A classifier of Malicious Android Applications

Gerardo Canfora, Francesco Mercaldo, Corrado Aaron Visaggio

Dept. of Engineering - University of Sannio

Benevento, Italy

{canfora, visaggio, fmercaldo}@unisannio.it

Abstract – Malware for smartphones is rapidly spreading out. This paper proposes a method for detecting malware based on three metrics, which evaluate: the occurrences of a specific subset of system calls, a weighted sum of a subset of permissions that the application required, and a set of combinations of permissions.
The experimentation carried out suggests that these metrics are promising in detecting malware, but further improvements are needed to increase the quality of detection.

***Index Terms*—security, malware, usability, smartphone, android**

I. INTRODUCTION

Mobile handsets are increasingly becoming full-fledged computing platforms, which are able to execute complete operating systems and a great number of applications. Like PCs, smartphones are exposed to security and privacy threats, some of which are new, because leverage features (GPS, sensitive data like contacts book, agenda and sms, microphone and camera) and limitations (small screen, reduced resources, limited power supply) that are typical of mobile devices. An example of the new security threats that affect smartphones is illustrated in [33], which describes how a trojan can obtain access to the video camera or microphone and steal private images or record conversations, sending them to a dropping server.

The fluidity of the markets [1], rapidly developed and deployed applications [27], coarse permission systems [5], privacy invading behavior [1][6][12], and limited security models [15][24][25] foster the production of malicious applications for mobile phones [10][14][26]. Existing solutions for protecting privacy and security on smartphones are still ineffective in many facets [17], and many analysts warn that the malware variants for Android are rapidly increasing. This scenario calls for new security models and tools to limit the spreading of malware for smartphones.

This paper proposes a method for detecting malware for smartphones, which consists of extracting three metrics from the candidate malware. The first metric computes the occurrences of a set of system calls invoked by the application under analysis. The assumption is that different malicious applications can share common patterns of system calls that are not present in trusted applications, as these common patterns are recurrently used to implement malicious actions.

The second metric computes a weighted sum of all the permissions requested by the application under analysis, where the weight is assigned according to the potential threat that the permission could cause if the application has evil goals. For instance, "send sms" permission could be more dangerous than "receive sms" permission.

The third metric is a weighted sum of selected combinations of permissions. The underlying idea is that specific combinations of permissions can be more effective to detect malware applications rather than a weighted sum of all the permissions. Relevant permission combinations were obtained from a literature analysis about smartphone malware.

The paper poses two research questions:

- RQ1: is a set of permissions able to distinguishing a malware from a trusted application for smartphones?
- RQ2: can the occurrences of a set of system calls be used for discriminating a malware from a trusted application for smartphones?

The paper proceeds as follows: the next section discusses the related work; the following section illustrates the proposed method; the fourth section discusses the experimentation, and, finally, conclusions are drawn in the last section.

II. RELATED WORK

Many threats for smartphones exploit the weaknesses of the permissions based mechanism. One of them is the problem of over-privileged applications [8], [9] which routinely request more permissions than they really need, exposing users to unnecessary permission warnings and increasing the impact of a bug or vulnerability.

Marforio et al. [17] show how permissions-based mechanisms allow attacks by colluding applications that communicate over overt and covert communication channels.

The literature presents many solutions for detecting malicious behaviors of smartphones application. A strong focus of current research concerns privacy leakage. TaintDroid [6] tracks the flow of privacy sensitive data through third-party applications. TaintDroid assumes that downloaded third-party applications are not trusted, and monitors–in realtime–how

these applications access and manipulate users personal data. The primary goal is to detect when sensitive data leaves the system via untrusted applications and to facilitate analysis of applications by phone users or external security services. PiOS [1] and AndroidLeaks [11] perform static analysis for detecting privacy leaks in smartphones, the latter on iOS applications, and the former on Android. The aim is to detect when sensitive information, such as device ID, location and phone number, address book, browser history and photos, is collected by malicious applications.

Decentralized information flow control enhanced operating systems, such as Asbestos [30] and HiStar [34], label processes and enforce access control based on Denning's lattice model for information flow security [1]. Flume [16] provides similar enhancements for legacy OS abstractions. DEFCon [19] uses events and modifies a Java runtime with lightweight isolation. Related to these system-level approaches, PRECIP [32] labels both processes and shared kernel objects such as the clipboard and display buffer.

However, these process-level information flow models do not track sensitive information within untrusted applications. Chandra and Franz [1] propose fine-grained information flow tracking within the JVM and instrument Java byte-code to aid control flow analysis. Similarly, Nair et al. [23] instrument the Kaffe JVM. Vogt et al. [31] instrument a Javascript interpreter to prevent cross-site scripting attacks.

Language-based information flow security [29] extends existing programming languages with labels on variables of security attributes. Compilers use the security labels to generate security proofs, e.g., Jif [21][22] and SLam [13]. Laminar [28] provides decentralized information flow control guarantees based on programmer defined security regions.

OS-level protections have been proposed in Kirin [5], Saint [25], and Security-by-Contract [3] for Android and Windows Mobile. Mulliner et al. [20] provide information tracking by labeling smartphone processes based on the interfaces they access, effectively limiting access to future interfaces based on acquired labels.

The method proposed in this paper has a low level of invasiveness and is easy to implement as it requires only information that could be straightforward retrieved.

