Detecting Symbian OS Malware through Static Function Call Analysis

Aubrey-Derrick Schmidt, Jan Hendrik Clausen,
Ahmet Camtepe, and Sahin Albayrak
Technische Universität Berlin - DAI-Labor
Ernst-Reuter-Platz 7, 10587 Berlin
{aubrey.schmidt, jan.clausen, ahmet.camtepe, sahin.albayrak}@dai-labor.de

Abstract

Smartphones become very critical part of our lives as they offer advanced capabilities with PC-like functionalities. They are getting widely deployed while not only being used for classical voice-centric communication. New smartphone malwares keep emerging where most of them still target Symbian OS. In the case of Symbian OS, application signing seemed to be an appropriate measure for slowing down malware appearance. Unfortunately, latest examples showed that signing can be bypassed resulting in new malware outbreak.

In this paper, we present a novel approach to static malware detection in resource-limited mobile environments. This approach can be used to extend currently used third-party application signing mechanisms for increasing malware detection capabilities. In our work, we extract function calls from binaries in order to apply our clustering mechanism, called centroid. This method is capable of detecting unknown malwares. Our 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.

1 Introduction

Our daily lives become more and more dependent upon smartphones due to their increasing capabilities. Smartphones are used for various purposes including web browsing or online payment. Thus, security threats to these devices become more and more critical.

Malware writers started creating malware for smartphones in 2004, mainly targeting Symbian OS. Symbian introduced mandatory application signing in version S60 3rd in 2006[1] for coping with this problem. Application signing was performed by third-party companies that checked submitted applications for a certain set of requirements, e.g. proper memory handling and certificate level. The certificate basically grants access to different kinds of API calls basing on the privilege level. The signing mechanism turned out to prevent malware development for this platform until the first spyware for Symbian OS 3rd was published [15]. After this spyware appearance it took almost 3 years until the first malware for the 3rd version appeared [6]. The reason for this is not obvious. On the one hand one could argue that the market share of Symbian OS is not as big as the one of MS Windows. On the other hand the complexity of Symbian OS programming might distract malware writers looking for "fast" results. Therefore, we can state that applying third-party application checking is a good approach towards malware prevention while applied mechanisms should consider more comprehensive checks for detecting malware.

In this work, we apply static analysis of function calls in order to detect malicious applications. This presents a novel approach to static malware detection in mobile environments and can extend third-party application checking for increased application security. Additionally, not only signing processes[2] 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[3] or iPhone[4]. 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 device.

Basing on common clustering methods, we developed a light-weight algorithm called *centroid machine*. This algorithm is used for detecting, in particular, Symbian OS malware on the basis of function calls which suffices the re-

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

[2] as known from Symbian OS

[3] `http://code.google.com/android`

[4] `http://www.apple.com/iphone/`

quirements 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 [11] as well as with a heavy-weight support vector machine method. Our 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.

This work is structured as follows. In Section 2, we present related work on static analysis for malware detection. In Section 3, we describe how we collected our data set on which our work is basing on. Section 4 presents our approach towards static analysis of Symbian OS function calls for detecting malicious applications. In Section 6, we conclude.

2 Related Work

Static analysis is a quite new field for mobile malware research, therefore most related publications still refer to stationary systems, like PCs.

Wang et al. [16] 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.

Egele et al. [5] 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. [3] use static analysis for creating assembler-based program automatons for detecting malicious activity using the disassembler IDA Pro. 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 an stationary system. The drawback is that on-device detection requires light-weight methods complicating a possible transfer of the presented approach to a limited mobile device.

Bayer et al. [1] 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.

Bergeron et al. [2] 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. [7] 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 path are created by creating states of the observed machine for analysis of control flows.

Provos [9] 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. [17] 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 gives 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.

Moser et al. [8] investigate the limits of static analysis for malware detection. They state that using obfuscation detection via static analysis can be evaded. Therefore, static analysis should be extended by dynamic analysis. In our case we belief that our approach is less vulnerable to some of the mentioned obfuscation techniques. Considering changes in call sequences, our approach will ignore any of these since simple statistics on function call occurrences are checked.

Schmidt et al. [13] use system and library functions for comparing malware with benign software on Android. Using state-of-the-art classifiers resulted in high detection rates with moderate false-positive rate. Since Android lacks corresponding malwares, Linux malwares were ported to this platform for building a source of malicious software. Applicability was checked by comparing original source code with ported source code where the results showed only minor and ignorable differences.

