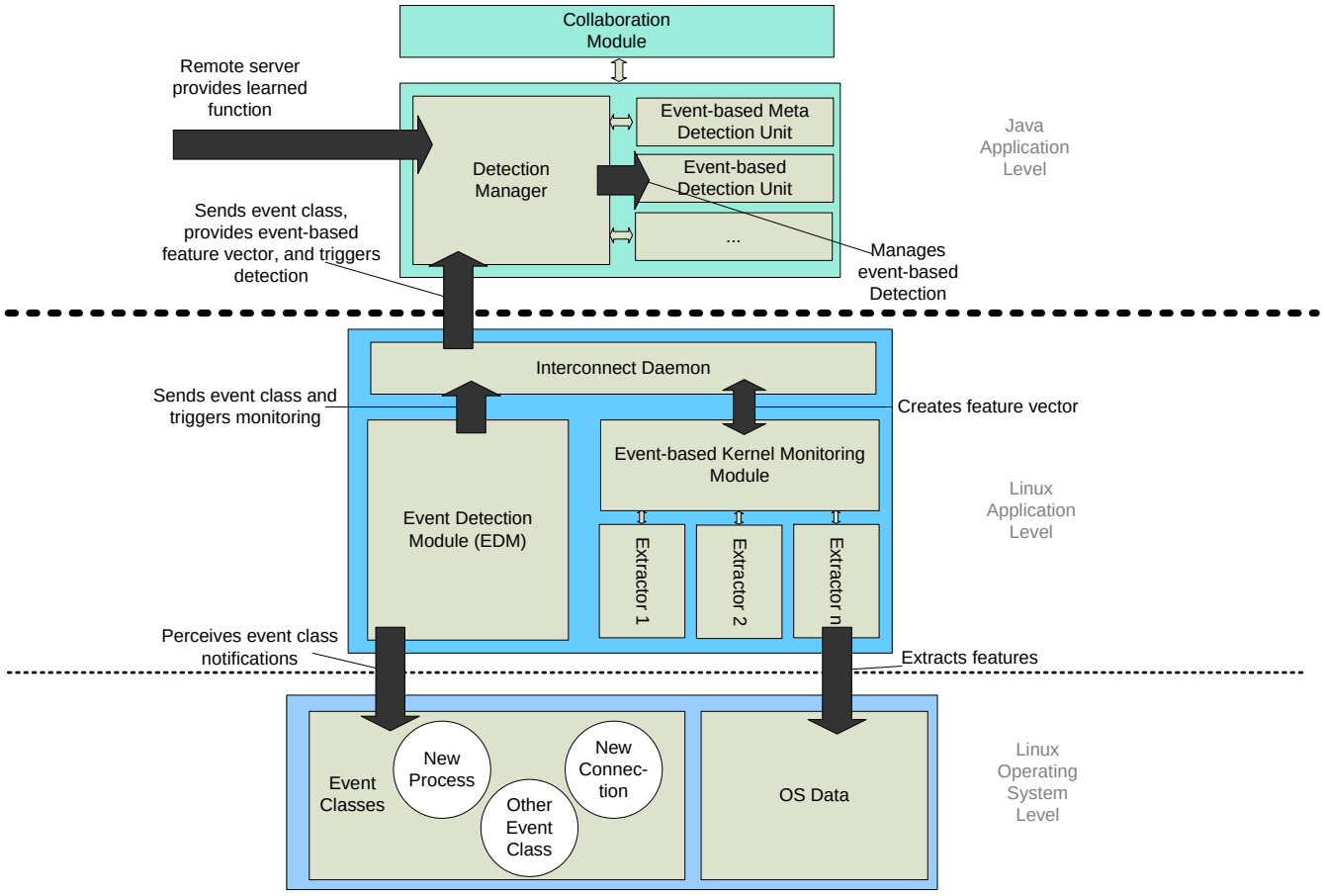


Figure 4.28: The architecture of detection mechanism consist of data extraction components on Linux-level and detection components on Java-level.

The jobs which have to be accomplished by this unit is receiving signals sent by the event detection module and starting a corresponding detection unit. The latter are implemented as sub-activities and assign to each feature vector a level of abnormality and return them to the detection manager. If it exceeds a predefined threshold, the detection manager will alert the user via GUI.

The external server does the computationally intensive work of statistical learning. The accumulated training data is sent from the database of a mobile device to this server, and in turn the server provides updated parameters for the detection to the mobile device. For a more detailed view on learning see 4.4.

Let the *interaction* of these units be described by an example. Assume that one of the events we described occurs, e.g. new process is being launched. This event is sensed by the event detection module, which informs the system monitor and the detection manager immediately about which kind of event has occurred. The system monitor then extracts some (event-specific) system features, in this case the sequence of system calls caused by this new process along with CPU/memory utilization and other process data. Meanwhile the detection manager has started an event-specific detection unit, i.e. the detection unit corresponding to the "process-started event". This detection unit evaluates then the level of alert from the feature vector provided by the system monitor.

Server-supported Learning Whatever reasonable learning technique is chosen, the computational costs for training cannot be carried by the mobile device in almost all cases. Hence, the training data, gained from monitoring, is gradually stored in a database and — after a certain amount of data has been accumulated — sent to a server, where the individual detection parameters are evaluated[12]. Training data is separated according to event class so that event specific detection parameters are determined and sent back to mobile afterwards. Each detection unit attains in this way an understanding of normal system behavior which follows each specific event. Furthermore, the server also calculates a common sense of normality based on the broad statistical data of all users and makes these common param-

[12]This approach opens discussions on the trade-off between sending data to a remote server which will also drain the battery and processing it on-device. For our case, our empirical results showed that using the approach of accumulating data for sending it in bigger chunks to a server for processing is the most efficient way to handle this.

eters available for the detection units of each user. The reason for that is that user behavior might switch abruptly if, e.g., a new application has been recently installed. Then the detection unit will state a high *individual* level of alert whereas it will claim that the system behavior is fairly normal relative to other users, since some of which might have already worked with this new application.

4.5 Tree-based Analysis for Malware Detection on Smartphones

The ubiquity of cloud computing gets more and more visible in our today's world. Most Internet users have been using cloud-based services for years, e.g. Google Mail[13], while new services are arriving with very short innovation cycles. Cloud computing can offer a tremendous amount of computational power that can be used for various purposes. Even complex games can be played *in the cloud*, like the third-person shooter game Quake 3. It was converted to QuakeLive[14], making the computational capabilities of the terminal computer almost unimportant[15]. This computational power can also be used for security-related purposes. Oberheide et al. [162] showed that overall results of anti-virus software can be improved when using cloud services. In their setup, Oberheide et al. constructed and deployed an in-cloud anti-virus system called *CloudAV* including several detection engines. Applying this kind of approach directly to low-cost computer systems, like netbooks or smartphones, is not feasible since these devices lack the appropriate amount of resources. Therefore, it makes sense to use cloud computing as remote service for low-cost devices relieving them from computational burden similar to [198]. The motivation for doing so is that Oberheide et al. [162] improved detection results by 35% using their cloud-based approach. This significant change has encouraged us to evaluate similar approaches for detecting malware on low-cost devices, e.g. smartphones. Additionally, although the approach of Oberheide et al. included behavioral-based detection, most engines used based on classic signatures implying a weakness against new and unknown malware. Hence, cloud-based machine learning could have a positive impact on this issue [151]

[13] `http://googlemail.com`, visited 3.5.2009.
[14] `http://quakelive.com`, visited 3.5.2009.
[15] Minimal requirements must be met in order to display graphics and play sounds.

since it is basically capable of detecting unknown and new threats.

Therefore, we present an approach basing on *dynamic analysis* for detection of malicious binaries on Linux-based smartphones basing on operating systems like Android, Maemo, MeeGo, Bada, and others. We want to make use of the computational capabilities provided by cloud computing while trying to keep processing effort on the devices low. *Dynamic analysis* for malware detection bases on acquiring processing data at runtime through monitoring activities of the observable. Due to this real-time monitoring, a lot of data can be generated making storing and computation of it exhaustive. More over, in our case we monitor system calls made by benign and malicious binaries which can lead to complex call sequences that need to be stored. For decreasing complexity, we simplify the data that is stored significantly: instead of storing complete sequences, we reduce the data to simple trees indicating *only* the frequency of calls being made by each binary. Similar approaches already worked well in [196, 194]. For processing these trees, we use tree kernels that can be used within Support Vector Machines (SVM). SVMs tend to be resource-exhaustive in comparison to traditional static approaches. Therefore, moving this computational work to a remote service hosted *in the cloud* will have significant impact on the battery life-time of a smartphone.

This section is structured as follows. In Section 4.5.1, we describe our approach on using simplified system call traces for classifying them using machine learning. Section 4.5.2 describes the requirements, set-up, and process of our experiments performed with real malware. Section 4.5.3 presents the results that have been achieved.

4.5.1 Approach

As shown more detailed in Section 4, dynamic analysis can be part of virus scanners and intrusion detection systems that protect host computers or networks. The key point about dynamic analysis is that data is acquired at runtime in comparison to static analysis which does not require executing binaries for investigating them. This can have the advantage that incidents are detected in real-time enabling the system to start appropriate countermeasures in time. In turn, dynamic analysis tends to be more resource exhaustive than, e.g. static analysis, since it processes mass data retrieved from real-time monitored instances. Considering this resource usage, apply-

ing exhaustive dynamic analysis on a mobile device, like a smartphone, does not seem feasible since it will drain the battery significantly due to higher resource usage. Therefore, formerly it only made sense to place this kind of analysis on stationary systems, like servers or stationary personal computers. With the evolution of early computer clusters to modern "cloud" which can provide tremendous computational power to connected devices, resource limitations lose importance.

Several approaches use system and function call traces for indicating malware or attacks [33, 117, 157, 157, 236, 142] which also partially include considering function call arguments. In our case, we also work with function and system calls where we try to simplify the approach by reducing call sequences to a general count of used calls. We try so since we had significant success with a similar approach applying static analysis to system and function call references by binaries in [196, 194]. By just considering simple call occurrences, we can decrease the computational burden of the remote cloud system which might seem strange just after arguing that clouds make resource limitations less and less important. But when considering that cloud cycles still need be paid or can be rented to costumers, being efficient in terms of computational complexity will save money in a practical scenario.

Similar to Wagner et al. [232] we use process trees for modeling binaries. Since process trees can get very complex, we just ignore interdependencies of nodes[16] and just count the number of call occurrences resulting in a tree having a depth of 1. Using solely these occurrences, we try to detect differences between benign and malicious binaries, represented as anomalies. Our aim is not to achieve a 100% detection rate, our intention is to provide a mechanism that might be used as pre-check that is capable of indicating malware but also can indicate the need for some more comprehensive checks being performed in the cloud. Therefore, our approach includes five steps that are performed within our analysis. This is depicted on Figure 4.29.

Interception of System and Library Calls

When trying to intercept system and library calls being made in the system, two main approaches should be considered:

- Patching the Linux-kernel to intercept system-calls as described in [232]

[16] In this case calls being made are meant.

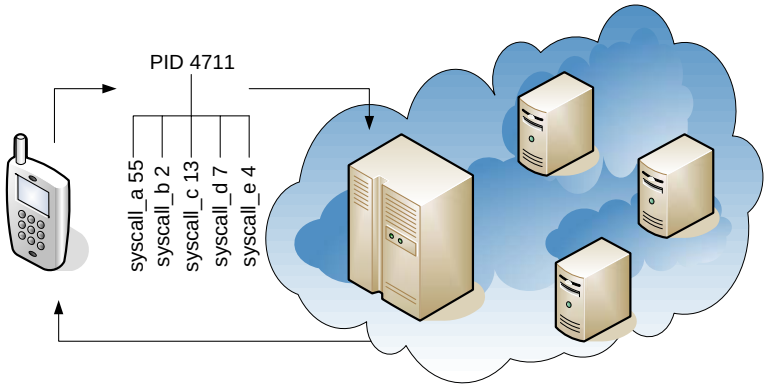


Figure 4.29: Steps taken in the detection approach: I) the smartphones traces binaries and generates tree models, II) the models are transfered to the cloud, III) the server learns the received models, IV) new tree models can be classified, and V) results are sent back

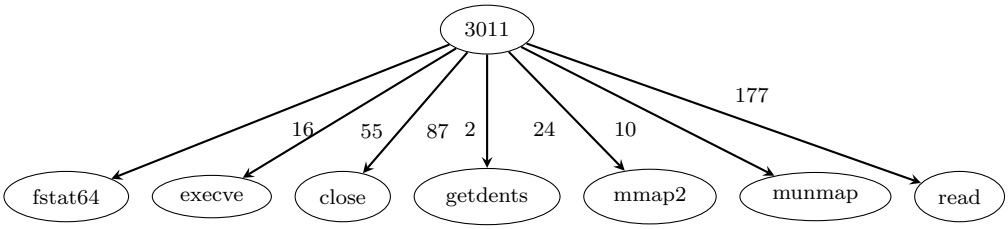


Figure 4.30: Excerpt from a tree showing occurrences of system calls made the command `lspci`. The root node shows the process ID of the traced binary. The edges show the number of system calls made, while the leafs carry the names of the calls themselves. `lspci` prints detailed information about PCI buses and devices in the system.

- Using strace/ltrace to gather system call information

Patching the Linux-kernel has the advantage that monitoring would only create a little overhead while additionally being harder to detect by attacker. The disadvantages of patching the Linux-kernel are that the code will operate in kernel-space. On failure, the system will probably crash. Patching the kernel will also result in maintenance problems: Linux-kernel is constantly changing, maintaining this patch across versions is a headache. Writing a loadable module instead is not a good option either since access to the syscall-table by kernel-modules is increasingly restricted. Any code, which

hacks around these restrictions will have similar maintenance problems as a kernel-patch.

When looking at the option to use the `strace/ltrace` commands, following advantages can be identified. First of all, these commands work across lots of different variants of Unix-based operating-systems making our approach applicable to different systems. Secondly, the output-parser works entirely in userspace and can be written in a modern high-level-language, such as Python or Java. This can result in much less effort needed to be spent on implementing. On the other hand, using `strace/ltrace` can result in a negative performance impact since due to synchronization's issues. Delaying the system and influencing the binary execution can result in a decreased accuracy where final tests still need to be performed by us. Another disadvantage of `strace/ltrace` is that malware can easily detect the usage of these commands. Some malwares prevent execution of most parts when they detect being traced of executed within a sandbox [91] influencing the results in turn.

Although having some disadvantages, we chose to use the `strace/-ltrace` commands in order to intercept system and library calls. A main reason for this is that we wanted to keep maintenance and programming effort low allowing us to verify our results in a continuous manner on various Linux/UNIX-based systems in future[17].

From a practical perspective, we use the following command arguments in order to intercept the system and library calls:

```
strace -f -r -o out /bin/executable
ltrace -f -r -o out /bin/executable
```

Both commands have the same output format so by writing a parser for this format, we can import `strace/ltrace` output into a database we set up.

Transformation of Intercepted Calls to Tree Models

As shown in Wagner et al. [232], complex trees can be used in order to use them for classification. In order to save resources, we try to simplify

[17]Another future alternative might be the ERESI framework from `http://www.dieresis-project.org/`. As soon this is ready for ARM, it may make sense to use the commands `e2dbg` or `etrace` to obtain system- and library-call information. But for now, using this framework in not possible.

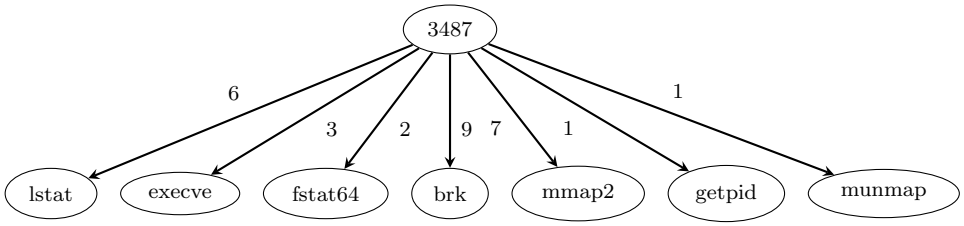


Figure 4.31: Excerpt from the tree resulting from the analysis of the binary infecter virus `42`

these trees significantly. We do so by ignoring any dependencies limiting the resulting tree to information describing the number of calls made for each system call. An excerpt from such trees can be seen on Figure 4.30 and Figure 4.31. This partial tree contains only seven leaves while the original one had 23[18]. We generated trees out of standard benign Linux applications as well as out of malware for being able to classify them with the means of machine learning.

Learning of Normality and Detection

We use Support Vector Machines (SVM) in order to evaluate our approach. In general, SVMs construct a hyperplane for separating instances that were processed. Therefore, SVMs represent a *classificators* that need labeled data for supervised learning. On a higher level, one can imagine an area with dots that gets separated by a line that the SVM constructs which also can be seen on Figure 4.32. In our case the dots would represent binaries which should get separated to benign and malicious software.

We use a tool called SVM-light from Joachims [148, 109] that implements the Support Vector Machine from Vapnik [228] applicable to pattern recognition and other problems. The optimization algorithms used in SVM-light can be found in [110, 109]. Additionally, the author states that the algorithm has scalable memory requirements and can handle problems with many thousands of support vectors efficiently.

For comparing the trees described in Section 4.5.1 we use a functionality provided by SVM-Light. SVM-Light can use tree kernels in order to measure similarity of trees. Practically speaking, the more the trees overlap in terms of number and kind of system calls made, the more sim-

[18]The tree was reduced for viewers convenience.

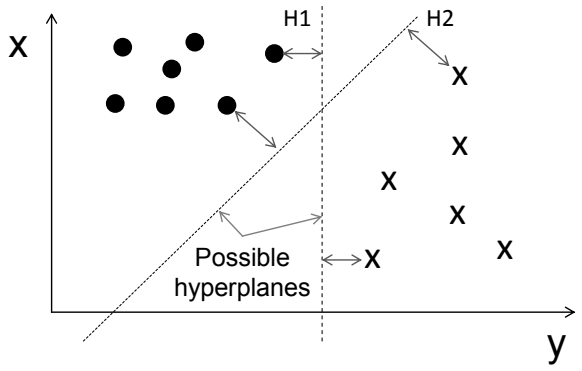


Figure 4.32: Simplified view on SVM showing hyperplane and training instances.

ilar they will be recognized. SVM-light reads input trees in the following format, where `-1` indicates the label for supervised learning, `|BT|` and `|ET|` indicate begin and end of a tree, and the braces structure the tree itself: `-1 |BT| (TREE (ARG0 (A1 NP)) (ARG1 (AM-NEG RB)) (ARG2 (rel fall)) (ARG3 (AM-TMP NNP)) (ARG4 (AM-TMP SBAR)) (ARG5 null) (ARG6 null)) |ET|`

4.5.2 Experiments

In our experiments, we recorded system call traces of about 1200 benign and 10 malicious binaries. These traces were used as input for the SVM. SVM-light can be called in two modes: in learning and classify mode. In learning mode, you can use various inputs for training your model. In classify mode, you can evaluate new samples within this model.

Environment and Malware Used for Experiments

Since Linux-Smartphone malware is practically not existent yet, we decided to run our tests on an ordinary Debian/Linux for our experiments. Basically, smartphone Linux systems do not differ much from stationary Linux system except that the ones for smartphones will be optimized for mobile use. Therefore, in most cases, smartphone Linux systems will only contain a subset of functions provided by the stationary ones plus some modifications

addressing mobility issues. We even suppose that detecting malware would be easier on smartphones because the *environment* that has to be considered is smaller. A good comparison for this is the amount of binaries that can be found in Android and Debian. Android contains about 100 binaries that are installed to the system. In Debian, we found about 1300.

We used a virtual machine basing on *VirtualBoxes*[19] which is easy to deploy. *VirtualBoxes* comes as an image that can be instantly used for several free and/or open-source operating systems, e.g. GNULinux or FreeNet/OpenBSD. Within this environment, we executed *real* malware which we fetched from *VX Heaven*[20]. The malwares we used for evaluating our approach target for Linux/BSD and are listed on Table 4.10. Generally speaking, it is pretty hard to find malware for Linux containing source code. Therefore, most of our malware set consisted of binary infectors that were available. Additionally, it is obvious that the amount of malware might cause problems when learning and classifying.

Table 4.10: Malwares used in experiments

Name	**Description**
Linux.FortyTwo	infects the host twice
Linux.Adhoca	infects a file in current directory
Linux.Adhocb	infects a file in current directory
Linux.Arches	ELF infector
Linux.Arian	compresses text areas within files
Linux.Csmall	binary infector
Linux.Egalite	FreeBSD binary infector
Linux.Futhorca	binary infector
Linux.Fothorcb	binary infector
Linux.fv	binary infector

Testing Process

For keeping bias low on our evaluation, we used *VirtualBoxes* in order to generate a clean snap shot of our testing system. This snap shot included a fresh installation of Debian OS including all required tools, add-ons, and malwares needed for our tests. We then loaded this snapshot, and executed

[19] `http://virtualboxes.or/`, visited 25.6.2009.
[20] `http://vx.netlux.or/`, visited 25.6.2009.

one single malware. On execution, we traced all system calls that were made and stored them in a database using some Python scripts we wrote. The data base was extracted and the original snapshot was reloaded. This procedure was repeated for each malware binary. After having created a database of system calls made by benign and malicious binaries, we used this data to train the SVM.

4.5.3 Results and Discussion

For our statistical investigation we performed various runs of different cross-validation, where in each loop execution the data is folded randomly into a training set containing more than 50% of the data and a test set containing the remaining percentage. Early results of our work-in-progress show unclear results that indicate problems with our data sets. When learning our model from major parts of the benign and malicious binaries, classification results show one to six miss-classified binaries. Considering the amount of binaries that have been checked, results seem pretty good but when checking the miss-classified files, it turned out that a high percentage of malwares caused these problems. After investigating this in detail, we found out that some of the malwares were too similar to benign binaries. This can have several reasons but the most obvious one is that the malware data set is too small and not representative. This resulted in making it very hard for the SVM to create a proper hyperplane that is capable of separating benign from malicious binaries. The problem is depicted on Figure 4.33. Another probable reason for these results might be that the information given through the simple trees is not sufficient. Using more complex trees may increase computation overhead but at the same time may improve analysis results.

Nevertheless, as shown in [194, 196], relying on calling frequencies of system calls works for detecting malware targeting smartphones. Therefore, we have a strong belief that improvements on our data sets will have positive impact on the results. Hence, one major task for our work will be to find a lot more Linux malware that can be used in our experiments. Without a significant increase of the amount of malwares, no statement can be made whether the presented approach works well or not. Another task will be the identification of additional machine learning approaches that might even be parallelized for improved results.

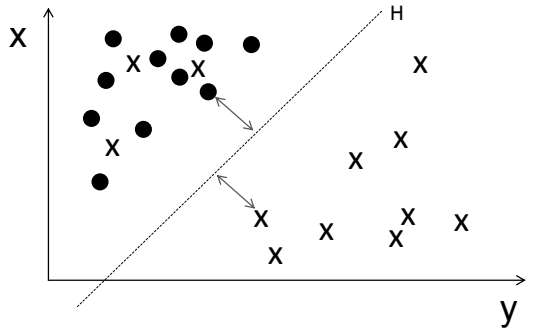


Figure 4.33: Trained data set resulting in miss-classification. Too many *malicious* instances are located next to benign ones.