III. THE PROPOSED METHOD

The proposed method is designed for extracting two classes of features from a smartphone application: invoked system calls and permissions that the application under analysis requires.

A Linux Kernel has more than 250 system calls identified by a number that can be retrieved by the table of the system calls in the kernel. By capturing and analyzing the system calls that go through the system call interface (glibc), it is possible to get accurate information with regards to the behavior of the application.

Given the high number of system calls, we capture a reduced number of calls that are thought as the ones that could be recurrently used for malicious purposes. The set of the system calls considered in the paper is related to the management of the processes (creation, binaries execution, and termination of processes) and to I/O operations (creation, elimination, writing and reading of file objects).

The first metric that we compute, named *syscall*, counts the occurrence of each system call in the reduced set of system calls we considered.

The other two metrics regard the permissions requested by the application, and are:

- *sumperm*, which is a weighted sum of a subset of permissions, i.e.:

$$sumperm = \sum_{i=1}^{n} p_i * w_i$$

where: p_i is 1 if the i-th permission in table 1 is claimed, 0 otherwise, and w_i is the weight assigned to the permission p_i.

- *risklevel*, which is a weighted sum over specific combinations of permissions, shown in table 2.

PERMISSION	WEIGHT
RECEIVE_BOOT_COMPLETED	3
SEND_SMS	2
WRITE_SMS	2
ACCESS_NETWORK_STATE	1
ACCESS_WIFI_STATE	1
KILL_BACKGROUND_PROCESSES	3
GET_TASKS	2
CALL_PHONE	3
GET_ACCOUNTS	2
RECEIVE_SMS	1
READ_SMS	2
INTERNET	3
READ_PHONE_STATE	2
READ_LOGS	2
RESTART_PACKAGES	1
ACCESS_FINE_LOCATION	2
BLUETOOTH_ADMIN	2
READ_CONTACTS	2

Table 1. The couples (permission, weight) of the subset of permissions used in the metric sumperm.

In table 1, the highest weight (3) is assigned to those permissions that are considered more dangerous for privacy or security, such as: RECEIVE_BOOT_COMPLETED, which can be used for launching malware installer, KILL_BACKGROUND_PROCESSES frequently employed by command and control tools, CALL_PHONE which can be used to start phone calls without the consensus of the smartphone's owner, and INTERNET, which can be used for connecting to the net and can allow the malware to perform malicious actions (send recorded conversations to a drop server, for instance).

The *risklevel* is obtained by enumerating specific combinations of permissions that could be a proxy of a

malicious intent in the application. The combinations listed in Table 2 are derived based on simple heuristics built by analyzing the features of the malware for Android platforms, censed in literature. Each time the combination illustrated in the column "permission's combination" occurs in the application, the risklevel metric is increased of the value in the "risklevel" column in table 2.

PERMISSION's combination	Risklevel
(RECEIVE_SMS ∧ SEND_SMS) ∨ CALL_PHONE	50
((READ_PHONE_STATE ∧ INTERNET) ∨ (GET_ACCOUNTS ∧ INTERNET) ∨ (READ_CONTACTS ∧ INTERNET) ∨ (READ_LOGS ∧ INTERNET))	30
((GET_TASKS ∧ KILL_BACKGROUND_PROCESSES) ∨ (GET_TASKS ∧ RESTART_PACKAGES))	15
((ACCESS_NETWORK_STATE ∨ ACCESS_WIFI_STATE))	5

Table 2. The couples (permissions' combination, risklevel) of the subset of permissions used in the metric risklevel.

IV. The Case Study

The aim of the case of study is to evaluate the effectiveness of the proposed method, expressed through the two research questions RQ1 and RQ2.

A dataset made of 200 trusted and 200 malware Android applications was collected: trusted applications by different categories (call & contacts, education, entertainment, GPS & travel, internet, lifestyle, news & weather, productivity, utilities, business, communication, email & SMS, fun & games, health & fitness, live wallpapers, personalization) were downloaded from the common Android Markets, while malware applications of different nature and malicious intents (premium call & SMS, selling user information, advertisement SMS spam, stealing user credentials, ransom) from public databases of antivirus companies (Symantec, F-Secure, Lookout, Panda).

The malware nature of each application was confirmed by at least two antivirus companies mentioned above.

Two kinds of analysis were performed: hypothesis testing and classification analysis. In the first analysis, the null hypothesis to be tested is:

H_0: "malware and trusted application have similar values of the metrics: syscall, sumperm and risklevel ".

The null hypothesis was tested with Mann-Whithney (with the p-level fixed to 0.05) and with Kolmogorov-Smirnov Test (with the p-level fixed to 0.05).

The classification analysis was aimed at assessing whether the metrics where able to correctly classify malware and trusted software. Five algorithms of classification were used: J48, LadTree ,NBTree, RandomForest, randomTree, RepTree. These algorithms were applied to three groups of metrics. The first group includes only the system calls that rejected the null hypothesis, the second group includes only the two aggregate metrics *sumperm* and *risklevel*, and the third group includes only the *risklevel* metric.

A. Analysis of data

The box plots of those system calls exhibiting a relevant difference between malware and trusted applications, which were *open*, *read*, *write*, and *recv*, are reported in figures 1 and 2.