3 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 hap-

pened shortly after Mulliner[5] 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. These events suggest us that the number of such malwares developed may increase substantially in near future.

In this work, 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[6].

Over 300 malwares appeared up to date for Symbian OS *2nd*, [12]. 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 proof 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 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, WiFi
- 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 to develop 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.

[5]Security Researcher from Berlin, Germany

[6]http://wiki.forum.nokia.com/index.php/Differences_between_S60_2nd_and_3rd_Edition

Table 1. 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 $\mathcal{X}_b$

- Most critical threats to smartphones are DoS attacks, information stealing, and financial service charge abuse.

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[7] for extracting the function calls. Comprehensive tutorials can be found online as well as in [4]. 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[8]" that was able to give us access to the corresponding files. As an alternative, the tool "SISInfo[9]" can be used to do the same.

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*[10] 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.

4 Static Function Call Analysis of Symbian OS Soft- and Malware

4.1 Descriptive Statistics

When extracting function calls we only take into account the function name itself without considering parameters or

[7]http://www.hex-rays.com/idapro/

[8]http://developer.symbian.com/main/tools_and_sdks/developer_tools/critical/unsis/index.jsp

[9]http://www.niksula.cs.hut.fi/~jpsukane/sisinfo.html

[10]http://unixhelp.ed.ac.uk/CGI/man-cgi?readelf+1

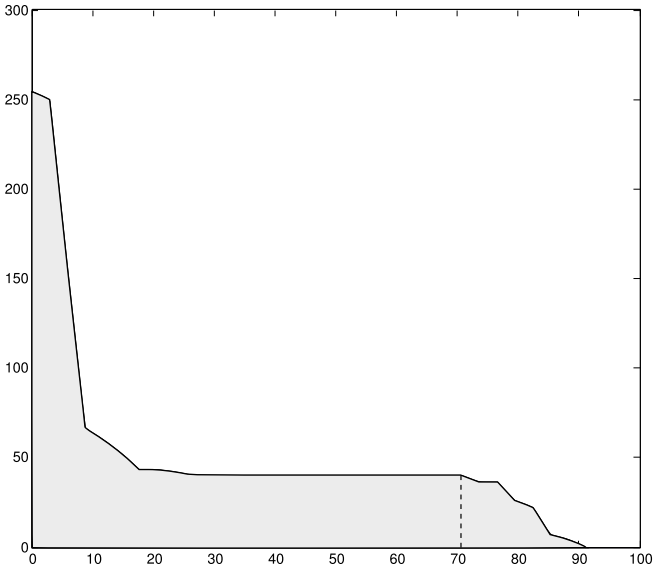

Figure 1. The x-axis displays the percentage of malware, the y-axis displays the number of calls from Ω which appear in the malware. The dotted line reveals that there are about 50 calls which appear in 70% of the malwares.

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

$$\zeta : \begin{array}{ccc} \mathcal{X} & \longrightarrow & \mathcal{P}(\Omega) \\ x & \longmapsto & \zeta(x), \end{array} \tag{1}$$

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 1 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 1 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 1 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 2 and will be called Λ. An attribute $\omega \in \Omega$ is regarded as more malware-typical the greater the quantity gets.

$$t(\omega) = \hat{\mu}_m(\omega) - \big(\hat{\mu}_b(\omega) + 3\hat{\sigma}_b(\omega)\big) \tag{2}$$

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,

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

the empirical standard deviation within the benign class X_b. Three times σ is chosen in (2) since it is a common empirical rule, meaning that for a real random variable almost all