4.6 Summary and Conclusion

In this chapter we presented our research on using monitoring for anomaly detection in order to detect malware on smartphones. Our experiments were conducted Symbian OS, Windows Mobile, and Android.

In Section 4.3, we demonstrated how a Windows Mobile and Symbian OS smartphone can be monitored in order to transmit feature vectors to a remote server. The gathered data is intended to be used for anomaly detection methods that analyze the data for distinguishing between normal and abnormal behavior. Abnormal behavior indicates malicious software activity. Furthermore, even unknown malware can be detected although no signatures are used. In our results we saw that most of the top ten applications preferred by mobile phone users affect the monitored features in different ways. This strengthens the approach of using monitoring and anomaly detection in order to detect malware on mobile devices.

In Section 4.4, we presented a general monitoring and detection architecture aiming for Linux-based Android platforms. This platform has several components dedicated to essential tasks, like data acquisition or detection, and will be our platform for future research. When running extensive analysis for detecting malware, such an architecture is needed for realizing malware detection on smartphones. One important factor of such a system is the monitoring capability. Malware detection will not work if the system does not get enough information to process. In terms of complexity this would be desirable of course, but a detection system will not be able

to identify malware basing e.g. on the amount of available RAM. Another important factor is the detection capability of the deployed algorithms.

Section 4.5 describes our approach on using the monitoring architecture from Section 4.4 for gathering runtime information from Linux-based smartphones. In particular, we traced the execution of binaries in order to extract the system calls being invoked. We use these calls to create simple call frequency trees that are trained and analyzed in a support vector machine. The interesting point about the call frequency trees is that they are not complex and base on data extracted in a static way. This makes them applicable to both, an online application store server checking submitted applications but also to on-device detection mechanisms. Since the data is extracted in a static way, devices basically are not threatened by an infection as they do not execute any code from the suspicious application.

Chapter 5

Malware Detection through Static Analysis

After having seen approaches towards dynamic analysis of malware in Chapter 4, static analysis will be focused in this chapter. Static analysis can be used to extend static pre-checks that are performed when developers upload software to online application markets that are available for most major smartphone platforms. Such extended pre-checks might increase malware detection capabilities preventing malware scattering through online application stores. Additionally, static analysis has the advantage that it is not bound to the runtime of binaries in order to work. It solely relies on the binaries themselves, which are investigated in a static manner. This means that it is suitable for on- and off-device detection.

Another advantage of static analysis is that certain approaches can be implemented using efficient and light-weight algorithms. This can turn valuable when detection mechanisms get directly deployed on the devices without increasing energy consumption significantly. Therefore, static analysis might be an appropriate measure to counter and even prevent malware from infecting smartphones. The contribution of this chapter is twofold.

First, static analysis on executables from the Android platform is performed in order to extract their function calls using the command *readelf*. Function call lists are compared with malware executables for classification with PART, Prism and Nearest Neighbor Algorithms, including an option to share results in a collaborative manner.

Second, clustering of function calls for static analysis is used. The

results are promising where the employed mechanism might find application at distribution channels, like online application stores. Additionally, it seems suitable for directly being used on smartphones for (pre-)checking installed applications.

5.1 Introduction

In this chapter, we focus on static, light-weight mechanisms for detecting malware presence on smartphones. Our static approaches for detecting malwares allow us to use simple classifiers and clustering methods which are not very resource consuming and, therefore, also fit well to mobile needs. Previous approaches [187, 197, 33] mostly rely on external servers for removing computational burden from the mobile device. In our case, the detection can benefit from a server but does not have to rely on it. Thus, for processing heavy-weight learning mechanism, we will benefit from the integration of a remote server.

These presented approaches are novel to the domain of smartphones and can extend third-party application checking for increased application security. Additionally, not only signing processes[1] can benefit from this approach: platforms mainly using online application stores can also employ this type of analysis for detecting malicious software, e.g. Android[2] or iPhone[3]. These online stores require the submission of the to be published application which is an appropriate time for applying the analysis. In some cases it is possible to bypass application stores for downloading software. Therefore, we also consider the option of moving these checks directly to the mobile devices.

The first approach, which is presented in Section 5.3, uses clustering of function calls in order to detect malware. Additionally, the approach employs collaboration for security to extend our malware detection results. Therefore, a set of entities is enabled to work on a common task without predefined roles in a heterarchical manner. The collaborative scheme is used to interact with other mobile devices in order to exchange detection data and system information. It can be considered as an operation mode whenever a mobile device is relying on the remote server but cannot access

[1] Such as known from Symbian OS.

[2] `http://code.google.com/android`, visited 28.6.2009.

[3] `http://www.apple.com/iphone/`, visited 28.6.2009.

it. Additionally, a second approach basing on simple decision trees for deciding the suspiciousness of the corresponding application is presented.

Our second approach, given in Section 5.4.1, uses a different method in order to detect malware. Basing on common clustering methods, we developed a light-weight algorithm called *centroid machine*. This algorithm is used in particular for detecting Symbian OS malware on the basis of function calls which suffices the requirements of mobile devices, e.g. efficiency, speed and limited resource usage. The results of the *centroid machine* are compared with the results of the light-weight naive Bayes classifier [184] as well as with a heavy-weight support vector machine method. This approach is not limited to Symbian OS, but since this platform has the highest amount of available malware among smartphone systems, we chose it for validation purpose.

5.2 Related Work

Moser *et al.* [149] present a binary obfuscation scheme that relies on the idea of opaque constants, which are primitives to load a constant into a register. From that register an analysis tool cannot determine its value in order to obscure program control flow, disguise access to local and global variables, and interrupt tracking of values held in processor registers. Using their proposed obfuscation approach, they show that advanced semantics-based malware detectors can be evaded and static analysis techniques are not sufficient alone to identify malware; thus, they need to be complemented by dynamic analysis.

Several publications were made in the field of smartphone malware detection and smartphone intrusion detection systems where tendencies can be seen that most promising approaches involve power usage data in order to detect attacks [158, 114, 105, 34]. Other approaches used feature vector- and signature-based techniques in order to detect malware or anomalies [229, 41, 187, 144, 33, 198, 197]. By now, no function call-based approaches for smartphones are known to the authors.

Venugopal *et al.* [229] outline the considerations for malware detection on mobile devices. They propose a signature-based malware detection method that is well suited for use in mobile device scanning due to its low memory requirements and high scanning speed.

Egele et al. [59] describe a static analysis for PHP web applications, checking the requests while considering the call parameters for creating more precise detection models.

Christodorescu et al. [44] use static analysis for creating assembler-based program automatons for detecting malicious activity using the disassembler IDA Pro[4]. In particular, they address the problem of obfuscation which current commercial anti-malware application still can not handle properly. Their tool is called *static analyzer for executables (SAFE)* and seems to be an appropriate approach for handling malware through a stationary system. The drawback is that on-device detection requires lightweight methods complicating a possible transfer of the presented approach to a limited mobile device.

Bergeron et al. [28] perform a semantic analysis of binary code. Their approach is separated into three stages: (1.) creation of an intermediate representation, (2.) flow-based behavior analysis, and (3.) static verification of critical behaviors against security policies. Flow-based analysis is a valuable technique for investigating malware but currently not suitable for smartphones due to resource constraints.

Krügel et al. [118] use static analysis on binaries in order to detect kernel-level rootkits via instruction sequences before corresponding modules get loaded into kernel. They state that their prototype did not produce any false-positive while detecting all tested rootkits. Additionally, the authors refer to a problem caused by "the exponential explosion of possible paths that need to be followed" which clearly indicates that currently, this approach cannot be applied to smartphones. These paths are created by creating states of the observed machine for analysis of control flows.

Provos [172] wrote a tool capable of generating and enforcing policies concerning system calls. The "Systrace" tool is intended to be efficient and does not impose significant performance penalties while currently aiming for stationary Linux/Unix systems.

Warrender et al. [236] compare sequences of system calls in order to distinguish normal from abnormal behavior. They test four methods with increasing complexity, e.g. Hidden Markov Models (HMM) and come to the conclusion that although HMM achieves the best accuracy, the less complex ones are sufficient. The problem remains that analyzing call sequences is a complex task currently not suitable for smartphones.

[4] `http://www.hex-rays.com/idapro/`, visited 28.6.2009.

We present a method of static analysis of executables by disassembly. Essential characteristics like system and library functions are extracted and build the basis for identifying malware. Such identification is done by a classifier, which is implemented using machine learning algorithms. Static analysis of executables is a well explored technique, recall for instance Christodorescu and Jha [44] or Zhang and Reeves [243] who propose such analysis to establish a similarity measure between two executables in order to identify metamorphic malware. Kruegel *et al.* describe static disassembly in [119]. Wang, Wu and Hsieh [235] present data mining methods to discriminate between benign executables and viruses, whose dynamically linked libraries and application programming interfaces are statically extracted. Support vector machines are used for feature extraction, training and classification. Eskin *et al.* [63] apply machine learning methods on a data set of malicious executables. Based on their data set they empirically show that the rule inducer RIPPER and naive Bayes estimators outperform simple signature-based scanner.

Mutz *et al.* [157] state that common system call-based approaches do not consider call arguments. This enables attackers to create methods for evading detection. They propose two primary improvements upon existing system. The first improvement applies multiple detection models to system call arguments. This enables them to analyze the system call arguments from various perspectives. The second improvement describes a sophisticated method for aggregating the results from all applied detection models. This method bases on Bayesian networks for classifying and improves detection accuracy and resilience against evasion attempts.

Liu *et al.* [134] propose a finite state automaton extended by call stack information for effectively capturing the control flow of programs. They use static analysis for creating a base model and add dynamic learning on call sequences. Detection results show that this approach has higher detection capabilities than static approaches alone. Additionally, the system has a lower false positive rate than models created by dynamic learning alone.

Wagner *et al.* [233] present an approach extracting non-deterministic finite automaton (NDFA) models from application source code. Then, the corresponding system calls of that application are traced for compliance to the created model at runtime. The presented results are still preliminary where the monitoring overhead for detecting attacks is high.

Different from these publications, the use of Android allows us to mod-

ify the system even at kernel-level. Therefore, up to our knowledge, this is the first time that a light-weight on-device function call analysis is investigated for smartphones.

We present a method of static analysis of executables by disassembly. Essential characteristics like system and library functions are extracted and form the basis for identifying malware. Identification is done by machine learning classifiers. Static analysis of executables is a well explored technique. Zhang and Reeves [243] propose a static analysis to establish a similarity measure between two executables in order to identify metamorphic malware. Kruegel *et al.* describe static disassembly in [119]. Wang, Wu and Hsieh [235] present data mining methods to discriminate between benign executables and viruses, whose dynamically linked libraries and application programming interfaces are statically extracted. They use support vector machines for feature extraction, training, and classification. Eskin *et al.* [63] apply machine learning methods on a data set of malicious executables.

Ad-Hoc networks can be considered as the enabling technology for the realization of collaborative intrusion detection among Android devices. In that scope, new challenges arise from the inherent dynamic characteristics of these networks.

Zhang *et al.* [244] mention that intrusion detection in mobile computing environment may benefit from distributed and cooperative approaches. In this regard, they propose to use anomaly detection models constructed using information available from the routing protocols. Huang *et al.* [12] present a cluster-based detection approach for intrusion detection system and showed that they could maintain the same level of detection performance as an original per-node detection scheme with less host CPU utilization. Sterne *et al.* [209] propose a generalized, cooperative intrusion detection architecture with dynamic topology and *clusterheads*. These clusterheads are determined according to valuable characteristics, e.g. distance, bandwidth etc. and they perform special tasks like aggregation and analysis of monitoring results. A general overview of intrusion detection in Ad-Hoc Networks is given in [216].

All these approaches target in special security concerns arising in Ad-Hoc networks, whereas our approach is striving for the opportunities Ad-Hoc networks offer. In this context, Bye *et al.* [36] present an overlay framework including an algorithm to find common groups and exchange security related data, e.g. monitoring results.

5.3 Static Analysis of Executables for Collaborative Malware Detection on Android

This section, describing our respective first approach on static analysis, is structured as follows. Section 5.3.1 describes how the data for our approach is collected. Section 5.3.2 presents our detection approach. Results are used for collaboration scenario in Section 5.3.4. Finally, we discuss the results in Section 5.3.4.

5.3.1 System and Function Call Analysis on Android

The overall system realizes a client-server architecture which can be seen on Figure 5.1. It basically provides three main functionalities: On-device analysis, Collaboration, and Remote analysis. The client gathers data for supporting these functionalities. For improving detection, data can be exchanged between two mobile clients in a collaborative manner. This data can consist, e.g. of detection results or anomalous feature vectors. Whenever on-device detection is not feasible, the client can send data to the remote server. In turn, the server can send detection results back to the client. Additionally, it can send commands for reconfiguring the client.

Data Extraction Architecture

The Android Java framework, as of time writing, only offers a restricted set of Java methods in order to access the underlying OS-level, e.g. it is not possible to get a list of all running system processes. In order to extract further information, a mediator is required that collects the desired data on OS-level and delivers it to an upper lying software stack. Responsible for this task is a self-written tool called *Interconnect Daemon*, a Linux server daemon which consists of several modules, e.g. system monitors. Additional module tasks are scanning the file system, creating hashes from important files, or waiting for operating system signals to indicate events.

The various modules work on top of Android's system binaries, mostly supported via *toolbox*, an all-in-one statically compiled binary. *Toolbox* offers a number of standard Linux system commands with a limited set of

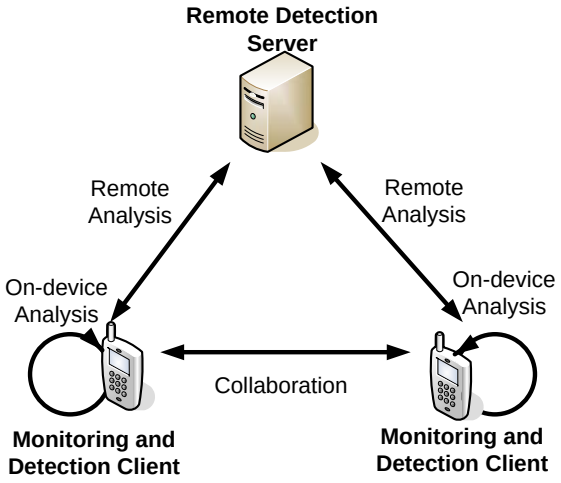


Figure 5.1: The overall system architecture includes a monitoring and detection client installed on a smartphone and a remote server that processes received data. Clients can share detection devices for improving detection quality.

parameters. Additional tools were added: busybox[5] supports a far greater number of Linux commands with appropriate parameters; strace[6] offers debugging and system call tracing capabilities. Further descriptions can be found in [195].

Creating a Training Set With Readelf

For this approach, a specific module within the *Interconnect Daemon* was responsible for identifying and extracting all Linux system executables, to be precise, all ELF (Executable and Linking Format) object files (excluding shared libraries). These executables (mostly in */bin*) hold static information which can be read out with the appropriate reader, in our case *readelf*. Most interestingly, the *readelf* outputs the static list of referenced function calls for each system command. The following example shows the first lines of the output of readelf running on a system command (/bin/ls):

```
Symbol table '.dynsym' contains 104 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
```

[5] `http://www.busybox.net/`, visited 7.7.2009.
[6] `http://sourceforge.net/projects/strace/`, visited 7.7.2009.

```
 0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
 1: 00000000   622 FUNC    GLOBAL DEFAULT  UND abort@GLIBC_2.0 (2)
 2: 00000000    29 FUNC    GLOBAL DEFAULT  UND __errno_location@GLIBC_2.0 (2)
 3: 00000000    84 FUNC    GLOBAL DEFAULT  UND sigemptyset@GLIBC_2.0 (2)
 4: 00000000    52 FUNC    GLOBAL DEFAULT  UND sprintf@GLIBC_2.0 (2)
 5: 00000000   433 FUNC    GLOBAL DEFAULT  UND localeconv@GLIBC_2.2 (3)
 6: 00000000    10 FUNC    GLOBAL DEFAULT  UND dirfd@GLIBC_2.0 (2)
 7: 00000000    87 FUNC    GLOBAL DEFAULT  UND __cxa_atexit@GLIBC_2.1.3 (4)
[...]
```

We identified a number of Linux system commands within Google Android (less than 100). After extracting those, inspecting them with *readelf*, and extracting the lists of function calls, this data formed our benign training set. In order to build a set of malicious training examples, we selected approximately 240 different malwares, found via Google Search, and extracted the static lists of function calls with the same method as described above. The malware set consisted of virus, worms, and Trojans specifically designed for Linux (not specifically designed for Android's ARM-architecture). A few malwares have been successfully compiled for ARM-architecture and compared with its i386-counterpart. The results showed only very minor differences leading us to the conclusion that using this set as preliminary malicious training set was a valid approach. The combination of both benign and malicious data set formed our final training set which has been used for further analysis.

5.3.2 Classification of Executables through Static Analysis

The executables can be fairly well identified as normal and malicious by simply looking at the names of the functions and calls appearing at the output of *readelf*. In the sequel, we will call these names simply *attributes*, which are grouped in *relocation* and *dynamic* attributes due to their appearance at the readelf output. The *combined* attribute set is an union of the relocation and dynamic attribute set. The set of attributes is further split: an attribute is in the set of *mutual attributes* if there is at least one malware ELF and at least one normal ELF whose readelf output contains it, whereas an attribute is in the set of *all attributes* if it is contained in the readelf output for at least one ELF, no matter if malicious or normal. Eventually, six attribute classes are gained by the just mentioned discrimination, the sizes of which are presented in table 5.1. An attribute class will

be denoted by ℵ. The attribute class which is, for instance, both dynamic and mutual have the shape ℵ = { abort, _errno_location, sigemptyset, ... }.

Table 5.1: Sizes of the attribute classes

	relocation	dynamic	combined
mutual attributes	174	145	189
all attributes	1662	2284	2816

The question arises whether these attribute sets have the potential to distinguish normal from malicious executables. By applying several state-of-the-art classifiers, it turned out this is the case for most of them. The table below indicates accuracy parameters, i.e. correctly classified instances rate (CC), detection rate (DR), and false positive rate (FP), for each attribute set and each applied classifier due to our data set. To check the generalizing ability of the trained classifiers, stratified ten fold cross validation is used, where each fold is constructed randomly. The data mining package *weka*[7] served as test environment.

Table 5.2: Accuracy values of classifiers according to attribute sets

	relocation			dynamic			combined		
	mutual attributes								
Accur.	CC	DR	FP	CC	DR	FP	CC	DR	FP
Prism	0.78	0.70	0.00	n.V.	n.V	n.V.	0.78	0.70	0.00
PART	0.94	0.99	0.15	0.97	1.00	0.12	0.97	1.00	0.12
n. Nb	0.92	0.98	0.21	0.90	0.92	0.13	0.96	0.98	0.11
	all attributes								
Prism	0.81	0.76	0.00	0.83	0.76	0.00	0.83	0.77	0.00
PART	0.95	1.00	0.16	0.97	1.00	0.12	0.97	1.00	0.12
nNb	0.94	0.99	0.12	0.96	0.99	0.10	0.96	0.99	0.10

Three classifiers of different kinds are applied to our data. The classifier *PART* extracts decision rules from the decision tree learner C4.5 [76]. *Prism* is a simple rule inducer which covers the whole set by pure rules [38]. Both take the interdependencies of attributes into account and are – once learned – efficient classifiers. The computational costs of learning could be shifted to a server, then mobile devices will be provided by rules. Prism produces in all cases we tested no false positive whereas it performs less well in detecting malware, and a higher set of rules (from 10 to 30) are usually induced than

[7] `http://www.cs.waikato.ac.nz/ml/weka/`, visited 7.7.2009.

with PART, which is satisfied with 2 to 12 rules. Our third classifier is the *nearest Neighbor* algorithm (nNb). We used the following light-weight version of this standard classifier: Let $\mathbb{M}$, $\mathbb{N}$ be the sets of malicious and normal ELFs respectively, and let $\aleph$ be an attribute set and ρ a metric on $\{0,1\}^{|\aleph|}$. An ELF is represented by $x = (x_i)_{i\in\aleph}$ where a component x_i is equal to 1 if this ELF has attribute i and equal to 0 if not. Here the simple metric

$$\rho(x,y) = \sum_{i\in\aleph} |x_i - y_i|, \qquad \text{for } x,y \in \{0,1\}^{|\aleph|}.$$

is applied. By $d(x,K) = \inf_{k\in K} \rho(x,k)$ the distance of x to a subset $K \subset \{0,1\}^{|\aleph|}$ is denoted. The classifier φ maps a formatted readelf output of an ELF x to the state space {malicious, normal},

$$\varphi(x) = \begin{cases} \text{malicious}, & \text{if } d(x,\mathbb{M}) < d(x,\mathbb{N}), \\ \text{normal}, & \text{else.} \end{cases} \tag{5.1}$$

The computational complexity of the detection by φ is of acceptable order, namely $O(|\aleph| \cdot (|\mathbb{M}| + |\mathbb{N}|))$. Nearest neighbor detection has the advantage that no server is required for training and that it behaved in our test stable w.r.t. thinning of the attribute set: if the most distinctive attributes due to our rules are omitted, the accuracy parameters of nNb do not vary significantly. A drawback is that the attributes of each single malicious or normal ELF have to be stored, which is acceptable in our case but might become inconvenient with a growing data basis.

The detection map in (5.1) makes binary decisions whereas it delivers no statement on the certainty of judgment. A simple solution is the strictly increasing function $\tilde{\varphi} : \{0,1\}^{|\aleph|} \to [0,1]$ which is derived from Equation (5.1) by affine linear transformation, where an output of 0 means that investigated ELF is normal and 1 that the ELF is malicious with probability one. The values in between represent the level of maliciousness. In

$$\tilde{\varphi}(x) = \begin{cases} \frac{1}{2} r(x), & \text{if } r(x) \in [0,1], \\ \frac{1/2-1/|\aleph|}{|\aleph|-1} r(x) + \frac{|\aleph|/2+1/|\aleph|-1}{|\aleph|-1}, & \text{if } r(x) \in (1,\infty), \\ 1, & \text{if } d(x,\mathbb{M}) = 0. \end{cases}$$

the ratio of the distance to normal set and the distance to malware set is abbreviated by $r(x) = d(x,\mathbb{N})/d(x,\mathbb{M})$. If the output of $\tilde{\varphi}$ overcomes a *threshold* $\theta \in (0,1)$ it might be concluded that the ELF is malicious.

For the threshold of 1/2 $\tilde{\varphi}$ will lead to the same decision as φ. If a lower false positive rate is desired, increase threshold θ. Note that this will be accompanied by a worse detection rate; recall the ROC graph in Figure 5.2.