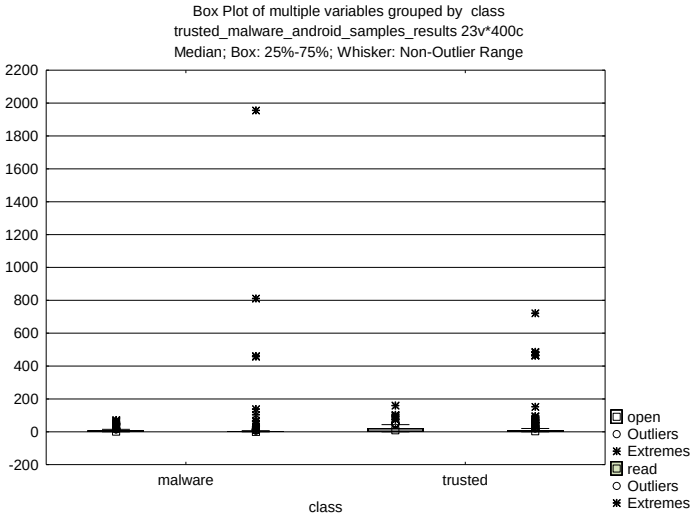


Fig 1. Box-plots for the system calls *open* and *read*

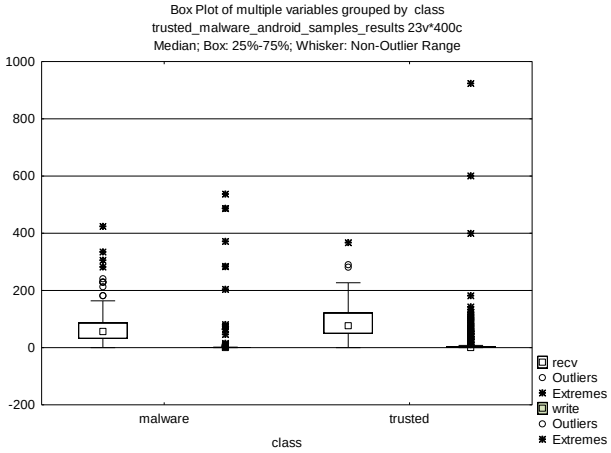


Fig. 2. Box-plots for the system calls *recv* and *write*

The differences between the box plots of trusted and malware applications for the *open*, *read*, *recv*, and *write* system calls suggest that the two populations could belong to different distributions. This hypothesis will be confirmed by the hypothesis testing.

The box plots related to *sumperm* and *risklevel* are represented in figure 3.

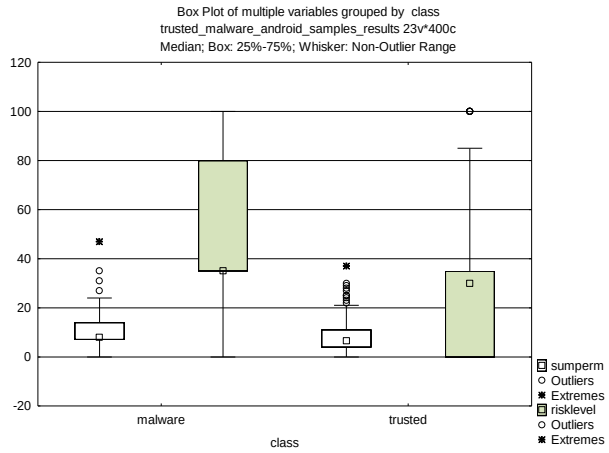


Fig. 3. Box plots of the metrics *sumperm* and *risklevel*.

The distributions of the two metrics for the two populations, i.e. malware and trusted, seem to be very different, thus the two metrics could be used for discriminating a malware application from a trusted application. This conjecture will be confirmed by the results of the hypothesis testing, shown in table 3.

Variable	**Mann-Whitney U Test**	**Kolmogorov-Smirnov Test**
open	0.000000	p < .001
read	0.000001	p < .001
recv	0.000029	p < .001
write	0.000790	p < .025
sumperm	0.000025	p < .001
risklevel	0.000000	p < .001

Table 3. Results of the test of the null hypothesis H_0.

The hypothesis can be rejected for the four system calls *open*, *read*, *recv*, and *write* and the metrics *sumperm* and *risklevel*. This means that there is statistical evidence that these metrics are potential candidates for correctly classifying malware and trusted applications. This conjecture is assessed by the classification analysis, whose results are shown in figures 4, 5, and 6. Three metrics were used to evaluate the classification results: recall, precision and ROC area. The Recall has been computed as the proportion of examples that were assigned to class X, among all examples that truly belong to the class, i.e. how much part of the class was captured. The Precision has been computed as the proportion of the examples that truly belong to class X among all those which were assigned to the class.

The classification analysis suggests several considerations. With regards to the recall:

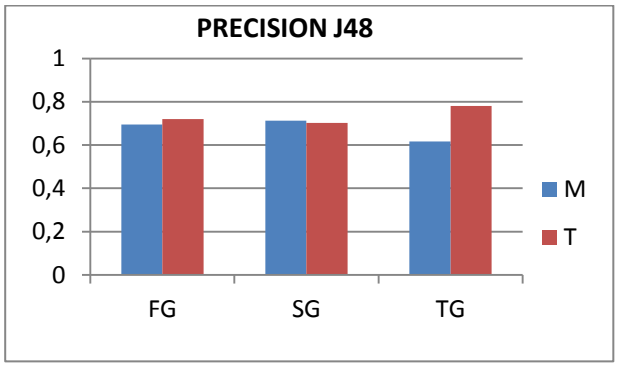
- all the algorithms are able to classify better the malicious applications than the trusted ones;
- the *risklevel* metric is the most effective to increase the recall of the classification, i.e. increasing the portion of correctly classified instances.