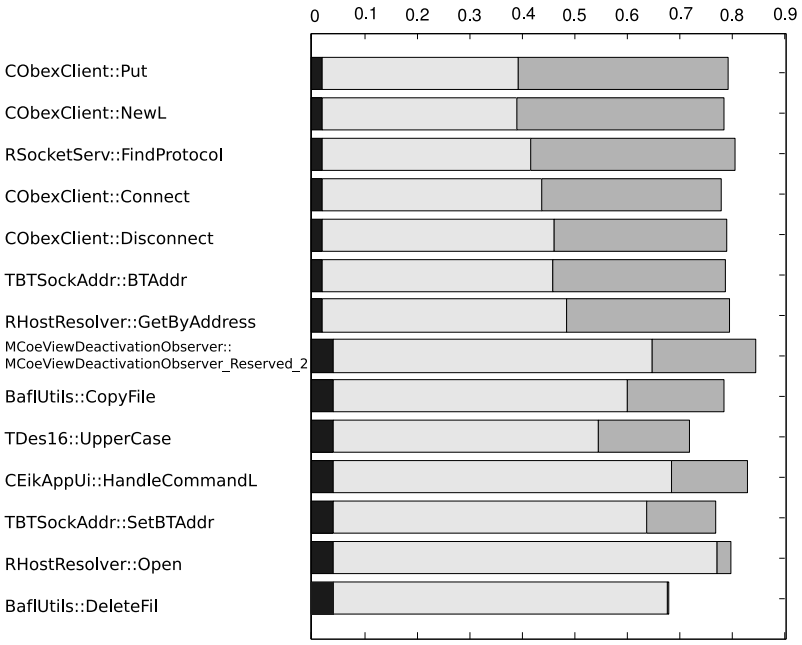

Figure 2. The x-axis shows the attributes of Λ, the y-axis their appearance frequencies: black displays the frequency in benign programs ($\hat{\mu}_b(\omega)$), light-gray the frequency in benign programs plus three times standard deviation ($\hat{\mu}_b(\omega) + 3\hat{\sigma}_b(\omega)$), dark-gray 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.

values lie within 3 standard deviations of the mean [10]. As our final feature set Λ, we picked attributes with the greatest $t(\omega) > 0$ values and $\omega \in \Theta$.

4.2 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 malware on the basis of function calls. This suffices the requirements of mobile devices, i.e. 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 [11] 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 its 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

$$\hat{k}: \begin{array}{ccc} \mathcal{P}(\Omega) & \longrightarrow & \mathbb{R}^{|\Omega|} \\ \mathcal{C} & \mapsto & \sum_{i=1}^{|\Omega|} 1_{\{\omega_i \in \mathcal{C}\}} \overrightarrow{e}_i \end{array} \tag{3}$$

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 (1), we attain a kernel which operates directly on the set of executables $k = k \circ \zeta$. A centroid for a class $j \in \{$'malicious', 'benign'$\}$ is chosen in a way that minimizes the sum of squared euclidean distances d to all data points of her 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.

$$C: \quad x \longmapsto \begin{cases} \text{malicious}, & \text{if } d(k(x), c_m) < d(k(x), c_b) \\ \text{benign}, & \text{else.} \end{cases} \tag{4}$$

Table 2. Statistical figures characterizing the quality of different learning models, excerpt from a 10-fold cross-validation.

Model	Malware Detection Rate	Accuracy	Attribute 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

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

$$\frac{d(k(x), c_m)}{d(k(x), c_b)} < \beta \tag{5}$$

and classified as benign otherwise. This means with increasing β, probability increases that arbitrary checked executables will be detected as malicious. Figure 3 visualizes the aforementioned classification.

5 Results and Discussion

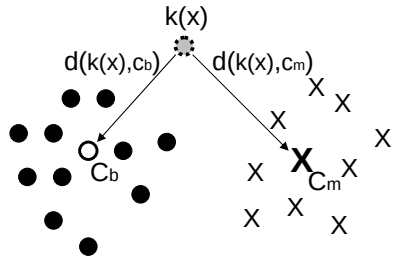

Figure 3. Example clusters of executables with benign center of gravity c_b and malicious c_m in relation to checked executable $k(x)$.

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 Chih-Jen[11] based on the algorithm proposed by Scholkopf et al. [14] with standard parameters. We also varied the SVM parameters, but results did not improve significantly. We used MatLab[12] as learning environment.

On Table 2, 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 the sensitivity varies. The resulting ROC-graph is depicted in Figure 4 while the area under the curve is $AUC = 0.9318$.

Referring to Figure 2, the function calls from Λ[13] point to Bluetooth-based calls giving a good indication for detecting malware. Due to this numbers, we can additionally state that most *sophisticated*[14] Symbian OS malwares obviously use at least Bluetooth for propagation.

Revisiting drawbacks considering static analysis as presented from Moser et al. [8], 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[15], 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. The same applies for the Android platform, while our estimations of course still need to be validated.