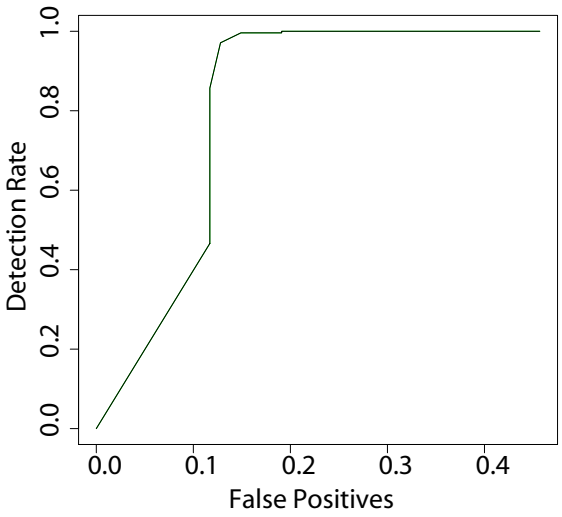


Figure 5.2: ROC graph for nNb with varying threshold and detection function $\tilde{\varphi}$

5.3.3 Static Analysis Using Decision Trees

In this section, we present a simple decision tree for deciding the suspiciousness of the corresponding application. It uses basically the same basis as the foregoing experiment except that in this case all binaries found on the Android system were considered[8].

Additionally, this example also refers to the intrusion detection system presented in Section 4.4. The considered event class is now the execution of a binary and the system observables are the *static function calls* of this executable. Information about functions which might be called by an executable is gained by means of disassembling, which is done by the Linux commands `readelf` and `objdump` in our case, recall Linux `man` pages. We show that normal executables are distinguishable from abnormal, which are represented by Linux malware, on the basis of function appearance in the static table. In our approach the set of normal executables consists of 94

[8]Even those binaries that were installed by us.

```
_edata = y
|  gethostbyname = y
|  |  sigaction = y: normal
|  |  sigaction = n: malicious
|  gethostbyname = n
|  |  fork = y
|  |  |  strerror = y
|  |  |  |  getgrgid = y: malicious
|  |  |  |  getgrgid = n: normal
|  |  |  strerror = n: malicious
|  |  fork = n: normal
_edata = n
|  exit = y: malicious
|  exit = n
|  |  fprintf = y: malicious
|  |  fprintf = n
|  |  |  uname = y: malicious
|  |  |  uname = n
|  |  |  |  execv = y: malicious
|  |  |  |  execv = n
|  |  |  |  |  malloc = y: malicious
|  |  |  |  |  malloc = n
|  |  |  |  |  |  putchar = y: malicious
|  |  |  |  |  |  putchar = n
|  |  |  |  |  |  |  memmove = y: malicious
|  |  |  |  |  |  |  memmove = n: malicious
```

Figure 5.3: Decision tree 1. y means that the function appears in the static table of an executables, n that not. This simple tree result in a detection rate higher than 95%.

Android Linux commands, mostly found in `/bin`, and the set of abnormal executables consists of Linux malware, via Google search we found 240 of the latter.

We induced decision rules in the following way. First, the set of functions, appearing in our normal and malicious set, is reduced by taking only those functions which appear in the malware set and normal set. This is done to exclude any Android specific calls, which are not called in the Linux malware. Second, we apply principal component analysis to reduce further the number of functions we will look at. Third, decision rules are created based on the remaining functions. With the help of the decision tree learner ID3, developed by Quinland [175], we created two efficient and

```
__bss_start = y
|  gethostbyname = y
|  |  sigaction = y: normal
|  |  sigaction = n: malicious
|  gethostbyname = n
|  |  fork = y
|  |  |  strerror = y
|  |  |  |  getgrgid = y: malicious
|  |  |  |  getgrgid = n: normal
|  |  |  strerror = n: malicious
|  |  fork = n: normal
__bss_start = n
|  printf = y: malicious
|  printf = n
|  |  fprintf = y: malicious
|  |  fprintf = n
|  |  |  execv = y: malicious
|  |  |  execv = n
|  |  |  |  memmove = y: malicious
|  |  |  |  memmove = n
|  |  |  |  |  perror = y: malicious
|  |  |  |  |  perror = n: malicious
```

Figure 5.4: Decision tree 2 also achieves detection rates higher 95%.

accurate decision rules based on different function sets, see Figure 5.3 and Figure 5.4.

The accuracy parameters are determined by stratified ten-fold cross validation. The malware detection rates are higher than 95% for both decision trees; the rate of false positives, i.e. normal executables erroneously classified as malicious, is 13% for the first and 11% for the second decision tree, respectively.

5.3.4 Collaborative Intrusion Detection

The collaborative approach is supposed to be support for the server on the one hand, but on the other hand a fall-back alternative, in the case the server is not available due to failure, attack or loss of communication channel. In particular, we show how to use the results of the detection scheme presented in Section 4.4 in a collaborative manner. We introduce the overall approach, give a sample scenario, conduct simulations and discuss the results.

Approach

The collaboration module is triggered when a specific *event* takes place. Subsequently, communication is established with neighboring nodes for the assistance. A *request* takes place for support, e.g. computation or available information. Next, responses are collected and an action is taken after *evaluating* them. Figure 5.5 gives an illustrative example of the collaboration scheme in the context of the *Collaborative Malware Detection* scenario presented in the following.

Based on the approach presented in Section 5.3.2, we extend the on-device detection with a collaborative approach. We introduce an uncertainty interval $[\theta - x, \theta]$, triggering the collaboration mechanism. Hence, the neighboring nodes are requested to determine the detection status according to their classifier. The initiating node collects the responses and builds the arithmetic mean. If the average of the responses is still below θ, the executable is defined as benign, otherwise as malicious. We conducted simulations for this specific scenario.

Simulation

We set up a simulation environment reflecting the characteristics of the ad hoc network scenario. 100 nodes are used in a simulated area of 1500 x 1500 units with transmission range of 200 units. A unit is an abstract term for a distance measure, e.g. meter. In each round, a node is able to communicate to one or several present nodes in his neighborhood determined by his transmission range. The conduction of the algorithm lasts four rounds: worm tries to infect, request for collaboration, response, and evaluation. We performed 100 runs with 100 rounds per run. Nodes are mobile and move every round according to a random walk model with a maximum of (+/-)5 units in each dimension. The attack vector is based on worm propagation, e.g. the Cabir worm [124]. Initially, a device is selected randomly to be infected. Then, the worm tries to infect all devices in transmission range. We apply the aforementioned collaborative detection scheme. If a new device becomes infected, the worm propagates further. If the worm is classified as malicious by a device, this device is removed from the set of susceptible devices.

We define the threshold for detection as $\theta = 0.5$. If the return value is higher, the installed application is considered malicious and removed from

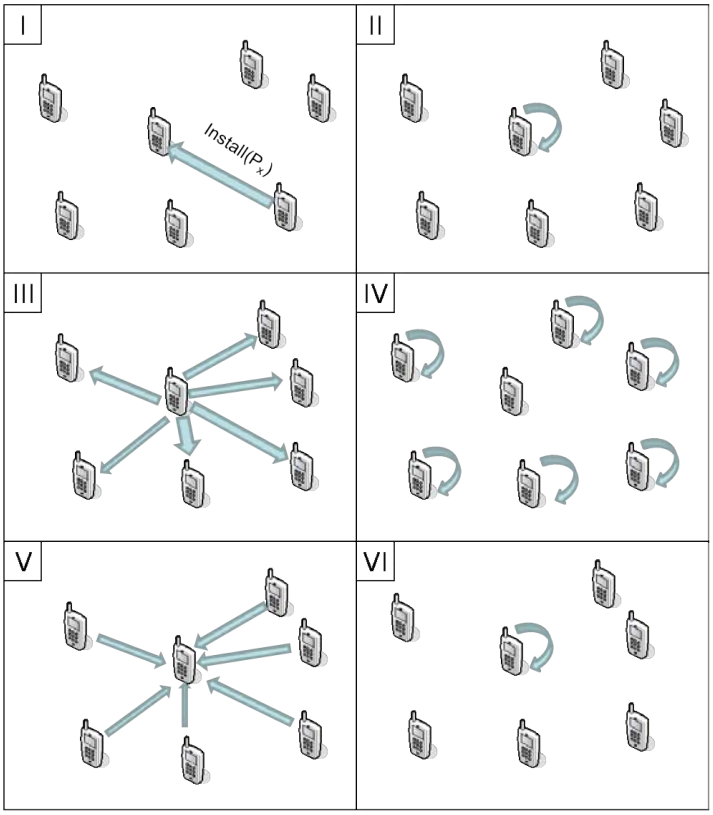


Figure 5.5: **Collaborative Malware Detection:** *I- An infected node tries to install a malicious program to the target device. II- The detection status is determined. In the case, the status is within the uncertainty interval, following steps are triggered. III- A request is sent out to the neighboring nodes with a feature vector containing output of static ELF analysis of the program. IV- Each neighbor nodes determines the detection status of the application according to its trained classifier. V- The initiating node is informed about the results. VI- Evaluation of results; if joint status still falls in the uncertainty interval or below, the node becomes infected. Otherwise, it is removed from the set of susceptible nodes.*

the set of susceptible nodes. If the return value is lower, in case of non-collaborative scenario the node becomes infected. For the simulation, we have two varying input variables. The first is the uncertainty interval $[\theta - x, \theta]$, where we use as x values from 0 to 0.5 in steps of 0.1.

The second input variable is the distribution function for the initial detection values. These are assigned according to normal distribution with a varying mean μ and a standard deviation σ of 1. All resulting values in the interval $[-2\mu, 2\mu]$ are normalized to the interval [0, 1]. Afterwards, for each applied distribution the mean is shifted by continuously adding 0.1. In the case, a value becomes bigger than one, it is set to one.

Results and Discussion

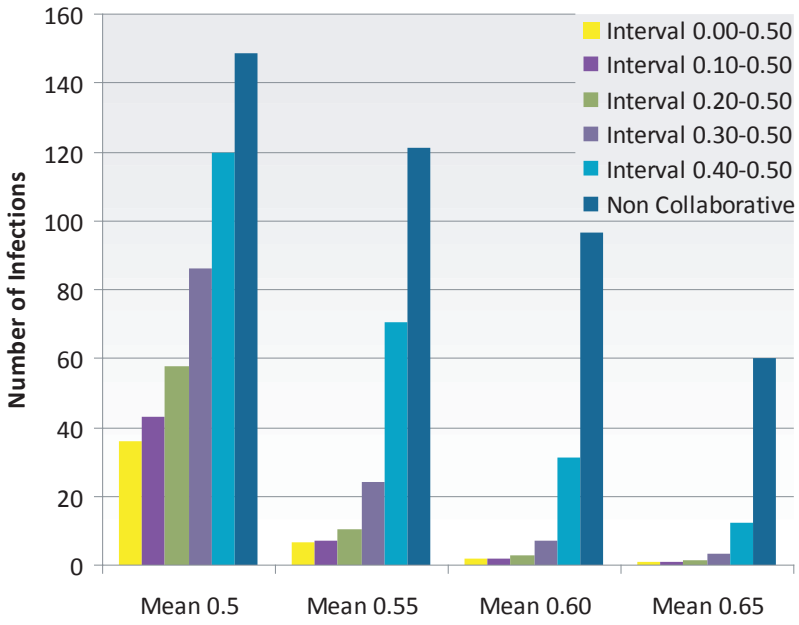


Figure 5.6: Simulation results of the collaborative scheme show that collaboration can significantly lower number of infections.

The results of the simulation are depicted in Figure 5.6. The chart shows the resulting number of infected devices with respect to the initial detection value distribution and varying uncertainty intervals. The first observation is that an increasing uncertainty interval reduces the false neg-

ative rate. In other words, if the collaborative scheme is executed more frequently, this results into fewer infections. On the other hand, it can be seen that the higher the detection value is, the more the collaborative scheme becomes effective. In the first distribution ($\mu = 0.5$), the fraction of the "Interval 0.40 -0.5" approach to the non collaborative approach is 80 per cent whereas with the third distribution ($\mu = 0.6$) this fraction decreases to 32 percent. The most costly combination in terms of communication is $\mu = 0.5$ and the uncertainty interval $x = 0.5$. Here, the averaged maximum of communication acts took place in round 2 with 0.4 per node. Although we focused on decreasing false negative rate, we assume false positives can be reduced by this approach similar as it is shown by Luther *et al.* in [141]. A collaborative scheme is susceptible to attacks. An extensive use of resources, e.g. to drain the battery, can be prevented by defining an explicit counter to serve only a maximum number of requests per time.

5.4 Detecting Symbian OS Malware through Static Analysis

This section represents our second approach and is structured as follows. In Section 5.4.1, we describe how we collected our data set on which our work is basing on. Section 5.4.2 presents our approach towards static analysis of Symbian OS function calls for detecting malicious applications. In Section 5.4.3, we discuss the corresponding results.

5.4.1 Function Call Extraction from Symbian OS Executables

Only few malwares are known for current Symbian OS *3rd*. First malware targeting Symbian OS *3rd* appeared in February 2009 which used a valid certificate. This happened shortly after Collin Mulliner presented a way to bypass the security mechanisms of Symbian OS *3rd* at Black Hat Conference 2008 in Japan. Mulliner additionally stated that he was wondering why no one else was trying this common approach earlier.

In this section, we consider using the former Symbian platform version, namely Symbian OS *2nd*, for benefiting from the huge amount of existing malwares. Although these binaries base on the older version of Symbian

OS, we believe that the results of this work can also be applied to the newer versions of Symbian OS. The main reason for this is that from a function call perspective most of the calls remained the same while some were removed and new ones were added[9].

Over 300 malwares appeared up to date for Symbian OS *2nd*, [190]. We start by eliminating malwares which are simple file containers (installers) overwriting critical files and which are based on similar code bases sometimes only changing the name of the installation file or the installation note. After filtering, we ended up with data sets consisting of 33 Symbian OS *2nd* malwares as well as on 49 popular applications for the same version. 33 malwares obviously do not form a statistically proven set but it is the only possibility to work on real data for smartphone platforms. One could argue that researching stationery systems can lead to transferable solutions for smartphones. But, as a reminder, key differences between these systems have to be taken into account:

- Smartphone are highly connected while frequent connection changes through different networks interfaces are common, e.g. 2G, 2.5G, 3G, Wi-Fi
- Smartphones are single-user systems in most cases which allows to disregard aspects of multi-user systems.
- Most cellular networks use NAT to assign IP-Addresses to cellular- and smartphones which decreases the possibility of attacking IP addresses directly.
- Smartphone operating systems allow developing security systems in a less complex environment.
- Although smartphones provide various functionality, their main purpose is the usage of communication-centric services, like phone, messaging, and nowadays Internet applications.
- In our opinion, most critical threats to smartphones are DoS attacks, information stealing, and financial service charge abuse.

[9] `http://wiki.forum.nokia.com/index.php/Differences_between_S60_2nd_and_3rd_Edition`, visited 16.7.2009.

These aspects encourage us to stick to real malwares for research. Ignoring research on such platform due to limited data sets can result in millions of unprotected users.

We used IDA Pro[10] for extracting the function calls. Comprehensive tutorials can be found online as well as in [58]. While exploring the installation binaries, we faced the problem that not all could be "unpacked" by IDA Pro. To solve this problem we used a tool called "UnSIS[11]" that was able to give us access to the corresponding files. As an alternative, the tool "SISInfo[12]" can be used to do the same.

Table 5.3: Mapping of variables and functions

Variable	**Mapping**
$\mathcal{X}$	database of executables
x	executable
Ω	union of function calls from all $x \in \mathcal{X}$
ω	function call in Ω
$\mathcal{P}(\Omega)$	power set of Ω
$\mathcal{X}_b$	class of benign executables in $\mathcal{X}$
$\mathcal{X}_m$	class of malicious executables in $\mathcal{X}$
Θ	most frequent calls from both $\mathcal{X}_m$ and $\mathcal{X}_b$
Λ	top 14 calls from Θ
$\hat{\mu}_m(\omega)$	frequency of attributes in $\mathcal{X}_m$
$\hat{\mu}_b(\omega)$	frequency of attributes in $\mathcal{X}_b$
$\hat{\sigma}_b(\omega)$	standard deviation in $\mathcal{X}_b$

We state that our findings can be applied to mobile devices. One drawback of this statement is of course that IDA Pro is not available for any smartphone platform. But our research has shown that implementing similar relevant functionality on current and future smartphone platforms will be possible. In case of Android you can install the *readelf*[13] application which delivers detailed information on relocation and symbol tables of each ELF object file. Most interestingly, it outputs the static list of referenced function calls for each application chosen.

[10] `http://www.hex-rays.com/idapro/`, visited 16.7.2009.

[11] `http://developer.symbian.com/main/tools_and_sdks/developer_tools/critical/unsis/index.jsp`, visited 16.7.2009.

[12] `http://www.niksula.cs.hut.fi/~jpsukane/sisinfo.html`, visited 16.7.2009.

[13] `http://unixhelp.ed.ac.uk/CGI/man-cgi?readelf+1`, visited 16.7.2009.

5.4.2 Static Function Call Analysis on Symbian OS Binaries

Descriptive Statistics

When extracting function calls we only take into account the function name itself without considering parameters or arguments. The set of all functions, appearing in at least one of the executables in our database $\mathcal{X}$, will be denoted by Ω, the elements of which are the *attributes* in our machine learning approach. The static function call analysis retrieves for each executable x a set of functions, which establishes the map

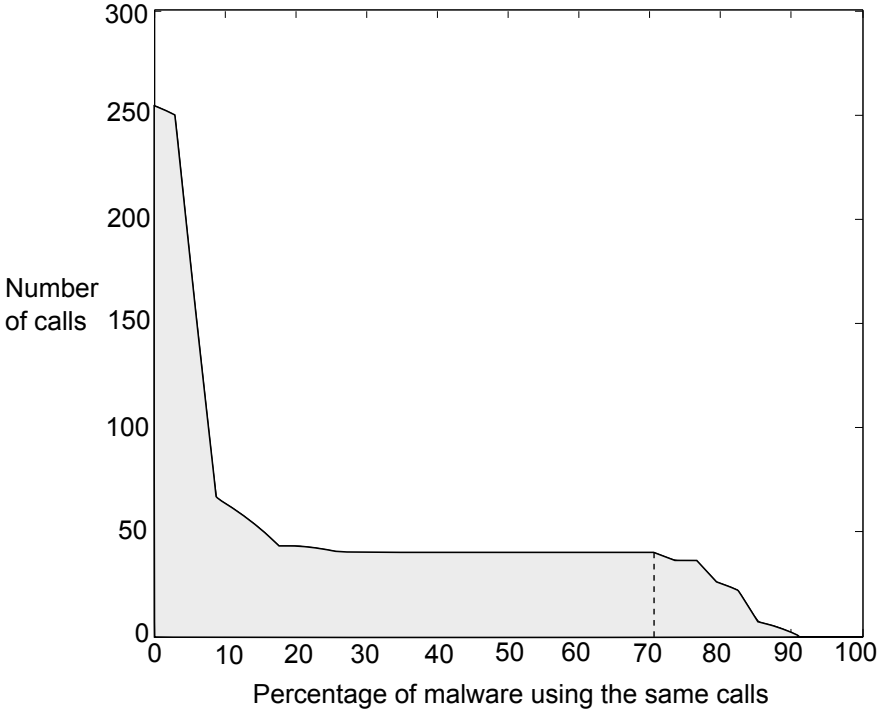


Figure 5.7: Function calls in malware: the x-axis displays the percentage of malware; the y-axis displays the number of common calls from Ω which appear in the malware. The dotted line reveals that there are about 50 calls which appear in 70% of the malwares.

$$\zeta : \begin{array}{ccc} \mathcal{X} & \longrightarrow & \mathcal{P}(\Omega) \\ x & \longmapsto & \zeta(x), \end{array} \tag{5.2}$$

where $\mathcal{P}(\Omega)$ is the power set of Ω.

In this section we reveal some facts on the appearance of function calls in the executables we analyzed and present a descriptive statistic on the attribute distributions. In our database we deal with 33 malicious and 49 benign programs. Overall, 3620 unique function calls where discovered, where 254 of these only appeared in malware, i.e. they are never called by any benign program. The graph in Figure 5.7 gives a rough overview on the attribute distribution on the malware class. It reveals that there are almost 50 attributes in Ω which appear in 70% of the malware, and about 40 which appear in the static analysis of 80% of the malware. Note that the curve in Figure 5.7 declines steeply beyond the 85% mark. Malware detection with a rate higher than 90% is unreachable by simple methods, such as looking at single attributes. The results from Figure 5.7 clearly emphasize that despite of that we already filtered out malwares with similar code bases the other still remain similar. This underlines estimations that most Symbian OS malwares base on a very small set of initial malware source code.

Feature Extraction. To make our detection system more efficient it should only be based on a subset of Ω. Additionally, one should not rely on single malware-specific calls since they may be replaced or omitted in future. To increase robustness with respect to such call replacements, one should take calls into account which are standard and widely used, i.e. calls appearing frequently in both malware and benign programs. The set of these calls will be denoted by Θ. Some characteristics of the most malware-typical calls in Θ are presented in Figure 5.8 and will be called Λ. An attribute $\omega \in \Omega$ is regarded as more malware-typical the greater the quantity gets.

$$t(\omega) = \hat{\mu}_m(\omega) - \big(\hat{\mu}_b(\omega) + 3\hat{\sigma}_b(\omega)\big) \tag{5.3}$$

where $\hat{\mu}_m(\omega) = \frac{1}{|\mathcal{X}_m|} \sum_{x \in \mathcal{X}_m} 1_{\{\omega \in \zeta(x)\}}$ is the frequency of the attribute within the malware class X_m (for '1' being the indicator function which equals 1 if the underscored statement is true and 0 if not), $\hat{\mu}_b(\omega)$ the frequency within the benign programs,

$$\hat{\sigma}_b(\omega) = \sqrt{\frac{1}{|\mathcal{X}_b| - 1} \sum_{x \in \mathcal{X}_b} (1_{\{\omega \in \zeta(x)\}} - \hat{\mu}_b)^2}$$

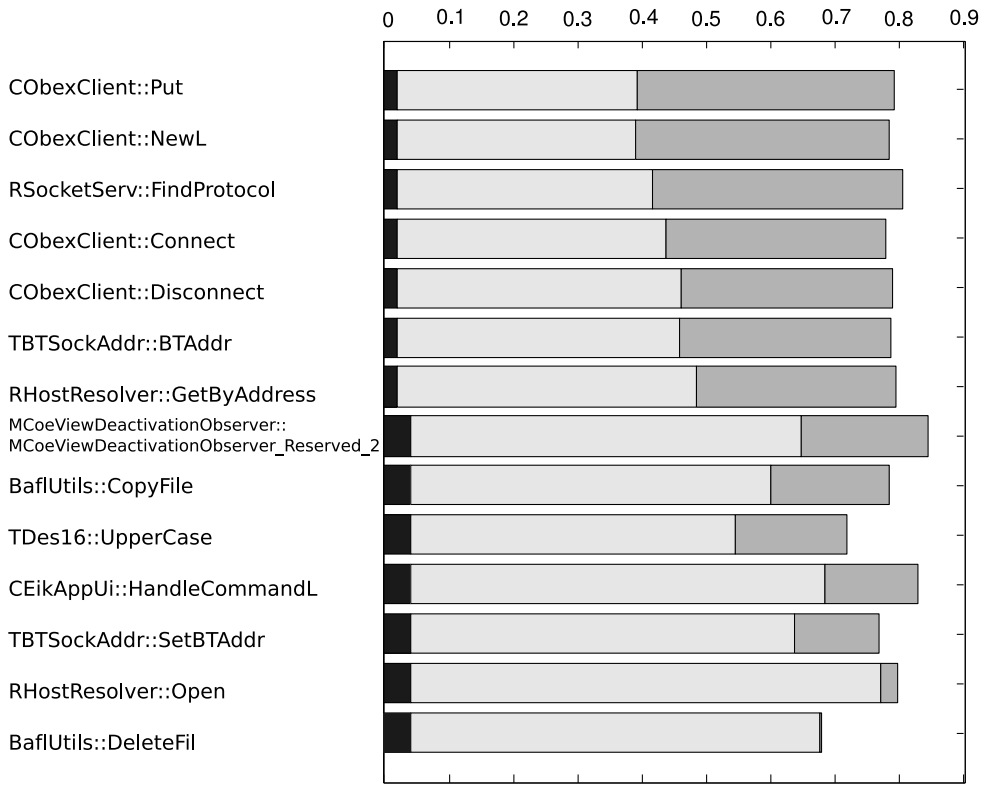


Figure 5.8: Top function calls indicating malware: the x-axis shows the attributes of Λ, the y-axis their appearance frequencies. Black displays the frequency in benign programs ($\hat{\mu}_b(\omega)$), light-gray the frequency in benign programs plus three times standard deviation ($\hat{\mu}_b(\omega) + 3\hat{\sigma}_b(\omega)$), dark-gray the frequency in malware ($\hat{\mu}_m(\omega)$). The longer the dark-gray bar gets, the higher the probability is that a program is malicious, premising the call occurs in the static analysis.

the empirical standard deviation within the benign class X_b. Three times σ is chosen in (5.3) since it is a common empirical rule, meaning that for a real random variable almost all values lie within 3 standard deviations of the mean [173]. As our final feature set Λ, we picked attributes with the greatest $t(\omega) > 0$ values and $\omega \in \Theta$.