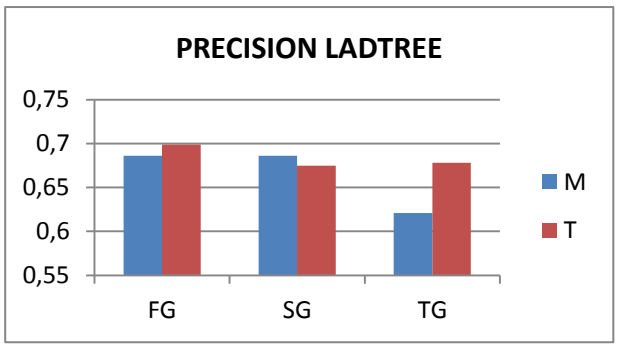
With regards to the precision:

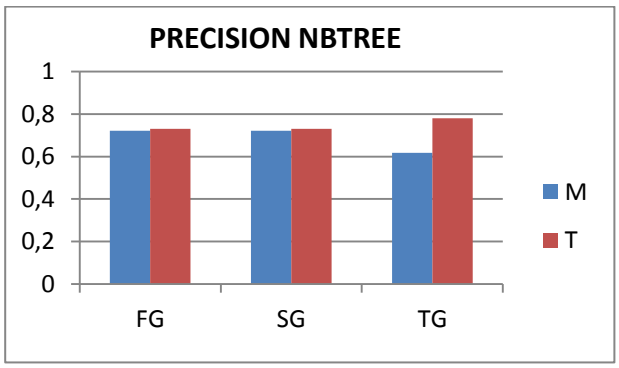
- the algorithms seem to classify better the trusted applications than the malicious ones, even if the difference is small;
- the *risklevel* metric seems to decrease the precision, so it allows the proliferation of misclassified instances.

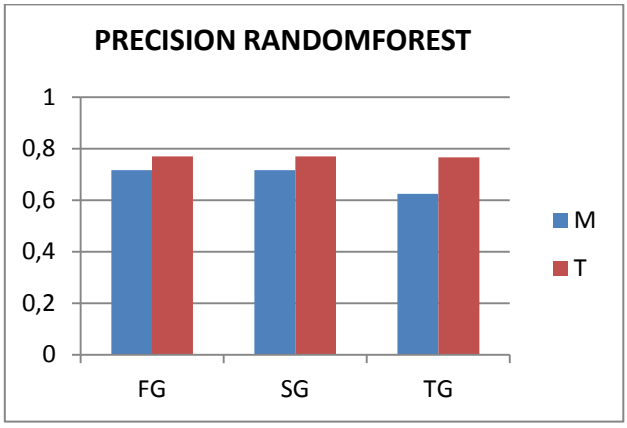
With regards to the roc area:

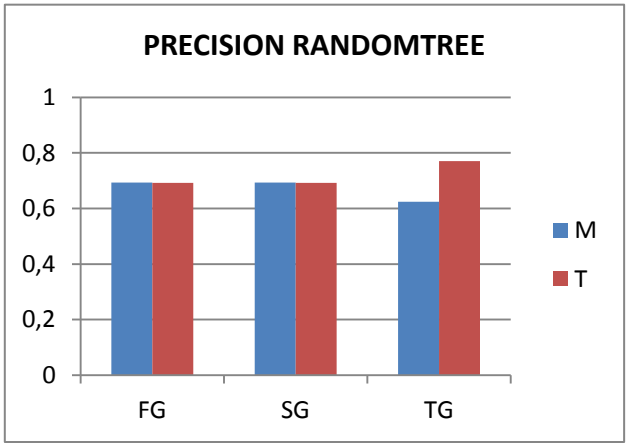
- the performances of all the algorithms are pretty the same for malware and trusted applications;
- the *risklevel* metric has the lowest values of roc-area.