6 Conclusion and Future Work

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

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

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

[13] representing the intersected top 14 calls from Symbian OS mal- and software

[14] being more sophisticated than malwares only overwriting files through exploiting features from the Symbian OS installation system

[15] except drivers

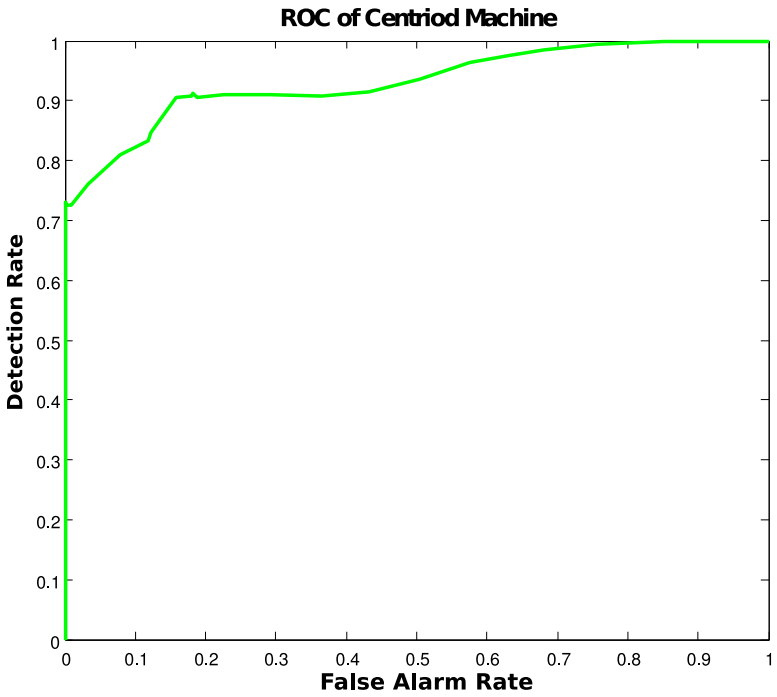

Figure 4. ROC Curve for the centroid machine for varying alarm threshold β from (5). The area under the curve is $AUC = 0.9318$.

malicious from benign program by a learning concept. 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 a attribute reduction method by which we successfully reduced dimensionality dramatically without significant loss of detection quality. We will continue with investigations on following topics:

- Examine different methods for improving classifiers for the requirements of mobile phone.
- Implement a first prototype.
- Extend our analysis on benign programs to create an malware detection system based on unsupervised learning, capable to learn a concept of normality and therefore capable to detect malware, which has been unknown until now.

References

[1] U. Bayer, A. Moser, C. Krügel, and E. Kirda. Dynamic analysis of malicious code function call. *Journal in Computer Virology*, 2(1):67–77, 2006.

[2] 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.

[3] 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.

[4] C. Eagle. *The IDA Pro Book: The Unofficial Guide to the World's Most Popular Disassembler*. No Starch Press, San Francisco, CA, USA, 2008.

[5] M. Egele, M. Szydlowski, E. Kirda, and C. Kruegel. Using static program analysis to aid intrusion detection. In R. Bschkes and P. Laskov, editors, *DIMVA*, volume 4064 of *Lecture Notes in Computer Science*, pages 17–36. Springer, 2006.

[6] F-Secure. Trojan:symbos/yxe, Feb. 2009. `http://www.virus.fi/v-descs/trojan_symbos_yxe.shtml`.

[7] 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.

[8] A. Moser, C. Kruegel, and E. Kirda. Limits of static analysis for malware detection. In *ACSAC*, pages 421–430, 2007.

[9] N. 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.

[10] F. Pukelsheim. The three sigma rule. *The American Statistician*, 48:88–91, 1994.

[11] I. Rish. An empirical study of the naive bayes classifier. Technical report, IBM Research Division, 2001.

[12] A. Schmidt and S. Albayrak. Malicious software for smartphones. Technical Report TUB-DAI 02/08-01, Technische Universität Berlin, DAI-Labor, Feb. 2008. `http://www.dai-labor.de`.

[13] A.-D. Schmidt, R. Bye, H.-G. Schmidt, J. Clausen, O. Kiraz, K. Yüksel, A. Camtepe, and S. 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*, Dresden, Germany, June 2009.

[14] B. Scholkopf, A. J. Smola, R. C. Williamson, and P. L. Bartlett. New support vector algorithms. *Neural Computation*, 12:1207–1245, 2000.

[15] Symantec. Spyware.FlexiSpy, Mar. 2006.

[16] 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.

[17] 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.