Detecting Malware by Means of Machine Learning

With the simple statistical methods of the previous section, detection rates of more than 90% can not be achieved. In order to gain better detection precision and accuracy we employ machine learning techniques. By applying appropriate models, we take the interdependencies of attributes into account. We propose an algorithm called *centroid machine*, designed for detecting Symbian OS malware on the basis of function calls. This suffices the requirements of mobile devices, i.e. being efficient, fast, and light-weight. Furthermore, we compare the quality of centroid machine with a version of a support vector machine and a naive Bayes classifier [184] since every simple classifying algorithm has to keep pace with these state of the art approaches.

Definition of "Centroid Machine". The centroid machine classifies an executable via clustering. Each cluster is defined by a centroid, where the clusters are called c_m and c_b for the malicious and benign classes respectively. An executable is classified as malicious if it is closer to c_m and benign if it is closer to c_b. To make such distance calculations possible, we have to map the set of attributes into a metric space via the kernel function

$$\hat{k} : \begin{array}{rcl} \mathcal{P}(\Omega) & \longrightarrow & \mathbb{R}^{|\Omega|} \\ \mathcal{C} & \mapsto & \sum_{i=1}^{|\Omega|} 1_{\{\omega_i \in \mathcal{C}\}} \overrightarrow{e}_i \end{array} \tag{5.4}$$

for an ordered $\Omega = \{\omega_1, \ldots, \omega_{|\Omega|}\}$. The set $\mathbb{R}^{|\Omega|}$ forms a metric space with the euclidean distance d. After applying the attribute-extracting function ζ from (5.2), we attain a kernel which operates directly on the set of executables $k = k \circ \zeta$.

A centroid for a class $j \in \{\text{'malicious'}, \text{'benign'}\}$ is chosen in a way that minimizes the sum of squared euclidean distances d to all data points of her

Table 5.4: Statistical figures characterizing the quality of different learning models, excerpt from a 10-fold cross-validation. Model approach, malware detection rate, accuracy of the approach, attribute set, and number of attributes in set are shown.

Model	**Detection Rate**	**Accuracy**	**Set**	**Attributes**
Centroid Machine	0.9667	0.9875	Ω	3620
Centroid Machine	0.9505	0.9750	Θ	254
Centroid Machine	0.9333	0.9650	Λ	14
Naive Bayes	0.7890	0.9020	Λ	14
Binary SVM	0.9800	0.9194	Λ	14

class $\mathcal{X}_j$

$$c_j = \min_{\hat{\mu} \in \mathbb{R}^{|\Omega|}} \sum_{x \in \mathcal{X}_j} d^2(\hat{\mu}, x).$$

Then, our classifier can be defined as a map C assigning each executable one of the classes c_m or c_b.

$$C: \quad x \longmapsto \begin{cases} \text{malicious}, & \text{if } d(k(x), c_m) < d(k(x), c_b) \\ \text{benign}, & \text{else.} \end{cases} \tag{5.5}$$

In order to vary the sensitivity we introduce an alarm threshold $\beta \in [0, \infty]$ such that any x is classified as malware if

$$\frac{d(k(x), c_m)}{d(k(x), c_b)} < \beta \tag{5.6}$$

and classified as benign otherwise. This means with increasing β, probability increases that arbitrary checked executables will be detected as malicious. Figure 5.9 visualizes the aforementioned classification.

5.4.3 Results and Discussion

For our statistical investigation we performed 1000 runs of ten-fold cross-validation, where in each loop execution the data is folded randomly into a training set containing 9/10-th of the data and a test set containing the remaining one tenth. We applied the algorithms with different attribute sets, which are the aforementioned Ω, Θ, and Λ. As a representative for support vector machine classifiers, we employed the implementation of Chang and

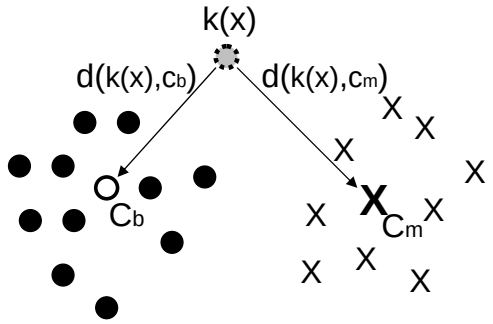


Figure 5.9: Sample clusters of executables with benign center of gravity c_b and malicious c_m in relation to checked executable $k(x)$.

Chih-Jen[14] based on the algorithm proposed by Scholkopf et al. [202] with standard parameters. We also varied the SVM parameters, but results did not improve significantly. We used MatLab[15] as learning environment.

On Table 5.4, the averages of classification rates are displayed. Note that Centroid Machine outperforms naive Bayes and has even a better accuracy than the heavy-weight support vector machine. Also note that the dramatic attribute reduction from 3620 to only 14 is accompanied by an acceptable decline of detection rate and accuracy. This reveals the possibility to only take a relatively small subset of Ω into account while keeping the discriminative potential of the attribute set. This potential small subset of features also allows moving detection logic to mobile devices while not encumbering the devices significantly. By varying the alarm threshold β of the centroid machine in Formula 5.6 the sensitivity varies. The resulting ROC-graph is depicted in Figure 5.10 while the area under the curve is $AUC = 0.9318$.

Referring to Figure 5.8, the function calls from Λ[16] point to Bluetooth-based calls giving a good indication for detecting malware. Due to this numbers, we can additionally state that most *sophisticated*[17] Symbian OS

[14] `libsvm`: a library for support vector machines, 2001, `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[15] `MATLAB, The Language of Technical Computing.`, R2008b, The MathWorks, Inc. `http://www.mathworks.com`, visited 18.7.2009.

[16] This represents the intersected top 14 calls from Symbian OS mal- and software.

[17] Malwares being more sophisticated than ones only overwriting files through exploit-

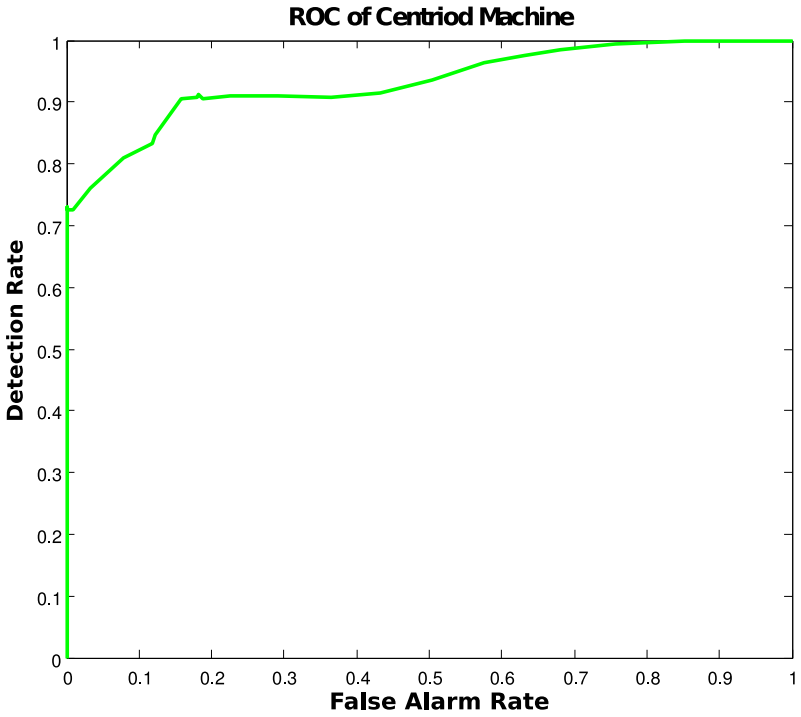


Figure 5.10: ROC curve for the centroid machine for varying alarm threshold β from (5.6). The area under the curve is $AUC = 0.9318$.

malwares obviously use at least Bluetooth for propagation.

Revisiting drawbacks considering static analysis as presented from Moser et al. [149], it is not trivial to estimate the vulnerability of *centroid* to common obfuscation techniques. Since direct usage of machine code is very uncommon in Symbian OS applications[18], appearance of related initial function calls should be detected. Additionally, since our approach only relies on the function calls themselves and not call *sequences*, modification on the sequences do not harm our approach.

5.5 Summary and Conclusion

In this chapter, we discussed the applicability of static analysis to the domain of smartphone platforms in order to detect and prevent infections by malwares. Similar to Chapter 4, our experiments were performed on two platforms: Android and Symbian.

Android represents a great opportunity for researching security aspects on mobile devices, like smartphones. Since it is set open source, this is the first time that most smartphone functionalities and APIs will be available to common developers. This availability and access to the whole system allows critical research on fundamental aspects of smartphones, in our case security. Using static ELF analysis turned out to be an efficient way to detect malware on Android using simple classifiers or decision trees. These results can be improved when applying collaborative measures which can reduce the false-negative rate.

Besides Android, we also investigated applicability of static analysis to Symbian OS since this platform was the main target for malware writers in the past. In terms of dynamic analysis, Symbian OS needs a lot of effort[19] in order to gain meaningful data that can indicate an infection on runtime. Static analysis can bypass this limitation since it only needs access to the binary itself. We described a set of Symbian OS malware analyzed by a clustering method called *centroid*. This clustering method was validated with the analyzed malware set. *Centroid* is based on a static function call analysis and distinguishes malicious from benign program by a learning con-

ing features from the Symbian OS installation system.

[18]Drivers use machine code.

[19]This includes the acquisition of an valid manufacturer certificate.

cept. Furthermore, we compared its quality parameters with some standard state-of-the-art learning algorithms and showed that it is competitive. Additionally, it is having the advantage of being lighter, which makes it more appropriate for the requirements of a smartphone platform. Moreover, we presented an attribute reduction method by which we successfully reduced dimensionality dramatically without significant loss of detection quality.

Chapter 6

Conclusions

6.1 Summary

In Chapter 2, the evolution of smartphones including a definition of the term smartphone is presented. This is necessary in order to have a common understanding of the term throughout this work since no industry standard exists for this. As smartphone hard- and software characteristics change in a continuous manner, a sample smartphone is presented and differences between classic computers and smartphones are discussed additionally. For being able to describe smartphone usage in the year 2010, a survey with about 150 participants is conducted. Additionally, smartphone security is discussed and corresponding threats and mitigation strategies are presented.

In Chapter 3, smartphone malware and its evolution until the end of 2010 is introduced. For understanding ongoing research in the field of anti-malware, commonly used approaches and related work for coping with this threat are presented. This includes signature-based detection, heuristic detection, and detection basing on intrusion detection systems. In this chapter, the fields of *dynamic* and *static analysis* are also introduced.

Chapter 4 describes contributions in the field of *dynamic analysis*. A monitoring system gathering behavior- and system-based information that are processed by a remote system using machine learning for anomaly detection is presented. Detected anomalies indicate activity of malicious applications, e.g. an application that is sending short messages to premium services without knowledge of the user. Anomaly detection has the advantage over classic signature-based approaches that it can be capable of

detecting even unknown threats. A drawback of anomaly detection can be seen in its typically high false-positive rate. Furthermore, a monitoring and detection architecture for Linux-based smartphones is presented which is used to trace execution of binaries. In particular, invoked system calls are monitored and a call frequency tree for analyzing these using a support vector machine as classificator is created.

In Chapter 5, the applicability of *static analysis* to the domain of smartphone platforms is discussed. Two different platforms for the experiments are used: on the one hand Symbian OS, since this was the major target of smartphone malware providing samples of both malicious and benign softwares, and on the other hand Android, since this allowed modifications on system level. In both cases function and system calls are extracted from binaries for being analyzed in a static manner. Results of the analyses are promising and show that the presented methods are even competitive with standard state-of-the-art learning algorithms, e.g. Naive Bayes.

6.2 Contributions and Results

The contribution of this work is a collection of information and methods for smartphone malware detection. The presented approaches base on either *static* or *dynamic analysis* and include all relevant steps starting from data extraction and observables until detection mechanism being run on- or off-device. The individual contributions can be summarized as follows:

Smartphones: The evolution of smartphones is presented and their differences to classic computing devices are explained. Key characteristics of smartphones are that they are hand-held mobile phones with standardized OS supporting native third-party applications. Additionally, small studies describing the usage of smartphones in the year 2010 are presented.

Smartphone malware: A list including all known malicious softwares until the end of the year 2010 is gathered and their key-characteristics are presented. Until November 2010, more than 450 Malware appeared affecting most major smartphone platforms: Symbian OS, Windows (Mobile), Android, and iPhone. Additionally, the motivation for creating smartphone malware shifted towards profit-oriented reasons. 20% of all malwares in November 2010 abused messaging for

sending text messages to premium services. Furthermore, a listing of current research on countermeasures is given.

Smartphone malware detection using dynamic analysis: A novel approach on monitoring smartphones for anomaly detection is presented. In this approach, system status information is extracted that is used in machine learning-based methods for indicating anomalies. It is shown that device usage has measurable impact on the system status and, hence, can potentially detect malicious software and even manual attacks. Fifteen features are recommended for detecting system status-based anomalies on Symbian OS-based smartphones. Additionally, an architecture that enables monitoring and detection of anomalies on Linux-based Android devices is described which is used in a cloud-based system. More experiments are required using extended data set containing samples from benign and malicious applications.

Smartphone malware detection using static analysis: Static analysis on executables from the Android platform is performed and results from classification with PART, Prism and Nearest Neighbor Algorithms, including an option to share results in a collaborative manner, are presented. Static analysis can bypass limitations on data acquisition since it only needs access to the binary itself. Additionally, even methods with low complexity like Nearest Neighbor perform very well when handling function call-based static data. A similar approach is presented that uses function calls from Symbian OS binaries in a static manner. The *Centroid Machine* is presented which outperforms common state-of-the-art classifier, like naive Bayes and SVM, when using a very small attribute set for detection malware. In particular, the *Centroid Machine* detected 96% where naive Bayes and SVM detected 90% and 91%, respectively.

As listed in the introduction, aspects of this dissertation were published as journal article [198], peer-reviewed conference papers [32, 22, 21, 194, 196, 199, 6, 200, 197], technical reports [86, 192, 190, 191, 195], and poster [193]. Additionally, content of this work was used to teach students in seminar and project courses, as well as to find problems to be addressed in bachelor, master, diploma theses.

6.3 Open Issues and Future Work

Most of our approaches suffer the absence of *quantity* and *variety* malware running on a recent operating system. Therefore, it will be an ongoing task to identify and analyze malware that can be used within our experiments.

In case of the runtime monitoring of smartphones, gathering more data from different smartphones running operating systems, like Google Android or iPhone, will be one of the tasks that we will focus in future. Furthermore, we will start to test methods from various fields from machine learning in order to attempt to detect the malicious activities similar to Oberheide *et al.* [162]. A first step towards this can be seen in Luther et al. [141] where biological inspired techniques for analyzing feature-based network data.

In case of the Android-based approaches involving *static analysis*, static ELF analysis turned out to be an efficient way to detect malware on Android using simple classifiers. These results can be improved when applying collaborative measures which can reduce the false-negative rate. Further investigations are needed in order to evaluate our findings using real Android hardware and malware. Real resource consumption will be a significant indicator whether this system can be extended to more complex tasks, e.g. adding more semantic information to the collaborative approach or using more complex classifiers.

In general, it can be assumed that malware detection on smartphones will be a hot topic in the future. Due to increased popularity of smartphones, more and more people will be threatened by upcoming malware. And since these device continuously gain new capabilities, usage will increase leading to more and more victims. Creating application online stores, as known from Apple and Android, was a good step towards protecting users against malicious software. Before being accessible in these stores, tests are run in order to check submitted applications for unintended behavior. Unfortunately, malware writers find ways to bypass these kinds of checks which underlines the omnipresent threat of malware. For these cases, on-device protection is the only possibility for keeping security at acceptable level. Five years ago, on-device detection was not realizable on a large scale of device due to restricted and limited hardware. Nowadays, devices run at 1 GHz and new dual processor architectures are announced making on-device detection feasible. Hence, future research will cover the topic of on-device detection for being able to protect smartphone users against malware.

Bibliography

[1] The 3rd Generation Partnership Project (3GPP). 3gpp – a global initiative. `http://www.3gpp.org/`, 2010. visited 23.02.2010.

[2] Gregory D. Abowd, Liviu Iftode, and Helena Mitchel. The smart phone: A first platform for pervasive computing. *IEEE Pervasive Computing*, 4(2):18–19, April-June 2005.

[3] Norman Abramson. The aloha system: another alternative for computer communications. In *AFIPS '70 (Fall): Proceedings of the November 17-19, 1970, fall joint computer conference*, pages 281–285, New York, NY, USA, 1970. ACM.

[4] W. Jacobi/Siemens AG. Halbleiterverstärker, 1952.

[5] A.A.E. Ahmed and I. Traore. A new biometric technology based on mouse dynamics. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 4(3):165, 2007.

[6] Sahin Albayrak, Katja Luther, Rainer Bye, Stephan Schmidt, Aubrey-Derrick Schmidt, and Karsten Bsufka. Autonomous security – eine neuartige architektur für netzwerkbasierte intrusion detection und response. In Christian Paulsen, editor, *15. DFN Workshop Sicherheit in vernetzten Systemen*, pages G–1 – G–19. DFN–Cert Services GmbH, 2008. ISBN:978-3-833-4-7381-4.

[7] Sahin Albayrak, Christian Scheel, Dragan Milosevic, and Achim Müller. Combining self-organizing map algorithms for robust and scalable intrusion detection. In M. Mohammadian, editor, *Proceedings of International Conference on Computational Intelligence for Modelling Control and Automation (CIMCA 2005)*, pages 123–130. IEEE Computer Society, 2005.

[8] Julia Allen, Alan Christie, William Fithen, John McHugh, Jed Pickel, and Ed Stoner. State of the practice of intrusion detection technologies. Technical Report CMU/SEI-99-TR-028, Carnegie Mellon Software Engeneering Institue, Pittsburgh, PA 15213-3890, jan 2000.

[9] Open Handset Alliance. Android - an open handset alliance project. `http://developer.android.com/`, 2008. visited 26.02.2010.

[10] Magnus Almgren and Ulf Lindqvist. Application-integrated data collection for security monitoring. In *Recent Advances in Intrusion Detection (RAID 2001)*, LNCS, pages 22–36, Davis, California, October 2001. Springer.

[11] Oliver Amft and Paul Lukowicz. From backpacks to smartphones: Past, present, and future of wearable computers. *IEEE Pervasive Computing*, 8(3):8–13, 2009.

[12] Yi an Huang. A cooperative intrusion detection system for ad hoc networks. In *In SASN '03: Proceedings of the 1st ACM workshop on Security of ad hoc and sensor networks*, pages 135—147, 2003.

[13] Anderson, Lunt, Javits, Tamaru, and Valdes. Detecting unusual program behavior using the statistical components of NIDES. Technical report, Computer Science Laboratory, May 1995.

[14] Ross Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*, chapter 10 Monitoring Systems, pages 207–230. Wiley & Sons, 2001.

[15] E. Anquetil and H. Bouchereau. Integration of an on-line handwriting recognition system in a smart phone device. In *ICPR '02: Proceedings of the 16 th International Conference on Pattern Recognition (ICPR'02) Volume 3*, page 30192, Washington, DC, USA, 2002. IEEE Computer Society.

[16] GSM Association. Global mobile communication is 20 years old. `http://gsmworld.com/newsroom/press-releases/2070.htm#nav-6`, September 2007. visited 05.05.2010.

[17] Stefan Axelsson. Intrusion detection systems: A survey and taxonomy. Technical Report 99-15, Department of Computer Engineering Chalmers University of Technology Göteborg, Sweden, March 2000.

[18] Stefann Axelsson. Research in intrusion-detection systems: A survey. Technical Report 98-17, Department of Computer Engineering Chalmers University of Technology Göteborg, Sweden, August 1998.

[19] Rafael Ballagas, Jan Borchers, Michael Rohs, and Jennifer G. Sheridan. The smart phone: A ubiquitous input device. *IEEE Pervasive Computing*, 5:70–77, 2006.

[20] Elad Barkan, Eli Biham, and Nathan Keller. Instant ciphertext-only cryptanalysis of gsm encrypted communication. *Journal of Cryptology*, 21(3):392–429, 2008.

[21] Leonid Batyuk, Aubrey-Derrick Schmidt, Hans-Gunther Schmidt, Ahmet Camtepe, and Sahin Albayrak. Developing and benchmarking native Linux applications on Android. In *Mobile Wireless Middleware, Operating Systems, and Applications*, 2009.

[22] Christian Bauckhage, Tansu Alpcan, and Aubrey-Derrick Schmidt. A probabilistic diffusion scheme for anomaly detection on smartphones. In *Proceedings of the Fourth International Workshop in Information Security Theory and Practice 2010 (WISTP'10)*, pages 31–46, 2010.