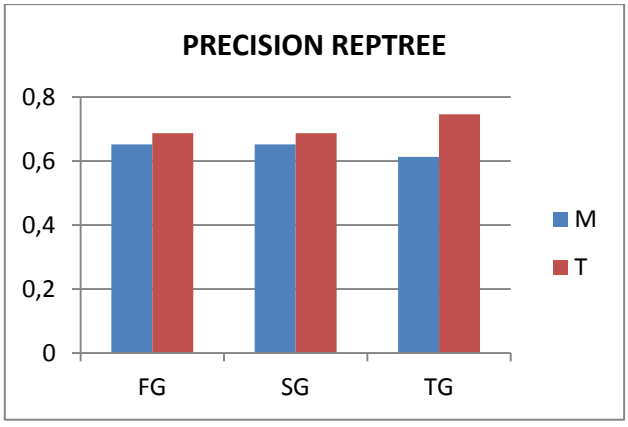


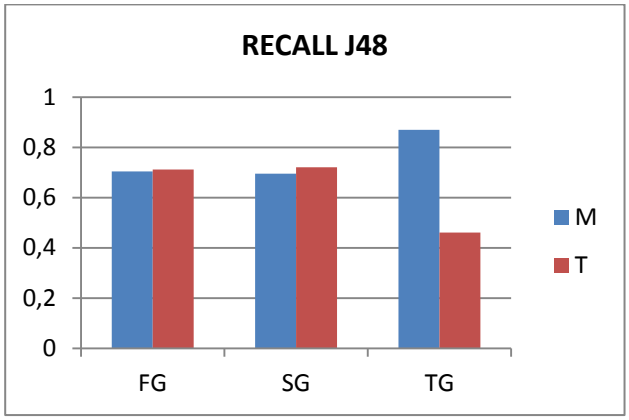


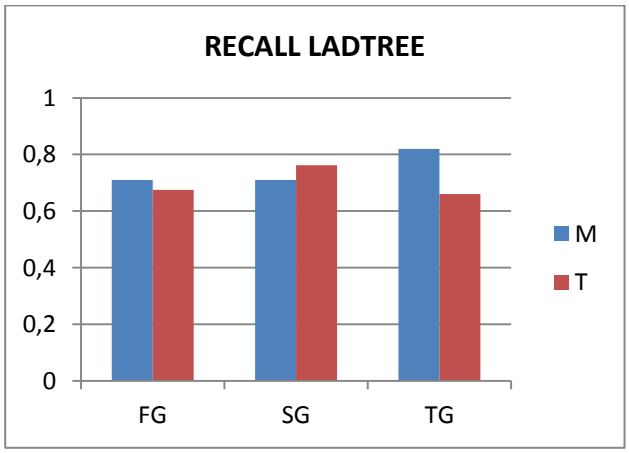


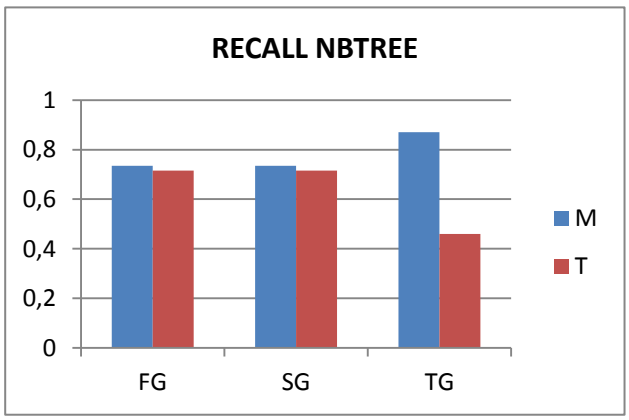












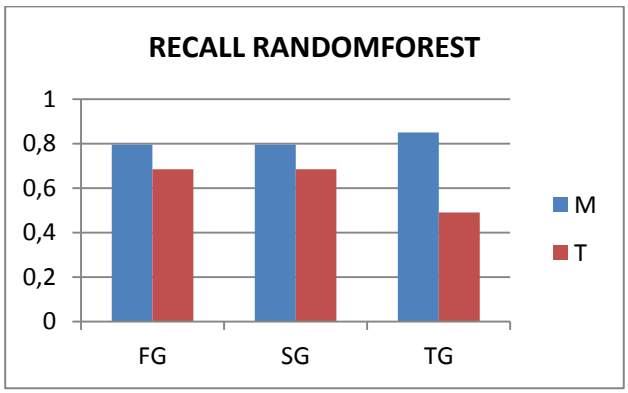


Fig 4. The values of precision for classifying Malware (M) and Trusted (T) applications, calculated with the metrics of the First Group (FG), the Second Group (SG), and the Third Group (TG), with the algorithms J48, LadTree, NBTree, RandomForest, Randomfree and RepTree