[23] Ulrich Bayer, Andreas Moser, Christopher Krügel, and Engin Kirda. Dynamic analysis of malicious code function call. *Journal in Computer Virology*, 2(1):67–77, 2006.

[24] Russell Beale. Supporting social interaction with smart phones. *IEEE Pervasive Computing*, 4(2):35–41, 2005.

[25] M. Becher, F.C. Freiling, and B. Leider. On the effort to create smartphone worms in windows mobile. In *Information Assurance and Security Workshop, 2007. IAW '07. IEEE SMC*, pages 199–206, 20-22 June 2007.

[26] Richard Bejtlich. *The Tao Of Network Security Monitoring: Beyond Intrusion Detection.* Addison-Wesley Professional, 2004.

[27] Steven M. Bellovin. Computer security - an end state? *Communications of the ACM*, 44(3):131–132, 2001.

[28] J. Bergeron, M. Debbabi, J. Desharnais, M. M. Erhioui, Y. Lavoie, and N. Tawbi. Static detection of malicious code in executable programs. In *Proceedings of the Symposium on Requirements Engineering for Information Security (SREIS'01)*, 2001.

[29] Jo Best. Analysis: What is a smart phone? `http://www.silicon.com/technology/mobile/2006/02/13/analysis-what-is-a-smart-phone-39156391/`, February 2006. Post on `Silicon.com`, visited on 05.03.2010.

[30] Alex Biryukov, Adi Shamir, and David Wagner. Real time cryptanalysis of a5/1 on a pc. In *FSE '00: Proceedings of the 7th International Workshop on Fast Software Encryption*, pages 1–18, London, UK, 2001. Springer-Verlag.

[31] Matthew A. Bishop. *The Art and Science of Computer Security*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[32] Thomas Bläsing, Leonid Batyuk, Aubrey-Derrick Schmidt, Seyit Ahmet Camtepe, and Sahin Albayrak. An android application sandbox system for suspicious software detection. In *Proceedings of the 5th International Conference on Malicious and Unwanted Software (Malware 2010)*, Nancy, France, 2010.

[33] Abhijit Bose, Xin Hu, Kang G. Shin, and Taejoon Park. Behavioral detection of malware on mobile handsets. In *Proceeding of the 6th international conference on Mobile systems, applications, and services*, pages 225–238, Breckenridge, CO, USA, 2008. ACM.

[34] Timothy K. Buennemeyer, Theresa M. Nelson, Lee M. Clagett, John P. Dunning, Randy C. Marchany, and Joseph G. Tront. Mobile device profiling and intrusion detection using smart batteries. In *HICSS '08: Proceedings of the Proceedings of the 41st Annual Hawaii International Conference on System Sciences*, page 296, Washington, DC, USA, 2008. IEEE Computer Society.

[35] Y. Bulygin. Epidemics of mobile worms. In *Proceedings of the 26th IEEE International Performance Computing and Communications Conference, IPCCC 2007, April 11-13, 2007, New Orleans, Louisiana, USA*, pages 475–478. IEEE Computer Society, 2007.

[36] Rainer Bye and Sahin Albayrak. CIMD- Collaborative Intrusion and Malware Detection. Technical Report TUB-DAI 08/08-01, Technische Universität Berlin - DAI-Labor, August 2008. http://www.dai-labor.de/fileadmin/files/publications/TRCIMD0808-01.pdf.

[37] Canalys. Smart phone market shows modest growth in q3. `http://www.canalys.com/pr/2009/r2009112.html`, November 2009. visited 26.02.2010.

[38] J. Cendrowska. Prism: An algorithm for inducing modular rules. *International Journal of Man-Machine Studies*, 27, No. 4:349–370, 1987.

[39] Andrew Charlesworth. The ascent of smartphone. *Engineering & Techology*, 4:32–33, 2009. Issue 3.

[40] Abhishek Chaturvedi, Eep Bhatkar, and R. Sekar. Improving attack detection in host-based ids by learning properties of system call arguments. In *In Proceedings of the IEEE Symposium on Security and Privacy*, 2006.

[41] Jerry Cheng, Starsky H. Y. Wong, Hao Yang, and Songwu Lu. Smartsiren: virus detection and alert for smartphones. In *International Conference on Mobile Systems, Applications, and Services (Mobisys 2007)*, pages 258–271, 2007.

[42] Adrian David Cheok, Anuroop Sreekumar, Cao Lei, and Le Nam Thang. Capture the flag: Mixed-reality social gaming with smart phones. *IEEE Pervasive Computing*, 5(2):62–69, 2006.

[43] Brian Chess and Gary McGraw. Static analysis for security. *IEEE Security and Privacy*, 02(6):76–79, 2004.

[44] M. Christodorescu and S. Jha. Static analysis of executables to detect malicious patterns. In *Proceedings of the 12th USENIX Security Symposium*, pages 169–186, 2003.

[45] N. L. Clarke and S. M. Furnell. Authenticating mobile phone users using keystroke analysis. *International Journal of Information Security*, 6:114, 2007.

[46] Oxygen Software Company. Oxygen forensic suite 2010. `http://www.oxygen-forensic.com/`, 2010. visited 09.04.2010.

[47] Pocket Computing. Bellsouth – ibm simon: Pda cellphone. http://cdecas.free.fr/computers/pocket/simon.php. visited 05.05.2010.

[48] Inc. Conexant Systems. Commands for host-processed and host-controlled modems – reference manual. http://www.zoom.com/documentation/dial_up/100498D.pdf, April 2001. visited 12.04.2010.

[49] HTC Corp. T-mobile g1. http://www.htc.com/www/product/g1/overview.html, December 2008. visited 05.03.2010.

[50] Microsoft Corp. Windws phone. http://www.microsoft.com/windowsmobile/en-us/default.mspx, 2010. visited 05.05.2010.

[51] Microsoft Corporation. Windows mobile. http://www.microsoft.com/germany/windowsmobile/default.mspx, 2007. visited 26.02.2010.

[52] Ian Cuddy. Mobile working: Gov connect pulls the plug, June 2009. visited on 02.03.2010.

[53] David Dagon, Tom Martin, and Thad Starner. Mobile phones as computing devices: The viruses are coming! *IEEE Pervasive Computing*, 3(4):11–15, 2004.

[54] Mark Daniel, Jake Honoroff, and Charlie Miller. Engineering heap overflow exploits with javascript. In *Proceedings of the 2nd conference on USENIX Workshop on offensive technologies table of contents*. USENIX Association Berkeley, CA, USA, 2008.

[55] Paul A David. Understanding the economics of qwerty: the necessity of history. In William N. Parker, editor, *Economic History and the Modern Economist*. Basil Blackwell Publishers, New York and Oxford, 1986.

[56] Sampath Deegalla and Henrik Bostrom. Reducing high-dimensional data by principal component analysis vs. random projection for nearest neighbor classification. In *ICMLA '06: Proceedings of the 5th International Conference on Machine Learning and Applications*, pages 245–250, Washington, DC, USA, 2006. IEEE Computer Society.

[57] NTT Docomo. Ntt docomo. http://www.nttdocomo.com/, 2010. visited 05.05.2010.

[58] Chris Eagle. *The IDA Pro Book: The Unofficial Guide to the World's Most Popular Disassembler*. No Starch Press, San Francisco, CA, USA, 2008.

[59] Manuel Egele, Martin Szydlowski, Engin Kirda, and Christopher Kruegel. Using static program analysis to aid intrusion detection. In Roland Bschkes and Pavel Laskov, editors, *DIMVA*, volume 4064 of *Lecture Notes in Computer Science*, pages 17–36. Springer, 2006.

[60] William Enck, Machigar Ongtang, and Patrick Drew McDaniel. On lightweight mobile phone application certification. In *ACM Conference on Computer and Communications Security*, pages 235–245, 2009.

[61] William Enck, Machigar Ongtang, and Patrick Drew McDaniel. Understanding android security. *IEEE Security & Privacy*, 7(1):50–57, 2009.

[62] Tobias Engel. Remote sms/mms denial of service – "curse of silence" for nokia s60 phones. `https://berlin.ccc.de/~tobias/cursesms.txt`, December 2008. visited 12.04.2010.

[63] Eleazar Eskin, Matthew G. Schultz, Salvatore J. Stolfo, and Erez Zadok. Data mining methods for detection of new malicious executables. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, 2001.

[64] European Telecommunications Standards Institute (ETSI). World class standards. `http://www.etsi.org`, 2010. visited 23.02.2010.

[65] F-Secure. Blankfont.a. `http://www.f-secure.com/v-descs/blankfont_a.shtml`. visited 04.06.2010.

[66] F-Secure. Cardblock.a. `http://www.f-secure.com/v-descs/cardblock_a.shtml`, 2005. visited 07.06.2010.

[67] F-Secure. Dampig.a. `http://www.f-secure.com/v-descs/dampig_a.shtml`, March 2005. visited 07.06.2010.

[68] F-Secure. Fontal.a. `http://www.f-secure.com/v-descs/fontal_a.shtml`, 2005. visited 07.06.2010.

[69] F-Secure. Hobbes.a. `http://www.f-secure.com/v-descs/hobbes_a.shtml`, March 2005. visited 04.06.2010.

[70] F-Secure. Mabtal.a. `http://www.f-secure.com/v-descs/mabtal_a.shtml`, July 2005. visited 07.06.2010.

[71] H.H. Feng, J.T. Giffin, Yong Huang, S. Jha, Wenke Lee, and B.P. Miller. Formalizing sensitivity in static analysis for intrusion detection. pages 194–208, 2004.

[72] Chris Fleizach, Michael Liljienstam, Per Johansson, Geoffrey M. Voelker, and Andras Mehes. Can you infect me now? malware propagation in mobile phone networks. In *Proceedings of the 5th ACM Workshop on Recurring Malcode (WORM 2007)*, Alexandira, Virginia, USA, nov 2007. ACM.

[73] Crownhill Mobile Forensics. Simis 2 - forensic sim card software. `http://www.crownhillmobile.com/simis2.htm`, 2008. visited 09.04.2010.

[74] Stephanie Forrest, Stephen A. Hofmeyr, Anil Somayaji, and Thomas A. Longstaff. A sense of self for unix processes. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 120–128, Los Alaitos, CA, 1996. IEEE Computer Society Press.

[75] Stephanie Forrest, Alan S. Perelson, Lawrence Allen, and Rajesh Cherukuri. Self-nonself discrimination in a computer. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 202–212. IEEE Computer Society Press, 1994.

[76] Eibe Frank and Ian H. Witten. Generating accurate rule sets without global optimization. In *Shavlik, J., ed., Machine Learning*, volume Proceedings of the Fifteenth International Conference, 1998.

[77] Joachim Frick and Rainer Bott. Method for identifying a mobile phone user or for eavesdropping on outgoing calls, 2008.

[78] Steven Furnell, Adila Jusoh, and Dimitris Katsabas. The challenges of understanding and using security: A survey of end-users. *Computers & Security*, 25(1):27–35, 2006.

[79] Matthew Glickman, Justin Balthrop, and Stephanie Forrest. A machine learning evaluation of an artificial immune system. *Evolutionary Computation*, 13(2):179–212, 2005.

[80] Ian Goldberg, David Wagner, Randi Thomas, and Eric A. Brewer. A secure environment for untrusted helper applications confining the wily hacker. In *SSYM'96: Proceedings of the 6th conference on USENIX Security Symposium, Focusing on Applications of Cryptography*, pages 1–1, Berkeley, CA, USA, 1996. USENIX Association.

[81] Stefan Görling. The myth of user education. In *Proceedings of the 16th Virus Bulletin International Conference, Montreal*, October 2006.

[82] Alexander Gostev. Mobile malware evolution: An overview, part 1. `http://www.viruslist.com/en/analysis?pubid=200119916`, September 2006. visited 06.04.2008.

[83] Alexander Gostev. Mobile malware evolution: An overview, part 2. `http://www.viruslist.com/en/analysis?pubid=201225789`, October 2006. visited 06.04.2008.

[84] Sophos Graham Cluley. First iphone worm discovered - ikee changes wallpaper to rick astley photo. `http://www.sophos.com/blogs/gc/g/2009/11/08/iphone-worm-discovered-wallpaper-rick-astley-photo/`, November 2009. visited 12.04.2010.

[85] Chuanxiong Guo, Helen J. Wang, and Wenwu Zhu. Smartphone attacks and defenses. In *Proceedings of the Third Workshop on Hot Topics in Networks HotNets-III*, San Diego, CA USA, 2004.

[86] Aubrey-Derrick Schmidt Seyit Ahmet Camtepe Hans-Gunther Schmidt, Karsten Raddatz and Sahin Albayrak. Google android – a comprehensive introduction. Technical Report TUB-DAI 03/09-01, Technische Universität Berlin, 2009.

[87] Simon Hansman and Ray Hunt. A taxonomy of network and computer attacks. *Computers & Security*, 24(1):31–43, 2005.

[88] Oskar Heil. Improvements in or relating to electrical amplifiers and other control arrangements and devices, 1935.

[89] Stephen Hofmeyr and Stephanie Forrest. Architecture for an Artificial Immune System. *Evolutionary Computation Journal*, 8(4):443–473, 2000.

[90] Albert J. Höglund, Kimmo Hätönen, and Antti S. Sorvari. A computer host-based user anomaly detection system using the self-organizing map. In *IJCNN '00: Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN'00)-Volume 5*, page 5411, Washington, DC, USA, 2000. IEEE Computer Society.

[91] Thorsten Holz and Frederic Raynal. Detecting honeypots and other suspicious environments. In *Proceedings of the IEEE Workshop on Information Assurance and Security (2005)*, United States Military Academy, West Point, NY, June 2005.

[92] David D. Hwang, Patrick Schaumont, Kris Tiri, and Ingrid Verbauwhede. Securing embedded systems. *Security & Privacy Magazine, IEEE*, 4(2):40–49, 2006.

[93] Mikko Hypponen. Malware goes mobile. *Scientific American*, November 2006:70–77, November 2006.

[94] Georges Ifrah. *The Universal History of Computing: From the Abacus to the Quantum Computer*. John Wiley & Sons, Inc., New York, NY, USA, 2001. Translator-Harding, E. F.

[95] Scott L. Fulton III. Report: Apple iphone not a smartphone. `http://www.betanews.com/article/Report-Apple-IPhone-Not-a-Smartphone/1169746276`, January 2007. visited on 26.02.2010.

[96] Research in Motion (RIM). Rim. `http://www.rim.com`, 2010. visited 26.02.2010.

[97] AdMob Inc. Admob mobile metrics december 2009. `http://metrics.admob.com/wp-content/uploads/2010/01/AdMob-Mobile-Metrics-Dec-09.pdf`, January 2010. visited on 02.03.2010.

[98] AdMob Inc. Admob mobile metrics february 2010. `http://metrics.admob.com/2010/03/february-2010-mobile-metrics-report/`, March 2010. visited 30.03.2010.

[99] Apple Inc. Apple's app store downloads top two billion. https://www.apple.com/nz/pr/library/2009/09/28appstore.html, September 2009. visited 02.03.2010.

[100] Apple Inc. iphone. http://www.apple.com/iphone/, 2009. visited 26.02.2010.

[101] Google Inc. live-android - a livecd for android. http://code.google.com/p/live-android/, July 2009. visited on 05.03.2010.

[102] Google Inc. Nexus one – web meets phone. http://www.google.com/phone/, 2010. visited 23.03.2010.

[103] Nokia Inc. Nokia 9210 communicator support. http://www.nokia.de/service-und-software/produktservice/nokia-9210-communicator/technische-dokumente, 2010. visited 23.02.2010.

[104] Texas Instruments. The chip that jack built. http://www.ti.com/corp/docs/kilbyctr/jackbuilt.shtml, 2008. visited 04.05.2010.

[105] G.A. Jacoby and N.J. Davis. Battery-based intrusion detection. In *Global Telecommunications Conference, 2004. GLOBECOM '04. IEEE*, volume 4, pages 2250–2255, 2004.

[106] Grant A. Jacoby, Randy Marchany, and Nathaniel J. Davis IV. How mobile host batteries can improve network security. *IEEE Security and Privacy*, 4(5):40–49, 2006.

[107] Jazilah Jamaluddin, Nikoletta Zotou, Reuben Edwards, and Paul Coulton. Mobile phone vulnerabilities: A new generation of malware. In *Proceedings of the 2004 IEEE International Symposium on Consumer Electronics*, pages 199–202, September 2004.

[108] Wayne Jansen and Rick Ayers. Guidelines on cell phone forensics. Technical Report Special Publication 800-101, National Institute of Standards and Technology, May 2007. http://csrc.nist.gov/publications/nistpubs/800-101/SP800-101.pdf.

[109] T. Joachims. Making large-scale SVM learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, chapter 11, pages 169–184. MIT Press, Cambridge, MA, 1999.

[110] Thorsten Joachims. *Learning to Classify Text Using Support Vector Machines: Methods, Theory and Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[111] Dae-Ki Kang, D. Fuller, and V. Honavar. Learning classifiers for misuse and anomaly detection using a bag of system calls representation. In *Information Assurance Workshop, 2005. IAW '05. Proceedings from the Sixth Annual IEEE SMC*, pages 118–125, June 2005.

[112] R.A. Kemmerer and G. Vigna. Intrusion detection: A brief history and overview. *Computer*, 35(4):27–30, Apr 2002.

[113] Allen Kent. *Encyclopedia of Library and Information Science: Volume 25*. CRC Press, 1978.

[114] Hahnsang Kim, Joshua Smith, and Kang G. Shin. Detecting energy-greedy anomalies and mobile malware variants. In *MobiSys '08: Proceeding of the 6th international conference on Mobile systems, applications, and services*, pages 239–252, New York, NY, USA, 2008. ACM.

[115] E. Kirda, C. Kruegel, G. Banks, G. Vigna, and R. Kemmerer. Behavior-based spyware detection. In *USENIX Security Symposium*, 2006.

[116] Teuvo Kohonen. *Self-Organizing Maps*, volume 30 of *Springer Series in Information Sciences*. Springer-Verlag, Third edition, 2001. ISBN 3–540–67921–9, ISSN 0720–678X.

[117] Andrew P. Kosoresow and Steven A. Hofmeyr. Intrusion detection via system call traces. *IEEE Softw.*, 14(5):35–42, 1997.

[118] C. Kruegel, W. Robertson, and G. Vigna. Detecting kernel-level rootkits through binary analysis. In *Proceedings of the Annual Computer Security Application Conference (ACSAC)*, pages 91–100, Dec. 2004.

[119] Christopher Kruegel, William Robertson, Fredrik Valeur, and Giovanni Vigna. Static disassembly of obfuscated binaries. *USENIX Security Symposium*, Volume 13:18 – 18, 2004.

[120] Christopher Kruegel, Fredrik Valeur, and Giovanni Vigna. *Intrusion Detection and Correlation: Challenges and Solutions*. Springer-Verlag TELOS, Santa Clara, CA, USA, 2004.

[121] Moto Labs. Android beyond the phone. `http://labs.moto.com/android-meets-e-ink/`, January 2009. visited on 05.03.2010.

[122] Moto Labs. Android media platform. `http://www.moto.com/amp/`, 2009. visited 05.03.2010.

[123] Carl E. Landwehr, Alan R. Bull, John P. McDermott, and William S. Choi. A taxonomy of computer program security flaws. *ACM Comput. Surv.*, 26(3):211–254, 1994.

[124] G. Lawton. Is it finally time to worry about mobile malware? *Computer*, 41(5):12–14, 2008.

[125] George Lawton. Open source security: Opportunity or oxymoron? *Computer*, 35(3):18–21, 2002.

[126] Neal Leavitt. Mobile phones: The next frontier for hackers? *IEEE Computer*, 38(4):20–23, 2005.

[127] Wenke Lee and Salvatore J. Stolfo. Data mining approaches for intrusion detection. In *In Proceedings of the 7th USENIX Security Symposium*, 1998.

[128] Wenke Lee, Salvatore J. Stolfo, and Kui W. Mok. Mining audit data to build intrusion detection models. In *Knowledge Discovery and Data Mining*, pages 66–72, 1998.

[129] Wenke Lee, S.J. Stolfo, P.K. Chan, E. Eskin, Wei Fan, M. Miller, S. Hershkop, and Junxin Zhang. Real time data mining-based intrusion detection. In *DARPA Information Survivability Conference & Exposition II, 2001. DISCEX '01. Proceedings*, 2001.

[130] Otto Lehmann. Über fliessende kristalle. *Zeitschrift fr Physikalische Chemie*, 4:462–472, 1889.

[131] Peter Leijdekkers and Valerie Gay. Personal heart monitoring system using smart phones to detect life threatening arrhythmias. In *CBMS '06: Proceedings of the 19th IEEE Symposium on Computer-Based Medical Systems*, pages 157–164, Washington, DC, USA, 2006. IEEE Computer Society.

[132] Julius Edgar Lilienfeld. Method and apparatus for controlling electric current, 1930.

[133] Steve Litchfield. Resistive vs capacitive: the invisible tech war in which both opponents can win? http://www.allaboutsymbian.com/features/item/Resistive_vs_Capacitive_the_invisible_tech_war_in_which_both_opponents_can_win.php, March 2009. 05.05.2010.

[134] Zhen Liu, S.M. Bridges, and R.B. Vaughn. Combining static analysis and dynamic learning to build accurate intrusion detection models. In *Proceedings of the Third IEEE International Workshop on Information Assurance (IWIA05)*, pages 164–177, March 2005.

[135] C.L. Lodin, S.W.; Schuba. Firewalls fend off invasions from the net. *Spectrum, IEEE*, 35(2):26–34, Feb 1998.

[136] Daniel Lowry Lough. *A taxonomy of computer attacks with applications to wireless networks*. PhD thesis, Virginia Polytechnic Institute and State University, 2001. Chairman-Davis,IV, Nathaniel J.

[137] NetQin Tech. Co. Ltd. Wince.cxover.a. http://www.netqin.com/en/virus/virusinfo_1366_2.html, 2009. visited 09.04.2010.

[138] Symbian Ltd. Symbian. http://www.symbian.org/, 2010. visited 26.02.2010.

[139] Symbian Ltd. The symbian platform plan. http://developer.symbian.org/wiki/images/f/ff/Symbian_Platform_Roadmap.pdf, 2010. visited on 02.03.2010.

[140] Teresa F. Lunt, R. Jagannathan, Rosanna Lee, Sherry Listgarten, David L. Edwards, Peter G. Neumann, Harold S. Javitz, and A. Valdes. Ides: The enhanced prototype, a real-time intrusion detection system. Technical Report Technical Report SRI Project 4185-010, SRI-CSL-88-12, CSL SRI International, Computer Science Laboratory, 1988.

[141] Katja Luther, Rainer Bye, Tansu Alpcan, Sahin Albayrak, and Achim Müller. A cooperative ais framework for intrusion detection. In *Proceedings of the IEEE International Conference on Communications (ICC 2007)*, 2007.

[142] Federico Maggi, Matteo Matteucci, and Stefano Zanero. Detecting intrusions through system call sequence and argument analysis. *IEEE*

Transactions on Dependable and Secure Computing, 99(PrePrints), 2008.