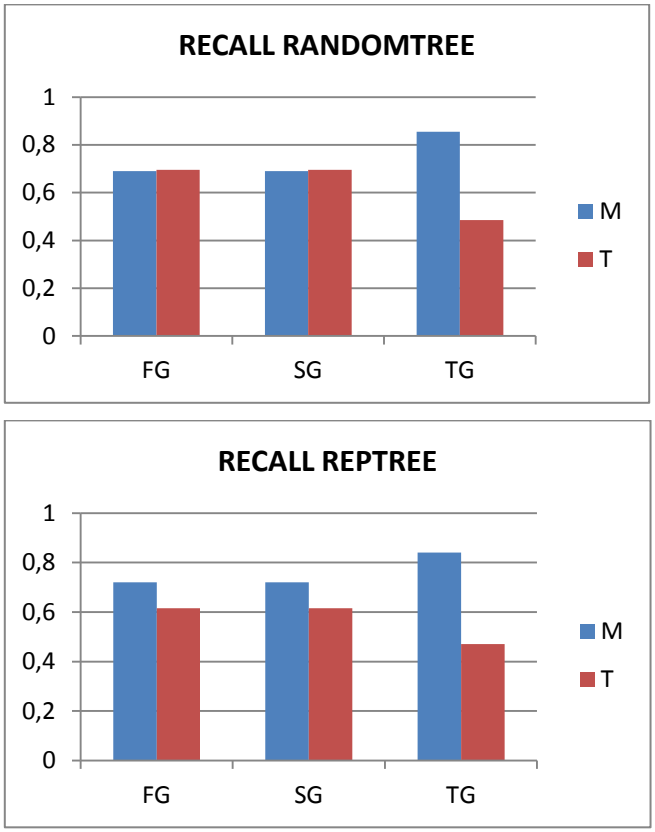
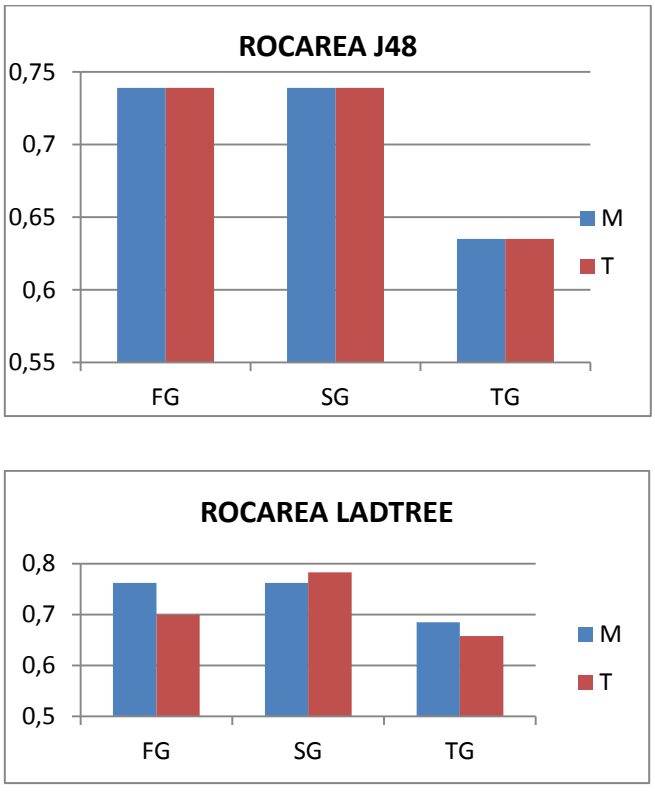


Fig 5. The values of recall for classifying Malware (M) and Trusted (T) applications, calculated with the metrics of the First Group (FG), the Second Group (SG), and the Third Group (TG), with the algorithms J48, LadTree, NBTree, RandomForest, Randomfree and RepTree.



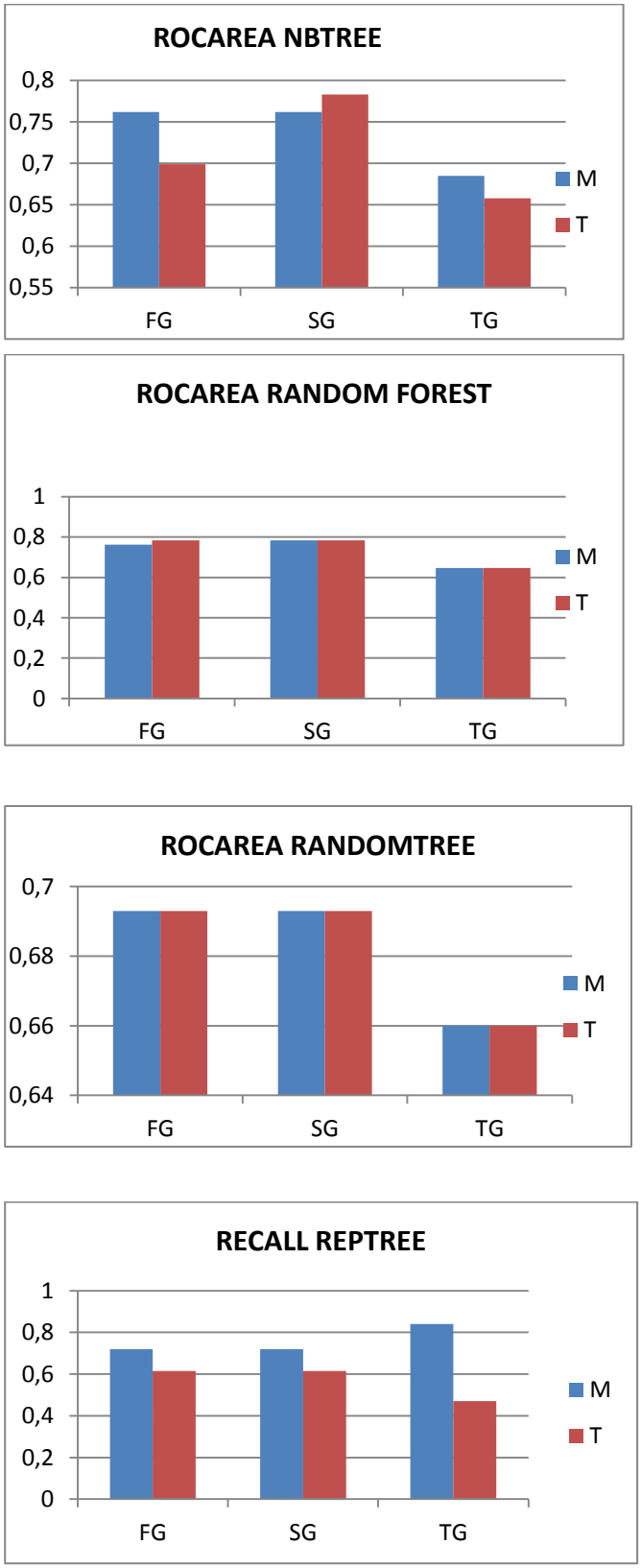


Fig 6. The values of the ROCArea for classifying Malware (M) and Trusted (T) applications, calculated with the metrics of the First Group (FG), the Second Group (SG), and the Third Group (TG), with the algorithms J48, LadTree, NBTree, RandomForest, Randomfree and RepTree.

In summary, the experiment shows that *syscall* and the combination of *sumperm* and *risklevel* produce similar results, while using *risklevel* alone leads to less stable results. As a matter of fact, while *syscall* and the couple (*sumperm*, *risklevel*) allow for classifying malware and trusted application with comparable levels of precision, recall and ROC area, *risklevel* alone presents two limitations. The first limitation is that the capability of recognizing malware greatly differs from the capability of recognizing trusted applications among the algorithms used. The second limitation is that precision and recall for *risklevel* is often below 50%, which corresponds to the random classification.

V. CONCLUSIONS AND FUTURE WORK

The wide diffusion of smartphones and their usage for a plethora of activities, like surfing internet, controlling mail, playing, connecting people on social networks, reading and modifying (different kinds of) documents, and so on is fostering the diffusion of malware, too.

Malware for smartphones shows the traditional features that the malware for personal computers has, adding new threats for privacy and security, as they leverage the peculiar characteristics of smartphones (GPS, sensitive data like contacts book, agenda and sms, microphone and camera) and limitations (small screen, reduced resources, power supply).

Current solutions for detecting malware on smartphones are still ineffective: it urges to provide new and successful methods and tools for contrasting the rapid spreading of malware.

This paper has proposed a technique for detecting malware for smartphones which relies on three metrics, which compute: the occurrences of a reduced set of system calls invoked by the applications and two metrics which are obtained by weighting and combining the permissions the application requires.

It emerged from the experimentation carried out that the performance of *syscall* and those of the couple of metrics (*sumperm*, *risklevel*) are comparable: both are effective for successfully detecting malware. Data of the experiment do not let to state the similar conclusions for *risklevel*, whose detection performances are poor.

The point of strength of these metrics is that they can be obtained straightforward, thus they can be implemented easily into an antimalware system.

The empirical study's possible threats to validity concern the conclusion and construct validity.

Threats to conclusion validity: the only issue that could affect the statistical validity of this study was the size of the sample data, which is limited to 400 applications.

The experiments suggest that the research direction is promising, but the precision and recall must be improved. This can be done by implementing further heuristics, which could strengthen the two metrics. We are aware of this, so the results of this experiment must be considered as preliminary findings for further studies.

Threats to construct validity. We used metrics which rely on weights assigned discretionally by the authors on their experience in analyzing malware. A similar consideration stands for the permission's combinations of the *risklevel* metric. Of course, the metrics can be improved by enlarging the malware analysis and the consequent knowledge on the current malware's features.

A future development of the research will be focused on the identification of precise patterns and sequences of system calls that could be recurrent in malicious malware's code fragments, in order to improve the quality of detection.

REFERENCES

[1] D. Chandra, and M. Franz. Fine-Grained Information Flow Analysis and Enforcement in a Java Virtual Machine. In *Proc. of the 23rd Annual Computer Security Applications Conference (ACSAC)* (December 2007).

[2] D. E. Denning. A Lattice Model of Secure Information Flow. *Communications of the ACM 19, 5* (May 1976), 236–243.

[3] L. Desmet, W. Joosen, F. Massacci,P. Philippaerts, F. Piessens, I. Siahaan, and D. Vanoverberghe. Security-by-contract on the .NET platform. Information Security Technical Report 13, 1 (January 2008), 25–32.

[4] M. Egele, C. Kruegel, E. Kirda, and G. Vigna. PiOS:Detecting Privacy Leaks in iOS Applications. In *Proc. ofthe Network and Distributed System Security Symposium* (2011).

[5] W. Enck, M. Ongtang, and P. McDaniel. On Lightweight Mobile Phone Application Certification. In *Proc. of the 16th ACM Conference on Computer and Communications Security (CCS)* (Nov. 2009).

[6] W. Enck, P. Gilbert, B.-G. Chun, L.P. Cox, J. Jung, P. McDaniel, and A.N.Sheth. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. In *Proc. of the USENIX Symposium on Operating Systems Design and Implementation* (2010).

[7] W. Enck, D. Octeau, P. Mcdaniel, S. Chaudhuri, A study of android application security. In *proc. Of 20th USENIX conference on Security, SEC'11*, pg: 21-22, Usenix Association Berkley(2011).

[8] A. P. Felt, E. Chin, S. Hanna, D. Song. D. Wagner. Android permissions demystified. In the *proc. of the 18th ACM conference on Computer and communications security, CCS 11*, ACM, 627-638, (2011)

[9] A. P. Felt, K. Greenwood, D. Wagner, The effectiveness of application permissions. In *proc. of the 2nd USENIX conference on Web application development, WebApps'11*, USENIX . Berkely, association (2011).

[10] FIRST TECH CREDIT UNION. Security Fraud: Rogue Android Smartphone app created. http://www.firsttechcu.com/home/security/fraud/security_fraud.html, (Dec. 2009).