[143] James W. Mickens and Brian D. Noble. Modeling epidemic spreading in mobile environments. In *WiSe '05: Proceedings of the 4th ACM workshop on Wireless security*, pages 77–86, New York, NY, USA, 2005. ACM Press.

[144] Markus Miettinen, Perttu Halonen, and Kimmo Hätönen. Host-based intrusion detection for advanced mobile devices. In *AINA '06: Proceedings of the 20th International Conference on Advanced Information Networking and Applications - Volume 2 (AINA'06)*, pages 72–76, Washington, DC, USA, 2006. IEEE Computer Society.

[145] Mobeel. Biowallet signature. `http://www.mobbeel.com/en/mobbeel/Products/biowallet.html`, 2010. visited 20.04.2010.

[146] moconews.net. Apple sells record 8.74 million iphones during holidays. `http://moconews.net/article/419-apple-sells-record-8.74-million-iphones-during-holidays/`, January 2010. visited 05.05.2010.

[147] Gorden E. Moore. Cramming more components onto integrated circuits. *Electronics Magazine*, 38(8), April 1965.

[148] Alessandro Moschitti. Making tree kernels practical for natural language learning. In *Proceedings of the 11st Conference of the European Chapter of the Association for Computational Linguistics (EACL 2006)*, 2006.

[149] Andreas Moser, Christopher Kruegel, and Engin Kirda. Limits of static analysis for malware detection. In *Proceedings of the 23rd Annual Computer Security Application Conference (ACSAC)*, pages 421–430, 2007.

[150] Research In Motion. Blackberry. `http://www.blackberry.com/`, 2010. visited 05.05.2010.

[151] Srinivas Mukkamala, Andrew H. Sung, and Ajith Abraham. Intrusion detection using ensemble of intelligent paradigms. *Journal of Network and Computer Applications*, 28:167–182, 2005.

[152] C. Mulliner. Advanced attacks against pocketpc phones. 2006.

[153] Collin Mulliner. Exploiting pocketpc, 2005. Talk on WhatTheHack 2005, http://wiki.whatthehack.org/images/c/c0/Collinmulliner_wth2005_exploiting_pocketpc.pdf.

[154] Collin Mulliner. Exploiting symbian. http://mulliner.org/symbian/feed/CollinMulliner_Exploiting_Symbian_BlackHat_Japan_2008.pdf, October 2008. BlackHat Japan, Tokio, visited online 04.06.2010.

[155] Collin Mulliner. Exploiting symbian: Symbian exploitation and shellcode development. http://mulliner.org/symbian/feed/CollinMulliner_Exploiting_Symbian_BlackHat_Japan_2008.pdf, 2008. Talk on BlackHat Japan 2008, visited 15.6.2009.

[156] Collin Mulliner and Giovanni Vigna. Vulnerability analysis of mms user agents. In *ACSAC '06: Proceedings of the 22nd Annual Computer Security Applications Conference on Annual Computer Security Applications Conference*, pages 77–88, Washington, DC, USA, 2006. IEEE Computer Society.

[157] Darren Mutz, Fredrik Valeur, Giovanni Vigna, and Christopher Kruegel. Anomalous system call detection. *ACM Trans. Inf. Syst. Secur.*, 9(1):61–93, 2006.

[158] Daniel C. Nash, Thomas L. Martin, Dong S. Ha, and Michael S. Hsiao. Towards an intrusion detection system for battery exhaustion attacks on mobile computing devices. In *PERCOMW '05: Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 141–145, Washington, DC, USA, 2005. IEEE Computer Society.

[159] Jarno Niemelä. What makes symbian malware tick. In *Proceedings of the 15th Virus Bulletin Conference*, pages 314–322. Virus Bulletin Ltd., 2005.

[160] Nokia. Nokia e61. http://www.nokia.co.uk/A4221036, 2007. visited 15.8.2007.

[161] Nokiaport. Aufbau eines mobiltelefons. http://nokiaport.de/content/de/inside/mobile_architecture.png, July 2009. visited 05.05.2010.

[162] Jon Oberheide, Evan Cooke, and Farnam Jahanian. Cloudav: N-version antivirus in the network cloud. In *Proceedings of the 17th USENIX Security Symposium (Security'08)*, San Jose, CA, July 2008.

[163] Department of Defense. Dod dictonary of military terms. `http://www.dtic.mil/doctrine/jel/doddict/data/s/04767.html`, 2001. visited 04.10.2007.

[164] National Institute of Standards and Computer Security Division Technology (NIST), Information Technology Laboratory. Standards for security cate-gorization of federal information and information systems. `http://csrc.nist.gov/publications/fips/fips199/FIPS-PUB-199-final.pdf`, February 2004. FIPS PUB 199, visited 18.3.2010.

[165] Chris O'Malley. Bellsouth's communicative simon is a milestone in the evolution of the pda. `http://web.archive.org/web/19990221174856/byte.com/art/9412/sec11/art3.htm`, December 1994. visited 05.05.2010.

[166] Machigar Ongtang, Stephen E. McLaughlin, William Enck, and Patrick Drew McDaniel. Semantically rich application-centric security in android. In *Proceedings of the 25th Annual Computer Security Application Conference (ACSAC)*, pages 340–349, 2009.

[167] Elisa Oyj. Radiolinjas history. `http://www.elisa.com/english/index.cfm?t=6&o=6532.50`, 2004. visited 05.05.2010.

[168] Vern Paxson. Bro: a system for detecting network intruders in real-time. In *SSYM'98: Proceedings of the 7th conference on USENIX Security Symposium, 1998*, pages 3–3, Berkeley, CA, USA, 1998. USENIX Association.

[169] M. Piercy. Embedded devices next on the virus target list. *IEE Electronics Systems and Software*, 2:42–43, December-January 2004.

[170] R.W. Pohl. *Einführung in die Physik*. Springer, Göttingen, 1924.

[171] Phillip A. Porras and Peter G. Neumann. EMERALD: event monitoring enabling responses to anomalous live disturbances. In *1997 National Information Systems Security Conference*, oct 1997.

[172] Niels Provos. Improving host security with system call policies. In *SSYM'03: Proceedings of the 12th conference on USENIX Security Symposium*, pages 18–18, Berkeley, CA, USA, 2003. USENIX Association.

[173] Friedrich Pukelsheim. The three sigma rule. *The American Statistician*, 48:88–91, 1994.

[174] Trirat Puttaraksa. Heap spraying: Introduction. `http://sf-freedom.blogspot.com/2006/06/heap-spraying-introduction.html`, 2006. visited 13.04.2010.

[175] R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[176] Radmilo Racic, Denys Ma, and Hao Chen. Exploiting mms vulnerabilities to stealthily exhaust mobile phone's battery. In *Proceedings of the Second IEEE Communications Society / CreateNet International Conference on Security and Privacy in Communication Networks (SecureComm), Baltimore, MD*, August 2006.

[177] Mika Raento, Antti Oulasvirta, Renaud Petit, and Hannu Toivonen. Contextphone: A prototyping platform for context-aware mobile applications. *IEEE Pervasive Computing*, 4(2):51–59, 2005.

[178] Paruj Ratanaworabhan, Benjamin Livshits, and Benjamin Zorn. Nozzle: A defense against heap-spraying code injection attacks. In *Proceedings of the Usenix Security Symposium*, August 2009.

[179] Nishkam Ravi, Peter Stern, Niket Desai, and Liviu Iftode. Accessing ubiquitous services using smart phones. In *PERCOM '05: Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications*, pages 383–393, Washington, DC, USA, 2005. IEEE Computer Society.

[180] F. Reinitzer. Beiträge zur kenntniss des cholesterins. *Monatshefte fü Chemie*, 9:421–441, 1888.

[181] Eric Rescorla. Security holes... who cares? In *SSYM'03: Proceedings of the 12th conference on USENIX Security Symposium*, pages 6–6, Berkeley, CA, USA, 2003. USENIX Association.

[182] Franklin Reynolds. A call for innovation. *IEEE Pervasive Computing*, 7(1):5–7, 2008.

[183] Brandon Craig Rhodes, James A. Mahaffey, and James D. Cannady. Multiple self-organizing maps for intrusion detection. In *23rd National Information Systems Security Conference - PROCEEDINGS, PAPERS, and SLIDE PRESENTATIONS*, 2000. `http://csrc.nist.gov/nissc/2000/proceedings/2000proceedings.html`, visited 19.04.2007.

[184] Irina Rish. An empirical study of the naive bayes classifier. Technical report, IBM Research Division, 2001.

[185] W. Robertson, C. Kruegel, D. Mutz, and F. Valeur. Run-time detection of heap-based overflows. In *Proceedings of the 17th Large Installation Systems Administrators Conference*, volume 10. USENIX Association, 2003.

[186] George Roussos, Andy J. March, and Stavroula Maglavera. Enabling pervasive computing with smart phones. *IEEE Pervasive Computing*, 4(2):20–27, 2005. April-June.

[187] Didier Samfat and Refik Molva. Idamn: An intrusion detection architecture for mobile networks. *IEEE Journal on Selected Areas in Communications*, 15(7):1373–1380, September 1997.

[188] Julio Sanchez and Maria P. Canton. *Microcontroller programming: the microchip PIC*. CRC Press, 2007. p. 37.

[189] Karen Scarfone and Peter Mell. Guide to intrusion detection and prevention systems (idps). `http://csrc.nist.gov/publications/nistpubs/800-94/SP800-94.pdf`, February 2007. National Institute of Standards and Technology (NIST) Special Publication 800-94.

[190] Aubrey Schmidt and Sahin Albayrak. Malicious software for smartphones. Technical Report TUB-DAI 02/08-01, Technische Universität Berlin, DAI-Labor, February 2008. `http://www.dai-labor.de`.

[191] Aubrey-Derrick Schmidt. Anomaly detection on smartphones. Technical Report Technical Report SR-2008-01, Technische Universität Berlin, August 2008.

[192] Aubrey-Derrick Schmidt. [abstract] smartphone malware evolution revisited. Technical Report SR-2009-01 in GI SIG SIDAR Technical Reports, Proceedings of the Fourth GI Graduate Workshop on Reactive Security (SPRING), September 2009.

[193] Aubrey-Derrick Schmidt. Static smartphone malware detection. Proceedings of the 5th Security Research Conference (Future Security 2010), Berlin, Germany, 2010. Poster presentation.

[194] Aubrey-Derrick Schmidt, Rainer Bye, Hans-Gunther Schmidt, Jan Clausen, Osman Kiraz, Kamer Yüksel, Ahmet Camtepe, and Sahin Albayrak. Static analysis of executables for collaborative malware detection on android. In *IEEE International Congress on Communication (ICC) 2009 - Communication and Information Systems Security Symposium*, pages 1–5, Dresden, Germany, June 2009.

[195] Aubrey-Derrick Schmidt, Rainer Bye, Hans-Gunther Schmidt, Kamer Ali Yüksel, Osman Kiraz, Jan Clausen, Karsten Raddatz, Ahmet Camtepe, and Sahin Albayrak. Monitoring android for collaborative anomaly detection: A first architectural draft. Technical Report TUB-DAI 08/08-02, Technische Universität Berlin - DAI-Labor, August 2008.

[196] Aubrey-Derrick Schmidt, Jan Hendrik Clausen, Seyit Ahmet Camtepe, and Sahin Albayrak. Detecting symbian os malware through static function call analysis. In *Proceedings of the 4th IEEE International Conference on Malicious and Unwanted Software (Malware 2009)*, pages 15–22. IEEE, 2009.

[197] Aubrey-Derrick Schmidt, Frank Peters, Florian Lamour, and Sahin Albayrak. Monitoring smartphones for anomaly detection. In *MOBILWARE '08, Proceedings of the 1st international conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications*, pages 1–6, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[198] Aubrey-Derrick Schmidt, Frank Peters, Florian Lamour, Christian Scheel, Seyit Ahmet Camtepe, and Sahin Albayrak. Monitoring smartphones for anomaly detection. *Mobile Networks and Applications*, 14(1):92–106, 2009.

[199] Aubrey-Derrick Schmidt, Hans-Gunther Schmidt, Leonid Batyuk, Jan Hendrik Clausen, Seyit Ahmet Camtepe, Sahin Albayrak, and Can Yildizli. Smartphone malware evolution revisited: Android next target? In *Proceedings of the 4th IEEE International Conference on Malicious and Unwanted Software (Malware 2009)*, pages 1–7. IEEE, 2009.

[200] Aubrey-Derrick Schmidt, Hans-Gunther Schmidt, Jan Clausen, Kamer Ali Yüksel, Osman Kiraz, Ahmet Camtepe, and Sahin Albayrak. Enhancing security of linux-based android devices. In *in Proceedings of 15th International Linux Kongress*. Lehmann, October 2008.

[201] Bruce Schneier. *Secrets & Lies: Digital Security in a Networked World*. John Wiley && Sons, Inc., New York, NY, USA, 2000.

[202] Bernhard Scholkopf, Alex J. Smola, Robert C. Williamson, and Peter L. Bartlett. New support vector algorithms. *Neural Computation*, 12:1207–1245, 2000.

[203] Andrew Sears, Catherine Plaisant, and Ben Shneiderman. A new era for high precision touchscreens. pages 1–33, 1992.

[204] R. Sekar. Classification of cert/cc advisories 1993–1998. `http://seclab.cs.sunysb.edu/sekar/papers/cert.htm`, July 2001. visited online 17.03.2010.

[205] Dong-Her Shih, Binshan Lin, Hsiu-Sen Chiang, and Ming-Hung Shih. Security aspects of mobile phone virus: a critical survey. *Industrial Management & Data Systems*, 108:478–494, 2008.

[206] Christopher Sholes, 1878.

[207] Alexander Sotirov. Heap feng shui in javascript. `http://www.blackhat.com/presentations/bh-europe-07/Sotirov/Presentation/bh-eu-07-sotirov-apr19.pdf`, 2007. visited 13.04.2010.

[208] Eugene Spafford and Diego Zamboni. Data collection mechanisms for intrusion detection systems. CERIAS Technical Report 2000-08, CERIAS, Purdue University, 1315 Recitation Building, West Lafayette, IN, June 2000.

[209] D. Sterne, P. Balasubramanyam, D. Carman, B. Wilson, R. Talpade, C. Ko, R. Balupari, C.-Y. Tseng, and T. Bowen. A general cooperative intrusion detection architecture for manets. In *Proceedings of the Third IEEE International Workshop on Information Assurance*, pages 57–70, March 2005.

[210] Symantec. Spyware.flexispy. `http://www.symantec.com`, March 2006. visited 27.04.2010.

[211] Peter Szor. *Virus Research and Defense*. Addison Wesley, 2005.

[212] Peter Szor. *Virus Research and Defense*, chapter 11 Antivirus Defense Techniques, pages 425–491. Symantec Press, 2005.

[213] TNS Technology. Consumer trends in mobile applications - a tns technology briefing for technology decision makers, 2005. `http://www.tns-global.com/` (online visited 2007.10.04).

[214] France Rode Thomas M. Whitney and Chung C. Tung. The 'powerful pocketful': an electronic calculator challenges the slide rule. *hp journal online*, 23(10), 1972.

[215] The New York Times. On this day: Phone to pacific from the atlantic. `http://www.nytimes.com/learning/general/onthisday/big/0125.html`, 2010. visited 04.05.2010.

[216] Jie Wu Tiranuch Anantvalee. *Wireless Network Security*, chapter A Survey on Intrusion Detection in Mobile Ad Hoc Networks, pages 159–180. Springer, 2007.

[217] Section 3542: Defintions Title 44, U.S. Code. Information security. visited 18.03.2010.

[218] Ray Tomlinson. The frirst network email. `http://openmap.bbn.com/~tomlinso/ray/firstemailframe.html`. visited 23.02.2010.

[219] Sampo Töyssy and Marko Helenius. About malicious software in smartphones. *Journal in Computer Virology*, 2(2):109–119, 2006.

[220] Patrick Traynor, William Enck, Patrick McDaniel, and Thomas F. La Porta. Mitigating attacks on open functionality in sms-capable cellular networks. In *MOBICOM*, pages 182–193, 2006.

[221] Patrick Traynor, William Enck, Patrick McDaniel, and Thomas F. La Porta. Exploiting open functionality in sms-capable cellular networks. *Journal of Computer Security*, 16(6):713–742, 2008.

[222] Patrick Traynor, William Enck, Patrick McDaniel, and Thomas F. La Porta. Mitigating attacks on open functionality in sms-capable cellular networks. *IEEE/ACM Trans. Netw.*, 17(1):40–53, 2009.

[223] Trifinite.org. Bluebug. `http://trifinite.org/trifinite_stuff_bluebug.html`, 2004. visited 12.04.2010.

[224] Trifinite.org. Bluesmack. `http://trifinite.org/trifinite_stuff_bluesmack.html`, 2004. visited 12.04.2010.

[225] Trifinite.org. Bluesnarf. `http://trifinite.org/trifinite_stuff_bluesnarf.html`, 2004. visited 12.04.2010.

[226] Universität Tübingen. Wer war wilhelm schickard? `http://www-ti.informatik.uni-tuebingen.de/deutsch/schickard/index.html`, December 1999. visited 04.05.2010.

[227] Bundesverband Informationswirtschaft Telekommunikation und neue Medien e.V. BITKOM. Mehr handys als einwohner in deutschland. `http://www.bitkom.de/41015_40990.aspx`, 2006. visited 10.12.2006.

[228] Vladimir N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1995.

[229] Deepak Venugopal and Guoning Hu. Efficient signature based malware detection on mobile devices. *Mobile Information Systems*, 4(1):33–49, 2008.

[230] Hannu Verkasalo and Heikki Hämmäinen. Handset-based monitoring of mobile subscribers. Presented at Mobility Roundtable, Helsinki, Finland, `http://sprouts.aisnet.org/6-50/`, 2006. ISSN 1535-6078, Research paper,.

[231] G. Vigna and C. Kruegel. *Handbook of Information Security*, chapter Host-based Intrusion Detection Systems. Wiley, December 2005.

[232] C. Wagner, G. Wagener, R. State, and T. Engel. Malware analysis with graph kernels and support vector machines. In *Proceedings of the 4th International Conference on Malicious and Unwanted Software (MALWARE), 2009*, pages 63–68, 13-14 2009.

[233] David Wagner and Drew Dean. Intrusion detection via static analysis. In *SP '01: Proceedings of the 2001 IEEE Symposium on Security and Privacy*, page 156, Washington, DC, USA, 2001. IEEE Computer Society.

[234] Pu Wang, Marta C. Gonzalez, Cesar A. Hidalgo, and Albert-Laszlo Barabasi. Understanding the spreading patterns of mobile phone viruses. *Science*, 324(5930):1071–1076, April 2009.

[235] T. Wang, C. Wu, and C. Hsieh. A virus prevention model based on static analysis and data mining methods. In *Computer and Information Technology Workshops*, pages 288–293, 2008.

[236] C. Warrender, S. Forrest, and B. Pearlmutter. Detecting intrusions using system calls: alternative data models. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, pages 133–145, 1999.

[237] Mark Weiser. The computer for the 21st century. *Scientific American*, 265(3):94–104, September 1991.

[238] Gregory White and Vdo Pooch. Cooperating security managers: Distributed intrusion detection systems. *Elsevier Computers & Security*, 15(5):441–450, 1996.

[239] Alma Whitten and J. D.Tygar. Why johnny can't encrypt: a usability evaluation of pgp 5.0. In *SSYM'99: Proceedings of the 8th conference on USENIX Security Symposium*, pages 14–14, Berkeley, CA, USA, 1999. USENIX Association.

[240] Wired.com. Denial-of-service attack knocks twitter offline. `http://www.wired.com/epicenter/2009/08/twitter-apparently-down/`, August 2009. visited 19.03.2010.

[241] www.3GNewsroom.com. Early 3g umts handsets burdened by complexity and high manufacturing costs. `http://www.3gnewsroom.com/3g_news/feb_04/news_4247.shtml`, February 2004. visited 05.05.2010.

[242] Zhiwen Yu, Xingshe Zhou, Daqing Zhang, Chung-Yau Chin, Xiaohang Wang, and Ji Men. Supporting context-aware media recommendations for smart phones. *IEEE Pervasive Computing*, 5(3):68–75, 2006.

[243] Qinghua Zhang and Douglas S. Reeves. Metaaware: Identifying metamorphic malware. In *Proceedings of the 23rd Annual Computer Security Application Conference (ACSAC)*, pages 411–420, 2007.

[244] Yongguang Zhang, Wenke Lee, and Yi-An Huang. Intrusion detection techniques for mobile wireless networks. *Wireless Networks*, 9(5):545–556, 2003.

Appendix A

Acronyms

API	Application Programming Interface
BT	Bluetooth
CPU	Central Processing Unit
DoS	Denial of Service
Email	Electronic Mail
FOMA	Freedom of Mobile Multimedia Access
GPRS	General Packet Radio Service
GSM	Global System for Mobile Communications
GUI	Graphical User Interface
IDS	Intrusion Detection System
IMEI	International Mobile Equipment Identity
IMSI	International Mobile Subscriber Identity
IP	Internet Protocol
IrDA	Infrared Data Association
J2ME	Java 2 Micro Edition
KVM	"Kilobyte" Virtual Machine
MMC	Multimedia Card
MMS	Multimedia Messaging System
OS	Operating System
SDK	Software Developing Kit
SIS	Symbian Installation System
SMS	Short Message Service
TCP	Transmission Control Protocol
UMTS	Universal Mobile Telecommunications System
W-CDMA	Wideband Code Division Multiple Access
WiFi	Wireless Fidelity

Appendix B

The Evolution of Smartphones

Popularity of smartphones increases steadily while the provided technology has taken a long path until today. For showing this long path, a long-term look on the evolution of smartphone technology will be given in this section describing a time span of about 4500 years[1]. The content of this section is not directly related to the topic of "smartphone security" but gives interesting insights to the technological evolution of mobile telecommunication and mobile computing[2].

Starting with the Sumerian abacus constructed about 2700-2300 BC [94], aided computing begun already in ancient times. The abacus represents a tool for performing arithmetic operations representing probably the first mobile computing device ever.

Another important event in history was the use of binary numbers, a general basis of computer technology. Following Sachnez et al. [188], an Indian writer named Pingala was the first to use a binary system for describing rhythm structure of poetry near the year 200. Binary numbers are obviously the fundament of modern computing and telecommunication technology.

Willhelm Schickard's mechanical calculating machine from 1623 [226] can also be seen as a milestone in smartphone evolution. This first mechanical calculator was able to add and subtract 6 digit numbers where "carry overs" needed manual interaction. This machine was followed by the

[1]Most of the relevant technologies were developed in the last 200 years are presented.

[2]Additional entries that might be interest refer to Charles Babbage, Ada Loverlace, COLOSSUS, ENIAC, Janos von Neumann, and Alan Turing.

"Analytical Engine" of Charles Babbage which can be seen as the predecessor of modern computers. Nowadays, highly complex calculation can be performed on mobile computing devices, like smartphones.

In 1833 Gauss and Weber succeeded in transmitting the first telegraphic message over a distance of about 1 kilometer [170]. Besides successful attempts from other researchers, e.g. Morse in the year 1837, the telegraph can be seen as another major step towards modern communication technology.