[11] C. Gibler, J. Crussell, J. Erickson, H. Chen, AndroidLeaks: Automatically Detecting Potential Privacy Leaks in Android Applications on a Large Scale. *Trust and Trustworthy Computing Lecture Notes in Computer Science Volume 7344* (2012), 291-307

[12] GOODIN, D. Backdoor in top iphone games stole user data, suit claims. The Register, November 2009. http://www.theregister.co.uk/2009/11/06/iphone_games_storm8_lawsuit/.

[13] N. Heintze, and J. G. Riecke, The SLam Calculus: Programming with Secrecy and Integrity. In *Proc. of the Symposium on Principles of Programming Languages (POPL)* (1998), 365–377.

[14] KASPERSKEY LAB. First SMS Trojan detected for smartphones running Android. http://www.kaspersky.com/news?id=207576158, (August 2010)

[15] N. Kralevich. Best Practices for Handling Android User Data. http://android-developers.blogspot.com/2010/ 08/best-practices-for-handling-android.html (2010).

[16] M. Krohn, A.Yip, M.Brodsky, N. Cliffer, M:F: Kaashoek, E. Kohler, and R. Morris. Information Flow Control for Standard OS Abstractions. In *Proc. of ACM Symposium on Operating Systems Principles* (2007).

[17] C. Marfofio, F., aurelien, C. Srdjan, Application Collusion Attack on the Permission-Based Security Model and its Implications for Modern Smartphone Systems, Department of Computer Science, ETH Zurich, Technical Report (2011).

[18] P. McDaniel, and W. Enck,. Not So Great Expectations: Why Application Markets Haven't Failed Security. *IEEE Security & Privacy Magazine 8, 5* (September/October 2010), 76–78.

[19] M. Migliavacca, I. Papagiannis, D.M. Eyers, B.Shand,J. Bacon, and Pietzuch, DEFCon: High-Performance Event Processing with Information Security. In *Proc. of the USENIX Annual Technical Conference* (2010).

[20] C. Mulliner, G. VIGNA, D. DAGON, and W. LEE. Using Labeling to Prevent Cross-Service Attacks Against SmartPhones. *In Proc. of Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)* (2006).

[21] A.C. Myers. JFlow: Practical Mostly-Static Information Flow Control. In *Proc. of the ACM Symposium on Principles of Programming Langauges (POPL)* (January 1999).

[22] A.C. Myers, and B. LISKOV. Protecting Privacy Using the Decentralized Label Model. *ACM Transactions on Software Engineering and Methodology 9, 4* (October 2000), 410–442.

[23] S.K. Nair, P. Simpson, B. Crispo, and A.S. Tanenbaum. A Virtual Machine Based Information Flow Control System for Policy Enforcement. In the *proc. the 1st International Workshop on Run Time Enforcement for Mobile and Distributed Systems (REM)* (2007).

[24] M. Ongtang, K. Butler, and P. McDaniel. Porscha:Policy Oriented Secure Content Handling in Android. In *Proc. of the Annual Computer Security Applications Conference* (2010).

[25] M. Ongtang, S. McLaughlin, W. Enck, and P. McDaniel. Semantically Rich Application-Centric Security in Android. In *Proc. of the Annual Computer Security Applications Conference* (2009).

[26] P. Porras, H. Saidi, and V. Yegneswaran. An Analysis of the Ikee.B (Duh) iPhone Botnet. Tech. rep., SRI International, Dec. 2009. http://mtc.sri.com/iPhone/.

[27] J. Raphel, Google: Android wallpaper apps were not security threats. Computerworld (August 2010).

[28] I. Roy, D. E. Porter, M.D. Bond, K. S. McKinley, and E. Witchel. Laminar: Practical Fine-Grained Decentralized Information Flow Control. In *Proc. of Programming Language Design and Implementation* (2009).

[29] A. Sabelfeld, and A. C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communication 21, 1* (January 2003), 5–19.

[30] S. Vandebogart, P. Efstathopoulos, E. Kohler, M. Krohn, C. Frey, D. Ziegler, F. Kaashoek, R. Morris, and D. Mazi `Eres, D. Labels and Event Processes in the Asbestos Operating System. *ACM Transactions on Computer Systems (TOCS) 25, 4* (December 2007).

[31] P. Vogt, F. Nentwich, N. Jovanovic, E. Kirda, C. Kruegel, and G. Vigna. Cross-Site Scripting Prevention with Dynamic Data Tainting and Static Analysis. *In Proc. Of Network & Distributed System Security* (2007).

[32] X. Wang, Z. LI, N. LI, and J.Y.Choi. PRECIP: Towards Practical and Retrofittable Confidential Information Protection. In *Proc. of 15th Network and Distributed System Security Symposium (NDSS)* (2008).

[33] N. Xu, F. Zhang, Y. Luo, W. Jia, D. Xuan, and J. Teng.Stealthy video capturer: a new video-based spyware in 3g smartphones. In *WiSec '09: Proceedings of the second ACM conference on Wireless network security*, pages 69–78, New York, NY, USA, 2009. ACM.

[34] N. Zeldovich, S. Boyd-Wickizer, E. Kohler, and D. Mazi `Eres. Making Information Flow Explicit in HiStar. In *Proc. of the 7th symposium on Operating Systems Design and Implementation (OSDI)* (2006).