Further important technological progress was achieved by Antonio Muecci [113] and Alexander Bell. Meucci invented a voice communication apparatus in 1857. He filed a patent caveat[3] on this in 1871 where he did not extend the caveat in 1874 which would have costed $10 USD. This basically allowed Alexander Bell to issue a patent caveat on the telephone 1876. Following the New York Times [215], Bell and Watson were the first to hold a wired conversation in the same year. The findings of Muecci, Gauss, and Weber enabled the development of wireless telephony.

Another interesting event in smartphone evolution was the introduction of QWERTY Keyboards with the Sholes and Glidden typewriters in 1878. Since current computer and smartphone keyboards still use a very similar key layout, several discussion were made on the actual usability and reason for the initial layout which you can see on Figure B.1. Some argue that this layout minimizes jamming of the metallic bars used in typewriters. Others even state that the layout aims for slowing down typist for also minimizing jams [55]. QWERTY keyboards can be found on most early smartphones easening typing data.

In 1888 Friedrich Reinitzer [180] was the first to publish on phenomenons related to melting and freezing of cholesteryl benzoate. Pointing Otto Lehman to these characteristics, Lehman published an article on floating crystals[4] [130]. Key characteristics from these crystals were: I.) the existence of two melting points II.) the reflection of circularly polarized light and III.) the ability to rotate the polarization direction of light. 80 years later, in 1968, the first Liquid Crystal Display (LCD) was built, basing on these essential findings. Current smartphones mainly use LCDs for their displays though a shift towards Organic light-emitting diodes (OLED) can be seen.

[3] A preliminary and less complex patent application was filed.

[4] The original term was "fliessende Kristalle".

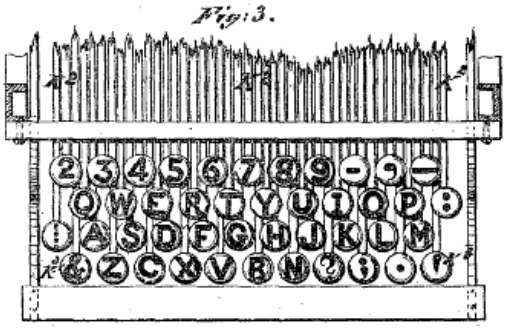


Figure B.1: QWERTY keyboard image taken from a patent filed in 1878 [206]. The image shows that the basic keyboard layout of modern computers did not change for more than 130 years.

After several decades of short range wireless telegraphy research, Guglielmo Marconi[5] was the first to extend range significantly from some hundred meters to about 1.5 kilometers in 1895. He was able to achieve this by positioning the antennas vertically while letting them having contact to the ground. Wireless communication, such as early wireless telegraphy, is obviously one of the most fundamental achievements that influenced the modern mobile information and communication technologies.

The development of wireless communication was followed by another technological milestone: the transistor. In 1926, Julius Edgar Lilienfeld was the first to patent the principle of a transistor while the first working one was constructed and patented by Oskar Heil [132, 88]. The Transistor can be seen as a key component in modern electronics.

According to Horst Zuse[6], the Zuse Z1 developed by his father Konrad Zuse was the first freely programmable binary computer in the world. It was constructed in the living room of Konrad Zuse's parents and was finished in 1938. The finished Z1 was using a vacuum cleaner engine, weighed about 1000 kg and was running at about 1 Hertz.

In 1956, the first fully automatic mobile phone system was commercially released by Ericsson in Sweden. It did not require any manual interaction on base station side and was named *Mobiltelefonisystem A (MTA)*.

[5]Marconi did acknowledge this later in his Nobel Award speech. See: Marconi, "Wireless Telegraphic Communication: Nobel Lecture, 11 December 1909." Nobel Lectures. Physics 1901-1921. Amsterdam: Elsevier Publishing Company, 1967: 196-222. Page 206.

[6]`http://www.horst-zuse.homepage.t-online.de/z1.html`, visited 23.02.2010.

The invention of Integrated Circuits (IC) is another important event in smartphone and electronics history. Early work in this field was patented by Werner Jacobi in 1949 [4] where the invention of the actual IC is credited to Jack Kilby [104]. For his part of this achievement, Kilby won the Nobel prize in the year 2000.

Originating from [147] in 1965, "Moore's Law" has been applied to transistor-based electronics, like modern computers, for decades now. Basically, this law describes a trend in computing evolution, in which the number of transistors that can be placed on an integrated circuit doubles approximately every two years[7]. From 1970 until today, actual development can be applied to this law, disregarding minor deviations.

In 1970, Abramson [3] and his team at the University of Hawaii deployed the ALOHAnet, using low-cost radio systems to create a computer network connecting the different campuses of the university. Challenges and solution faced in the ALOHAnet project influenced the later development of Ethernet.

Following Ray Tomlinson [218], it is very probable that he was the first person to send an email from one computer to another. The exact content of that message is unknown but Ray Tomlinson states that it might have been "QWERTYUIOP" or something similar using two computer connected through ARPANET. In general, email is credited having been the killer application for ARPANET and hence, having remarkable share in the evolution of the Internet.

On 14th April 2009, Hewlett Packard had been awarded the "IEEE Milestone in Electrical Engineering and Computing" for its HP-35 calculator. In 1972, the HP-35 was the first hand-held-sized scientific calculator in world to perform transcendental functions, such as trigonometric, logarithmic, and exponential functions [214]. This calculator can be seen as the beginning of sophisticated hand-held computing and hence being ancestor for most mobile computing devices.

In 1979, Sony introduced its "TPS L2 Walkman" being the first small and portable cassette player with earphones. Smartphones include MP3-Players representing the modern replacement of mobile cassette and compact disc players.

In 1980, Japanese Companies, like Sharp, Casio or Matsushita produced

[7]Originally, Moore predicted a doubling every year.

the first hand-held computers at clock speed of 1 MHz.

The Nordic Mobile Telephony (NMT) system was the first fully-automatic cellular phone system. It was opened in Norway and Sweden in 1981 and also had commercial service in Saudi Arabia at that time. NMT is using an analog communication technique and represents a first generation (1G) mobile phone network.

In 1989, GRiD systems introduced its GRiDPad hand-held computer at a clock speed of 10 MHz supporting handwriting recognition.

A Finnish operator called "Radiolinja" [167] was the first to run a Global System for Mobile Communications (GSM) second generation (2G) phone network in 1991. GSM originated from a memorandum of understanding that was signed by 13 European countries to develop a common cellular phone system in 1987 [16]. GSM originally used two codecs for transmitting a voice spectrum of 3.1 kHz: Half Rate (6.5 kbit/s) and Full Rate (13 kbits/s) [1].

An elementary step towards today's smartphone was the joint venture of IBM and BellSouth, resulting in the final release of the IBM Simon in 1994. The IBM Simon was a device combining functionalities of a cellular phone, a personal digital assistant, a pager and a fax machine. RAM and ROM were each sized 1 Mb and the black and white touch screen supported a resolution of 160x293 pixels. Some sources claim that the Simon was the first smartphone ever [47, 165] but considering our definition of a smartphone to follow in Section 2.1, it was not.

In 1996, Microsoft Windows CE version 1.0 was released as an operating system supporting devices called "hand-held PCs (HPC)". Today, different components of the current Windows CE are used in different operating systems, which are: Windows Mobile Classic (formerly Pocket PC), Windows Mobile Standard (SmartPhone), and Windows Mobile Professional (formerly PocketPC Phone Edition) [50].

EPOC16 is an operating system that was developed by Psion in the end of the eighties. It can be seen as the predecessor of EPOC32 which was released in 1997 (version 1.0). A year later, in 1998, Symbian Ltd. was founded including that EPOC32 was renamed to Symbian OS. Nowadays, this relation still can be seen through the starting command for the Symbian OS emulator, which is `epoc`.

In 1999, the general packet radio service (GPRS) was introduced repre-

senting a circuit switched packet-oriented best effort service. For each transmission, GPRS established a fixed circuit for being able to send packet-based data. GPRS can provide data rates of 56-114 kbits/s while fixed transmission rates and delivery times cannot be guaranteed. GPRS is considered as service being placed between second and third generation (2.5G) of mobile phone networks while it was standardized by ETSI [64] and now is under control of 3GPP [1].

The first phone supporting native third-party applications through an public SDK was the Nokia 9210. Therefore, it can be seen as the first smartphone according to our definition which is described more detailed in the next section. The 9210 was part of Nokia's Communicator series in 2001 and ran Symbian 6.0. Its LCD screen supported a resolution of 640x200 pixel while being able to display 4096 colors. The device provided 14 Mb memory to applications and 2 Mb to users. Additionally, it had an interface for Multi Media Cards (MMC) with up to 16 Mb storage [103].

The Kyocera QCP 6035 was the first smartphone that was released on the U.S. market. It was a combination of a Palm PDA with phone and Internet capabilities. Since it was using Palm OS, third-party application could be installed to the device making it a real smartphone. It supported voice-dialing and used a 20 MHz processor[8].

In 2002, the first commercial Universal Mobile Telecommunications System (UMTS) network of the third generation (3G) was started by NTT DoCoMo's Freedom of Mobile Multimedia Access (FOMA) which is an implementation of UMTS [57]. In the same year, the first Research in Motion (RIM) Blackberry device was released that included push email services [150].

Improving GSM (2G) and GPRS (2.5G) in terms of data transmission rates and delay, Enhanced Data Rates for GSM Evolution (EDGE) (2.75G) was developed providing a theoretical maximum bandwith of 473.6 kbits/s. This is about four times more than GPRS provided.

In 2003, the first UMTS handsets appeared on the market where it is not clear, which device was the very first of the following ones: the NEC e-606, the Motorola A839, or the NOKIA 6650. According to [241] the complexity of mobile phones increased with these first UMTS devices significantly. Considering the integrated circuits, an ordinary 2.5G handset

[8] `http://www.kyocera-wireless.com/qcp-6035-smartphone/index.htm`, visited 11.11.2010.

with EDGE support had *13* ICs, the Nokia 6650 *29*, the Motorola device *68*, and the NEC e-606 *108*.

Windows Mobile 2003 was released in June 2003 where this operating system as the first Windows Mobile version to be affected by smartphone malware. In July 2004, the virus WinCE.Duts.A was discovered. This virus attaches itself to every executable that was not infected beforehand. One month earlier, Symbian OS (S60) was hit by the first smartphone malware ever which propagated via Bluetooth.

In June 2007, Apple Inc. introduced the Apple iPhone in United States of America and sold more than 1.1 million device in fourth quarter 2007 world wide [146]. Sales rates indicated a great success of the iPhone not only in the United States. Key characteristics of the first iPhone were an ARM architecture, four or 8 GB of memory, a LCD screen with 18-bit colors and a resolution of 320x480 pixel. Furthermore, it was using a touchscreen as main user input interface. Interestingly, the first iPhone did not support UMTS, instead it used EDGE and WiFi for higher data transmission rates. Another interesting point is that at time of release, there was no SDK available for the iPhone and no announcement that there will be one in future [100].

About one year later in October 2008, the first Android handset called *HTC Dream* was released. Android is an almost[9] open source operating system for mobile devices using a customized Linux Kernel beneath a modified Java environment [9]. Android not only supports native third party applications, it also allows developers and researchers to replace key components on the system using special developer devices. On most other platforms, access to critical parts of the system is restricted or forbidden. In Spring 2010, the Google Nexus One got available running at 1 GHz having Android 2.1 installed [102].

A summary of these events is listed on Table B.1 and hence represents the evolution of smartphones in a compact format. When considering this evolution, it of course interesting to discuss the possible future of these devices. From the authors point of view, it is more than obvious that the computational and storage capabilities of these devices will increase. Additionally, it is probable that Weiser's [237] vision will become truth also in context of smartphones. The device will shrink and may get an essential part of clothings, bags and similar things. The only remaining

[9]Most parts are set open source except, e.g. some drivers and proprietary applications.

visible parts will be input and output devices, although visability could be also understood as part of virtiual environments, e.g. keyboards being projected on desks, body parts or walls. Smartphones might even get the central computation environment for people since they will provide all the IT-related functionality required being packed into a mobile format.

Table B.1: Significant historic events in smartphone evolution

Year	Event
2700-2300 BC	Sumerian abacus
200	Use of binary numbers
1623	Calculator
1833	Telegraph
1857	Antonio Meucci invented a "sound telegraph"
1868	QWERTY keyboard (first typewriter)
1888	Liquid crystals were discovered
1895	Wireless transmission by Guglielmo Marconi
1926	Transistor
1938	Binary computer Z1 from Konrad Zuse
1956	Mobile Telephone system A (MTA) by Ericsson (0G)
1959	Integrated circuit (IC)
1965	"Moore's Law"
1968	First LCD
1971	Wireless computer network ALOHAnet
1971	First email
1972	First scientific calculator
1980	Matsushita hand-held computers at 1 MHz
1981	Nordic Mobile Telephony (NMT) (1G)
1989	Grid systems hand-held computer at 10 MHz supporting handwriting recognition
1991	First GSM network (2G)
1994	IBM Simon
1996	Palm Pilot
1996	Windows CE
1998	Symbian Ltd.
1999	GPRS Networks (2.5G)
2001	Nokia 9210 Communicator at 66 MHz
2001	Kyocera QCP-6035 at 20 MHz
2002	First commercial UMTS network (3G)
2002	First Blackberry push email device
2003	EDGE (2.75G)
2003	First UMTS devices
2003	Windows Mobile 2003
2007	Apple iPhone at 412 MHz
2008	HTC Dream (Android) at 528 MHz
2010	Google Nexus One (Android) at 1 GHz

Appendix C

List of Extractable Values from Symbian OS

In Table C.1 you can find some more of the extractable information from Symbian OS devices. These information can be accessed by using the given APIs. The following values base on API calls, that were granted using a *developer certificate* that basically every developer can request. Further values, e.g. mobile network information or very sensitive OS data can be accessed using a *phone manufacturer approved certificate* that only trusted partners of Symbian Ltd. can acquire. The table is has three columns, where the name of the extractable values, the complexity of computing, and a description is given.

More Features extracted on Symbian OS devices

Name	**Complexity**	**Description**
KEYLOCK STATUS	simple	Is Keylock activated?
USER INACTIVITY TIME	simple	Time in seconds, where user was inactive
BATTERY CHARGE LEVEL	medium	Battery charge level
BATTERY STATUS	medium	Power supply plugged?
CONNECTION DATA	medium	How many connection interfaces are used and which amount of data was sent (e.g. WLAN, 3G, BT, IrDA, ...)
DATE AND TIME	medium	Date and time on the device
DISK DATA	medium	Size, available space
FILE SYSTEM DATA	medium	files
IMEI	medium	Device identification
IMSI	medium	User identification

More Features extracted on Symbian OS devices

Name	**Complexity**	**Description**
IP ADDRESS	medium	IPv4 and IPv6 Address if assigned
REMOVABLE DATA	medium	Size, available space
REMOVABLE PLUGGED	medium	Is a storage module plugged?
PROCESSES	medium	Running processes, tasks, and threads
CONTACT LIST	medium	Represents the whole contact list
INSTALLED APPLICATIONS	complex	List of installed applications (IDs, names)
OS DATA	complex	CPU usage, available RAM, RAM size
MAIL DATA	complex	Inbox, Outbox, Sentbox, Draft, receipents, contents
MMS DATA	complex	Inbox, Outbox, Sentbox, Draft, receipents, contents
SMS DATA	complex	Inbox, Outbox, Sentbox, Draft, receipents, contents
LOCATION	complex	Cell and GPS information

Appendix D

Malware List

The following tables represent the malwares that could be excerpted from online virus databases. The table gives names, types, days, months, years, and descriptions of the corresponding malwares. The tables are sorted by the discovery dates where the earliest date listed was chosen from the different databases. Please note, that due to a lack of detailed public information, no addtional valuable data after 2008 can be given on smartphone malware.

Malware List

Name	**Type**	**D**	**M**	**Y**	**Payload**
Palm.Libertycrack	Troj.	8	30	2000	Deletes applications and files
Palm.Vapor	Troj.	9	22	2000	Deletes applications and files
Palm.Phage	Virus	9	25	2000	Deletes applications and files
Palm.MTX.II.A	Virus	?	?	2001	displays messages
SymbOS.Cabir.A	Worm	6	15	2004	replicates via Bluetooth
WinCE.Duts.A	Virus	7	17	2004	appends itself to all non-infected exe
SymbOS.Skulls	Troj.	11	19	2004	replaces files, disables apps
SymbOS.Cabir.B	Worm	11	22	2004	replicates via bt, same as cabir.a only txt different
SymbOS.Cdropper.H	Troj.	11	30	2004	drops
SymbOS.Skulls.B	Troj.	11	30	2004	replaces files, disables apps, and drops
SymbOS.Cdropper.C	Troj.	11	30	2004	drops
SymbOS.Cdropper.A	Troj.	12	9	2004	replaces files, drops, disables apps
SymbOS.Cabir.E	Worm	12	14	2004	replicates via bt, only txt is different to cabir b
SymbOS.Cabir.D	Worm	12	14	2004	replicates via bt, only txt and filename changed

Malware List

Name	Type	D	M	Y	Payload
SymbOS.Cabir.C	Worm	12	14	2004	replicates via bt, only txt is different to cabir.b
SymbOS.Cdropper.B	Troj.	12	22	2004	drops
SymbOS.Cabir.J	Worm	12	22	2004	replicates via bt and creates files
SymbOS.Skulls.C	Troj.	12	22	2004	replaces files, disables apps
SymbOS.MGDropper	Troj.	12	22	2004	replaces files and disables apps, drops cabir
SymbOS.Cabir.H	Worm	12	22	2004	replicates via bt
SymbOS.Cabir.G	Worm	12	22	2004	replicates via bt
SymbOS.Cabir.I	Worm	12	29	2004	replicates via bt
SymbOS.Cabir.L	Worm	12	29	2004	replicates via bt
SymbOS.Cabir.F	Worm	12	30	2004	replicates via bt, only filename changed
SymbOS.Cdropper.M	Troj.	12	30	2004	drops cabir.j
SymbOS.Cabir.K	Worm	12	30	2004	replicates via bt and creates files
SymbOS.Cabir.T	Worm	1	5	2005	replicates via bt, only filename changed
SymbOS.Cabir.N	Worm	1	5	2005	replicates via bt, only filename changed
SymbOS.Cabir.O	Worm	1	5	2005	replicates via bt, only filename changed
SymbOS.Cabir.P	Worm	1	5	2005	replicates via bt, only filename changed
SymbOS.Cabir.R	Worm	1	5	2005	replicates via bt, only filename changed
SymbOS.Cabir.Q	Worm	1	5	2005	replicates via bt, only txt and filename changed
SymbOS.Cabir.S	Worm	1	5	2005	replicates via bt ,only txt and filename changed
SymbOS.Skulls.D	Troj.	1	5	2005	drops cabir.m, disables apps, shows image to screen
SymbOS.Cabir.M	Worm	1	6	2005	propagates via bt, only txt and filename changed
SymbOS.Lasco.A	Worm	1	10	2005	replicates via bt, file injection bases on cabir.h source
SymbOS.Cdropper.D	Troj.	2	1	2005	drops cabir variants and shows messages
SymbOS.Cdropper.E	Troj.	2	1	2005	drops cabir.b and locknut
SymbOS.Locknut	Troj.	2	2	2005	drops cabir variants and replaces files which cause a dysfunctional device
SymbOS.Commwarrior.A	Worm	3	7	2005	replicates via bt and mms

Malware List

Name	Type	D	M	Y	Payload
SymbOS.Commwarrior.B	Worm	3	7	2005	replicates via bt and mms, does not choose clock for deciding on replication method
SymbOS.Dampig.A	Troj.	3	8	2005	drops cabir variants disables apps replaces files
SymbOS.Drever.A	Troj.	3	21	2005	disables apps
SymbOS.Drever.B	Troj.	3	22	2005	disables app
SymbOS.Skulls.F	Troj.	3	22	2005	drops cabir variants and locknut.b replaces files, disables apps flashed skull pictures
SymbOS.Drever.C	Troj.	3	22	2005	replaces files, disbles apps-virus scanners
SymbOS.Skulls.F	Troj.	3	24	2005	drops cabir variants and locknut.b, replaces files disbles apps, shows skulls
SymbOS.Skulls.E	Troj.	3	24	2005	replicates via bt drops variants cabir disables apps
SymbOS.Skulls.H	Troj.	3	30	2005	drops cabir variants and locknut.b, replaces files, disables apps
SymbOS.Skulls.G	Troj.	3	30	2005	disables apps, replaces files
SymbOS.Mabir.A	Worm	4	4	2005	replicates via bt and mms, listen on incoming mms and sms and answers with infected mms
SymbOS.Fontal.A	Troj.	4	6	2005	replaces files, prevents reboot
SymbOS.Hobbes.A	Troj.	4	17	2005	replaces files, disables app,s possibly only phone calls work
SymbOS.Locknut.B	Troj.	4	18	2005	drops cabir.v and locknut.b prevents boot installs corrupted files
SymbOS.Cabir.V	Troj.	4	29	2005	replicates via bt, only filename is changed
SymbOS.Cabir.Y	Troj.	4	29	2005	replicates via bt, only name changed
SymbOS.Skulls.I	Troj.	5	5	2005	drops cabir variants and locknut.b, replaces files, disables apps
SymbOS.Skulls.K	Troj.	5	9	2005	drops cabir.m, replaces files, disables apps
SymbOS.AppDisabler.A	Troj.	5	18	2005	disables apps
SymbOS.Skulls.J	Troj.	6	13	2005	drops appdisabler.a which drops cabir.y and locknut.b, disables apps, replaces files

Malware List

Name	Type	D	M	Y	Payload
SymbOS.Singlejump.C	Troj.	6	15	2005	disables files, drops singlejump.b, uses modified variant of cabir to replicate
SymbOS.Fontal.B	Troj.	6	22	2005	replaces files, prevents reboot, disables apps
SymbOS.Skulls.M	Troj.	6	22	2005	replaces files, disables apps
SymbOS.Doomboot.A	Troj.	7	7	2005	replaces files, prevents reboot drains power through sending commwarrior.b via bt prevent reboot
SymbOS.Doomboot.B	Troj.	7	14	2005	replaces files, prevents reboot
SymbOS.Skulls.L	Troj.	7	14	2005	replaces files, drops cabir variants, disables apps
SymbOS.Doomboot.C	Troj.	7	21	2005	replaces files, prevents reboot
SymbOS.Cabir.U	Worm	7	27	2005	replicates via bt
SymbOS.Blankfont.A	Troj.	8	10	2005	replaces files
SymbOS.Cabir.Z	Troj.	8	31	2005	replicates via bt, only filename changed
SymbOS.Fontal.C	Troj.	9	7	2005	replaces files, disables apps, prevents rebooting
SymbOS.Doomboot.D	Troj.	9	7	2005	prevent reboot, replaces files
SymbOS.Skulls.N	Troj.	9	16	2005	replaces files, disables apps
SymbOS.Doomboot.E	Troj.	9	19	2005	prevents reboot, replaces files
SymbOS.Doomboot.G	Troj.	9	22	2005	drops commwarrior.a+b and fontal.a, replaces files, prevents rebooting
SymbOS.Cardtrap.A	Troj.	9	22	2005	copies windows malware to mem card, replaces files, disables apps
SymbOS.Skulls.O	Troj.	9	22	2005	drops fontal.a and commwarrior.b, replaces files, disables apps
SymbOS.Doomboot.F	Troj.	9	22	2005	drops skulls.d, cabir.m, and fontal.a, replaces files, prevents reboot
SymbOS.Appdisabler.D	Troj.	9	23	2005	replaces files, disables apps
WinCE.Brador.A	Troj.	9	23	2005	backdoor
SymbOS.Appdisabler.E	Troj.	9	23	2005	drops cabir.b, replaces files, disables apps
SymbOS.Cardtrap.B	Troj.	9	23	2005	drops doomboot.a, copies windows malware to memory card, replaces files, disables apps
SymbOS.Skulls.P	Troj.	9	26	2005	drops mabir.a, prevents rebooting, replaces files, disables apps

Malware List

Name	Type	D	M	Y	Payload
SymbOS.Singlejump.D	Troj.	9	26	2005	drops cabir variants, replaces files, disables apps, prevents rebooting, malware renamed to onehop.d
SymbOS.Skulls.Q	Troj.	9	27	2005	drops commwarrior.b and cabir variants, replaces files, disables apps
SymbOS.Appdisabler.F	Troj.	9	27	2005	replaces files, disables apps
SymbOS.Appdisabler.G	Troj.	9	29	2005	replaces files, disables apps, drops cabir variants
SymbOS.Cardblock.A	Troj.	10	3	2005	deletes files, sets password to memory card
SymbOS.Skulls.R	Troj.	10	4	2005	drops mabir.a, replaces files, disables apps
SymbOS.Fontal.C	Troj.	10	4	2005	replaces files, disables apps, prevents rebooting
SymbOS.Cardtrap.C	Troj.	10	7	2005	drops components of doomboot.a
SymbOS.Commwarrior.C	Worm	10	14	2005	replicates via bt, mms, and memory card
SymbOS.Cabir.V	Worm	10	24	2005	replicates via bt, only filename is changes
SymbOS.Cardtrp.D	Troj.	11	9	2005	replaces files, disables apps, drops malwares as doomboot component
SymbOS.Doomboot.M	Troj.	11	10	2005	replaces files, prevents rebooting, drops caommwarrior.f
SymbOS.Doomboot.N	Troj.	11	10	2005	replaces files, prevents rebooting
SymbOS.Locknut.C	Troj.	11	10	2005	replaces files, disables apps, prevents rebooting, drops cabir.b
SymbOS.Skulls.S	Troj.	11	10	2005	drops cabir.f, replaces files, disables apps
SymbOS.Skulls.T	Troj.	11	11	2005	replaces files, disables apps, drops locknut.c
SymbOS.Cardtrap.G	Troj.	11	11	2005	drops windows malware to memory card, drops doomboot components
SymbOS.Cardtrap.F	Troj.	11	14	2005	replaces files, disables apps, prevents rebooting
SymbOS.Skulls.U	Troj.	11	14	2005	drops locknut.a and doomboot.a components drops cabir.b cabir.x locknut.c mgdropper.a replaces files, disables apps

Malware List

Name	Type	D	M	Y	Payload
SymbOS.Skulls.V	Troj.	11	18	2005	replaces files, disables apps, drops mgdropper.a locknut.a doomboot.a cabir.b cabir.x
SymbOS.Pbstealer.A	Troj.	11	21	2005	reads private information and send this via bt (contact data)
SymbOS.Doomboot.P	Troj.	11	28	2005	replaces files, prevents reboot
SymbOS.Drever.D	Troj.	11	28	2005	replaces files, disables apps
SymbOS.Ruhag.C	Troj.	11	28	2005	replaces files, disables apps
SymbOS.Cardtrp.H	Troj.	11	28	2005	installs to memory card, replaces files, disables apps
SymbOS.Fontal.G	Troj.	11	29	2005	replaces files, disables apps, prevents rebooting
SymbOS.Doomboot.I	Troj.	11	29	2005	replaces files, disables apps, prevents rebooting
SymbOS.Fontal.D	Troj.	11	29	2005	replaces files, disables apps, drops commwarrior.b
SymbOS.Fontal.E	Troj.	11	29	2005	replaces files, disables apps, prevent rebooting
SymbOS.Fontal.D	Troj.	12	2	2005	replaces files, disables apps, prevents rebooting
SymbOS.Hidmenu.A	Troj.	12	3	2005	replaces files
SymbOS.Pbstealer.B	Troj.	12	4	2005	read provate information and sends this via bt
SymbOS.Pbstealer.B	Troj.	12	5	2005	reads private information and sends this via bt
SymbOS.Doomboot.Q	Troj.	12	5	2005	replaces files, disables apps, prevents rebooting
SymbOS.Bootton.C	Troj.	12	7	2005	replaces files, disables apps, prevents rebooting
SymbOS.Cardtrap.I	Troj.	12	12	2005	replaces files, disables apps, installs windows malware to mem card
SymbOS.Cardtrp.K	Troj.	12	12	2005	replaces files, disables apps, installs windows malware to mem card
SymbOS.Cardtrap.J	Troj.	12	12	2005	reaplces files disables apps, installs windows malware to mem card
SymbOS.Cardtrap.L	Troj.	12	12	2005	replaces files, disables apps, drops windows malware to memory card manipulates private data (deletes calendar and phonebook)

Malware List

Name	Type	D	M	Y	Payload
SymbOS.Singlejump.I	Troj.	12	13	2005	replaces files, disables apps, drops doomboot components
SymbOS.Skulls.O	Troj.	12	13	2005	replaces files, disables apps, drops fontal a and commwarrior.b
SymbOS.Skulls.P	Troj.	12	13	2005	replaces files, disables apps, drops mabir.a cabir variants doomboot and fontal components
SymbOS.Cardtrap.M	Troj.	12	14	2005	replaces files, disables apps, installs windows malware to mem card
SymbOS.Skulls.Q	Troj.	12	14	2005	replaces files, disables apps, drops commwarrior.b and doomboot compnents
SymbOS.Cardtrap.N	Troj.	12	14	2005	replaces files, disables apps, installs windows malware to mem card
SymbOS.Bootton.D	Troj.	12	14	2005	drops doomboot.a and cabir.g, replaces files, disables apps
SymbOS.Dampig.B	Troj.	12	15	2005	drops cabir disables apps replaces files
SymbOS.Cabir.W	Worm	12	15	2005	replicates via bt, only filename changed
SymbOS.Cardtrap.O	Troj.	12	15	2005	replaces files, disables apps, installs windows malware to mem card
SymbOS.Doomboot.R	Troj.	12	15	2005	replaces files, disables apps, prevents rebooting
SymbOS.Cabir.W	Troj.	12	15	2005	replicates via bt, only filename changed
SymbOS.Dampig.C	Troj.	12	16	2005	replaces files, disables apps, drops malware
SymbOS.Cardtrap.P	Troj.	12	16	2005	replaces files, disables apps, drops windows malware to memory card
SymbOS.Bootton.B	Troj.	12	25	2005	replaces files, prevents reboot
SymbOS.Bootton.A	Troj.	12	25	2005	replaces files, disables apps
SymbOS.Singlejump.F	Troj.	12	28	2005	replaces files, disables apps, prevents rebooting, sends singlejump.b to bt devices in range

Malware List

Name	Type	D	M	Y	Payload
SymbOS.Singlejump.G	Troj.	12	28	2005	replaces files, disables apps, drops doomboot.a components sends doomboot.a to bt devices in range
SymbOS.Singlejump.H	Troj.	12	28	2005	reaplces files disables apps, prevents rebooting, sends cabir-dropper to device in bt range
SymbOS.Pbstealer.C	Troj.	1	3	2006	reads private information and sends this via bt
SymbOS.Pbstealer.D	Troj.	1	18	2006	reads private information and sends this via bt
SymbOS.Bootton.E	Troj.	1	18	2006	replaces files, prevents rebooting
SymbOS.Sendtool.A	Troj.	1	18	2006	spreads other malware via bt user interaction needed
SymbOS.Cardtrap.P	Troj.	1	22	2006	replaces files, disables apps, installs windows malware to mem card
SymbOS.Cardtrap.R	Troj.	1	27	2006	replaces files, disables apps, installs windows malware to mem card
SymbOS.Cardtrap.S	Troj.	1	27	2006	replaces files, disables apps, installs windows malware to mem card
SymbOS.Cardtrap.Q	Troj.	1	27	2006	replaces files, disables apps, installs windows malware to mem card
SymbOS.Cardtrap.T	Troj.	2	1	2006	replaces files, disables apps, installs windows malware to mem card
SymbOS.Cardtrap.E	Troj.	2	1	2006	replaces files, disables apps, installs windows malware to mem card
SymbOS.Cardtrap.X	Troj.	2	2	2006	replaces files, disables apps, installs windows malware to mem card
SymbOS.Cardtrap.U	Troj.	2	8	2006	replaces files, disables apps, installs windows malware to mem card
SymbOS.Cardtrap.X	Troj.	2	8	2006	replaces files, disables apps, installs windows malware to mem card
SymbOS.Cardtrap.V	Troj.	2	8	2006	replaces files, disables apps, installs windows malware to mem card

Malware List

Name	Type	D	M	Y	Payload
SymbOS.Cardtrap.W	Troj.	2	8	2006	replaces files, disables apps, installs windows malware to mem card
SymbOS.Cardtrap.Y	Troj.	2	11	2006	replaces files, disables apps, installs windows malware to mem card
SymbOS.Cardtrap.AB	Troj.	2	17	2006	replaces files, disables apps, installs windows malware to mem card
SymbOS.Cardtrap.Z	Troj.	2	17	2006	replaces files, disables apps, installs windows malware to mem card
J2ME.RedBrowser.a	Troj.	2	28	2006	abuses messaging system
SymbOS.Cardtrap.AA	Troj.	3	6	2006	replaces files, disables apps, installs windows malware to mem card
SymbOS.Appdisabler.I	Troj.	3	7	2006	replaces files, disables apps
SymbOS.Commwarrior.D	Worm	3	9	2006	replicates via bt and mms, only txt is changed
SymbOS.Mabtal.A	Troj.	3	10	2006	drops mabir.a, fontal.a, and locknut.b
WinCE.Cxover.A	Worm	3	15	2006	replicates via MS ActiveSync
SymbOS.Doomboot.S	Troj.	3	16	2006	replaces files, prevents rebooting
SymbOS.Commwarrior.E	Worm	3	17	2006	replicates via bt and mms
SymbOS.Commdropper.D	Troj.	3	20	2006	sends commwarrior.e via mms
SymbOS.Cdropper.L	Troj.	3	23	2006	drops cabir.ad
SymbOS.Cardtrap.AC	Troj.	4	5	2006	replaces files, disables apps, installs windows malware to mem card
SymbOS.Cdropper.N	Troj.	4	6	2006	drops cabir.a
WinCE.Letum.A	Worm	4	8	2006	replicates via MS information reads private data sends itself to captured addresses, uses usenet registry entries to propgate in usenet
SymbOS.Arifat.A	Troj.	4	13	2006	reads private information (user password logger) and sends this via sms
SymbOS.Blankfont.B	Troj.	4	16	2006	replaces files, prevents rebooting
WinCE.Brador.B	Troj.	5	6	2006	
SymbOS.Commdropper.C	Troj.	5	17	2006	drops commwarrior.h
SymbOS.Commwarrior.F	Worm	5	17	2006	replicates via bt and mms
SymbOS.Mabtal.B	Troj.	5	17	2006	drops mabir.a

Malware List

Name	Type	D	M	Y	Payload
SymbOS.Commdropper.A	Troj.	5	17	2006	drops commwarrior variants
SymbOS.Bootton.F	Troj.	5	17	2006	replaces files, prevents reboot
SymbOS.Commwarrior.H	Worm	5	18	2006	replicates via bt and mms
SymbOS.Commwarrior.G	Worm	5	18	2006	replicates via mms and bt, reads private information(local contact list)
SymbOS.Commdropper.B	Troj.	5	18	2006	drops commwarrior.a+b+c
SymbOS.Cardtrp.AF	Troj.	5	19	2006	replaces files, disables apps, installs windows malware to memory card
SymbOS.RommWar.A	Troj.	5	19	2006	replaces files, disables apps and buttons
SymbOS.Stealwar.B	Troj.	5	20	2006	drops commwarrior.a, pbstealer.a, and rommwar.a
SymbOS.Stealwar.C	Troj.	5	20	2006	drops pbstealer.f, cabir.k, mabir.a, and commwarrior.b
SymbOS.Stealwar.E	Troj.	5	20	2006	drops cabir.a, commwarrior.a, and pbstealer.f
SymbOS.Stealwar.D	Troj.	5	20	2006	drops cabir.k, pbstealer.f, and commwarrior.c
SymbOS.Stealwar.A	Troj.	5	20	2006	drops pbstealer, commwarrior, or cabir
SymbOS.Cardtrap.AE	Troj.	5	21	2006	replaces files, disables apps, installs windows malware to mem card
SymbOS.Cardtrap.AD	Troj.	5	24	2006	reaplces files disables apps, installs windows malware to mem card
SymbOS.Commwarrior.I	Worm	5	25	2006	replicates via bt and mms
SymbOS.RommWar.B	Troj.	5	25	2006	replaces files, prevents rebooting
SymbOS.Doomboot.T	Troj.	5	25	2006	replaces files drops commwarrior.l prevents rebooting
SymbOS.RommWar.D	Troj.	5	25	2006	reaplces files disables apps and buttons
SymbOS.RommWar.C	Troj.	5	25	2006	replaces files, prevents rebooting
SymbOS.Romride.B	Troj.	6	2	2006	replaces files disables files
SymbOS.Romride.A	Troj.	6	2	2006	replaces files, disables apps
SymbOS.Romride.E	Troj.	6	5	2006	replaces files, disables apps
SymbOS.Commwarrior.L	Worm	6	5	2006	replicates via bt and mms
SymbOS.Romride.D	Troj.	6	5	2006	replaces files, disables apps
SymbOS.Commwarrior.K	Worm	6	5	2006	replicates via bt and mms
SymbOS.Commdropper.D	Troj.	6	5	2006	drops commwarrior.e
SymbOS.Romride.C	Troj.	6	5	2006	replaces files, disables apps

Malware List

Name	Type	D	M	Y	Payload
SymbOS.Commwarrior.J	Worm	6	6	2006	replicates via bt and mms
SymbOS.Commdropper.E	Troj.	6	6	2006	drops commwarior.d
SymbOS.Romride.F	Troj.	6	21	2006	replces files disables apps
SymbOS.Romride.H	Troj.	6	21	2006	replaces files, disables apps
SymbOS.Romride.G	Troj.	6	21	2006	reaplces files disables apps
SymbOS.Dropper.A	Troj.	6	22	2006	drops windows malware
SymbOS.Commdropper.G	Troj.	6	22	2006	drops commwarrior.m
SymbOS.Cardtrp.AG	Troj.	6	22	2006	reaplces files disables apps, installs windows malware to the memory card
SymbOS.Commwarrior.N	Worm	6	22	2006	replicates via bt and mms
SymbOS.Commwarrior.M	Worm	6	22	2006	replicates via bt and mms
SymbOS.Commdropper.F	Troj.	6	23	2006	drops commwarrior.k
SymbOS.Cdropper.F	Troj.	6	28	2006	drops cabir variants
SymbOS.Cdropper.K	Troj.	6	28	2006	drops cabir.b components
SymbOS.Cdropper.G	Troj.	6	28	2006	drops cabir and skulls components
SymbOS.Cdropper.I	Troj.	6	28	2006	drops locknut and cabir
SymbOS.Cdropper.J	Troj.	6	29	2006	drops cabir.b
SymbOS.Cdropper.O	Troj.	6	30	2006	drops cabir.a+b
SymbOS.Cdropper.R	Troj.	6	30	2006	drops cabir
SymbOS.Dampig.D	Troj.	6	30	2006	drops dampig.a and cabir variants
SymbOS.Cdropper.S	Troj.	6	30	2006	drops cabir variants
SymbOS.Doomboot.U	Troj.	6	30	2006	replaces files, prevents rebooting
SymbOS.Cdropper.P	Troj.	6	30	2006	drops cabir variants
SymbOS.Cdropper.Q	Troj.	7	2	2006	drops cabir variants
SymbOS.Doomboot.W	Troj.	7	4	2006	replaces files, prevents reboot
SymbOS.Doomboot.V	Troj.	7	4	2006	replaces files, prevents reboot
SymbOS.Ruhag.D	Troj.	7	5	2006	replaces files, disables apps
SymbOS.Ruhag.E	Troj.	7	6	2006	replaces files, disables apps
SymbOS.Cabir.X	Worm	7	6	2006	replicates via bt, only file name changed
SymbOS.Skulls.R	Troj.	7	6	2006	replaces files, disables appsdrops mabir.a
SymbOS.Commdropper.H	Troj.	7	7	2006	drops commwarrior.g
SymbOS.Doomboot.X	Troj.	7	7	2006	replaces files, prevents rebooting
SymbOS.Mabir.B	Troj.	7	8	2006	replicates via mms and bt
SymbOS.Doomboot.P	Troj.	7	26	2006	replaces files, prevents rebooting
SymbOS.Commwarrior.Q	Troj.	8	1	2006	replicates via bt mms memory card uses browser
SymbOS.Bootton.G	Troj.	8	8	2006	replaces files, prevents rebooting
J2ME.Wesber.a	Troj.	9	6	2006	abuses nessaging

Malware List

Name	Type	D	M	Y	Payload
SymbOS.Blankfont.C	Troj.	9	10	2006	replaces files, disables apps, prevents rebooting
SymbOS.Appdisabler.L	Troj.	10	26	2006	reaplces files disables apps
SymbOS.Appdisabler.K	Troj.	10	26	2006	replaces files, disables apps
SymbOS.Appdisabler.J	Troj.	10	26	2006	replaces files, disables apps
SymbOS.Keaf	Worm	10	29	2006	reads private information abuses messaging (sends link for downloading itself to all contacts)
SymbOS.Appdisabler.M	Troj.	10	31	2006	replaces files, disables apps
SymbOS.Appdisabler.N	Troj.	11	7	2006	replaces files, disables apps
SymbOS.Appdisabler.Q	Troj.	11	7	2006	replaces files disableas apps
SymbOS.Appdisabler.O	Troj.	11	7	2006	replaces files, disables apps
SymbOS.Stealwar.F	Troj.	11	7	2006	doprs cabir.a commwarrior.a mosquit.a lasco.a pbstealer.f
SymbOS.Appdisabler.P	Troj.	11	7	2006	replaces files, disables apps
SymbOS.Cardtrap.AH	Troj.	11	7	2006	replaces files, disables apps install windows malware to mem card
SymbOS.Romride.I	Troj.	11	9	2006	reaplces files causes boot loop
SymbOS.Flerprox.A	Troj.	11	9	2006	reaplces files disables apps
SymbOS.Romride.J	Troj.	11	9	2006	replaces files replaces files causes boot loop
SymbOS.Appdisabler.R	Troj.	11	11	2006	replaces files, disables apps
SymbOS.Appdisabler.S	Troj.	11	29	2006	replaces files, disables apps
SymbOS.Appdisabler.T	Troj.	12	11	2006	replaces files, disables apps
SymbOS.Appdisabler.U	Troj.	12	11	2006	reaplces files disables apps
SymbOS.Commdropper.J	Troj.	12	22	2006	drops commwarrior.e
SymbOS.Commwarrior.T	Troj.	1	15	2007	replicates via bt mms memory card
SymbOS.Commwarrior.h	Worm	1	15	2007	reads private data replicates via mms and bt
SymbOS.RommWar.c	Troj.	1	25	2007	no description available
SymbOS.Cabir.AD	Troj.	1	25	2007	replciates via bt, only filename changed
SymbOS.Cabir.AI	Troj.	1	25	2007	replicates via bt
SymbOS.Cabir.AE	Troj.	1	25	2007	replicates via bt
SymbOS.Commwarrior.i	Worm	2	11	2007	replicates via bt and mms
SymbOS.Mrex.a	Troj.	3	27	2007	no description available
SymbOS.Viver.A	Troj.	5	15	2007	abuse messaging
SymbOS.Viver.B	Troj.	5	17	2007	abuses messaging
SymbOS.Feaks.a	Troj.	5	29	2007	abuses messaging
SymbOS.Appdisabler.V	Troj.	5	29	2007	replaces files, disables apps
SymbOS.Feak.a	Troj.	5	29	2007	no description available

Malware List

Name	Type	D	M	Y	Payload
SymbOS.Bootton.H	Troj.	6	27	2007	reaplces files prevents rebooting
SymbOS.Bootton.I	Troj.	6	28	2007	replaces files, prevents rebooting
SymbOS.Fontal.i	Troj.	7	31	2007	replaces files, disables apps
SymbOS.SHT.a	Troj.	8	29	2007	no description available
SymbOS.Skuller.af	Troj.	8	31	2007	no description available
SymbOS.Delcon.a	Troj.	8	31	2007	no description available
SymbOS.Pbstealer.f	Troj.	8	31	2007	abuses messaging read private information
SymbOS.Appdisabler.W	Troj.	8	31	2007	replaces files, disables apps
SymbOS.Appdisabler.x	Troj.	10	31	2007	no description available
SymbOS.HatiHati.a	Worm	12	13	2007	abuses mesaging replicates via mmc
SymbOS.Fonzi.a	Troj.	1	5	2008	no description available
SymbOS.Killav.a	Troj.	1	10	2008	replaces files, disables apps
SymbOS.Beselo.a	Worm	1	2	2008	replicates via bt and mms
SymbOS.Cabir.o	Worm	1	23	2008	no description available
SymbOS.Beselo.b	Worm	1	23	2008	replicates via bt and mms
SymbOS.Lasco.b	Worm	1	26	2008	no description available
SymbOS.Acallno.b	Troj.	1	26	2008	no description available
SymbOS.Kiazha.A	Troj.	3	4	2008	reads private information
SymbOS.Multidropper.A	Troj.	3	4	2008	drops
SymbOS.Flocker.A	Troj.	4	29	2008	abuses messaging
SymbOS.Commwarrior.AA	Worm	5	20	2008	drops
SymbOS.Commdropper.L	Troj.	5	20	2008	drops
SymbOS.Beselo.E	Worm	5	20	2008	replaces files, drops
SymbOS.Pbstealer.H	Troj.	5	20	2008	sends files
SymbOS.Pbstealer.I	Troj.	5	20	2008	sends files
SymbOS.Cabir.H	Worm	9	8	2008	drops
J2ME.Konov.A	Troj.	10	27	2008	sends email, infects Windows PCs
J2ME.Konov.B	Troj.	11	3	2008	sends email, infects Windows PCs
WinCE.PMCryptic.A	Troj.	11	18	2008	drops, uses memory card

Please note, that due to a lack of detailed public information, no addtional valuable data after 2008 can be given on smartphone malware.