Towards a Hybrid Intrusion Detection System for Android-based PPDR Terminals

Pedro Borges[1], Bruno Sousa[1,2], Luis Ferreira[2], Firooz B. Saghezchi[3], Georgios Mantas[3], Jose Ribeiro[3], Jonathan Rodriguez[3], Luis Cordeiro[2], Paulo Simoes[1]

[1]University of Coimbra, Portugal
[2]OneSource, Consultoria Informatica Lda. Coimbra, Portugal
[3]Instituto de Telecomunicações, Campus Universitário de Santiago, 3810-193 Aveiro, Portugal

***Abstract*—Mobile devices are used for communication and for tasks that are sensitive and subject to tampering. Indeed, attacks can be performed on the users' devices without user awareness, this represents additional risk in mission critical scenarios, such as Public Protection and Disaster Relief (PPDR). Intrusion Detection Systems are important for scenarios where information leakage is of crucial importance, since they allow to detect possible attacks to information assets (e.g., installation of malware), or can even compromise the security of PPDR personnel. HyIDS is an Hybrid IDS for Android and supporting the stringent security requirements of PPDR, by comprising agents that continuously monitor mobile device and periodically transmit the data to an analysis framework at the Command Control Center (CCC). The data collection retrieves resource usage metrics for each installed application such as CPU, memory usage, and incoming and outgoing network traffic. At the CCC, the HyIDS employs Machine Learning techniques to identify patterns that are consistent with malware signatures based on the data collected from the applications. The HyIDS's evaluation results demonstrate that the proposed solution has low impact on the mobile device in terms of battery consumption and CPU/memory usage.**

I. Introduction

Nowadays, users devices are more than simple mobile phones, they have the processing power of a small computer and provide diverse functionalities such as document processing, photo editing, pay wallet among others. In addition, they are capable of using multiple technologies to connect to the Internet, such technologies can include GSM, LTE, WiFi, Bluetooth, NFC. The proportion of mobile phones with broadband access globally is 47% and there is a clear tendency for mobile broadband access ratios [1]. Android's market share ranges from 70% to 80% [2]. Additionally, the advances in LTE, mainly to support mission critical functionalities of Private Mobile Radio (PMR) networks such as group and air encryption communications also increase the possibility of using personal devices of PPDR (Public Protection Disaster Relief) users in such scenarios. Indeed, Bring Your Own Device (BYOD) can further enhance situation awareness by collecting images, videos, transmitting information of biosensors. Nonetheless, it also introduces additional security risks [3].

BYOD devices, managed by personal users represent a security threat to the security model that is employed in PMR networks, where the network and mobile terminals are managed by a single entity. The applications installed in BYOD can be malware, or other kind of malicious applications that aim to allow hackers to meddle sensitive information stored in mobile devices (e.g., credit card information, keys for authentication). These attacks come in several forms and ways and compromise the device by disrupting the operating system and network service, inducing data and financial loss and leaking private information. While the infection rate in Android devices fell from 2014 to 2015 the attackers are quick to take advantage of vulnerabilities and just in the first half of 2015 the number of Android malware samples more than doubled with most of the attacks being spyware [4].

Host Intrusion Detection Systems (IDS) are employed as a solution to perform single host monitoring of several assets such as network traffic (e.g., web-sites visited, secure communications in place), system logs, running processes, application activity, file access and modification, and system or application configuration changes. During monitoring, the characteristics are analysed to detect possible violations or possible threats of security policy violation [5] [6]. Anomalous events detection can rely on several mechanisms: (1) signature-based detection which compares features to identify known threats; (2) anomaly-based detection, which compares patterns of normal activities with suspicious behaviour to determine if significant deviations occured; and (3) stateful protocol analysis that compares predetermined profiles of benign protocol interaction against observed events to identify deviations. Upon detection of an intrusion event, such systems perform several important features: gather detected event information and forward it to a logging server; notify the secure system's administrator through an alert; produce monitored events summaries or provide further detail on events of interest; and put in place intrusion prevention/protection policies.

Nevertheless, such systems are not suited for mobile device use: they require high processing power to perform analysis of several criteria. Even though mobile devices are becoming more powerful, there are energy constraints that are tightly linked with device usability; they are not scalable for mobile devices because they have no coordination and control mechanism; they are not prepared to collect features and metrics inline with the information that is on the mobile device.

This paper presents and validates a Hybrid architecture for an Intrusion Detection System – HyIDS. It enables the identification of malware and security-threatening events by relying on data collection mechanisms, correlation mechanisms and Machine Learning (ML) algorithms. The HyIDS architecture is designed to run on the majority devices, without requiring root access, to collect data that is provided by the logging services and Android APIs. Such information, is securely transmitted to a Command Control Center (CCC) to be analysed to reveal malware behaviour – Dynamic analysis. HyIDS also supports Static Analysis by correlating mobile device application's permission information. The evaluation results demonstrate that HyIDS has a neglectable impact in the data collection process, in terms of CPU, memory and battery usage.

The rest of this paper is organized as follows. Firstly, Section II describes the related work and Section III describes the architecture of the HyIDS solution. Secondly, in Section IV we present the evaluation methodology to assess the designed solution's impact in terms of resource usage. Thirdly, in Section V we analyse and discuss the results work that need to be performed to further improve the solution. Finally, in Section VI we conclude the paper and present our future work.

II. Related Work

There are several techniques which can be employed to mitigate the security threats of Android malware. They can either be classified as static or dynamic techniques. The static analysis approaches were the first to be used in Android and aim to analyse the code without executing it. They disassemble and decompile the application and try to recreate its algorithms through reverse engineering techniques. The requested permissions' analysis is a common method of static analysis; nevertheless, this method is not accurate since it relies on the specification that developers perform, which may request permissions that are not needed for the designed application.

Static analysis methods are quick but can be fooled by code obfuscation or transformation techniques. For instance, AndroSimilar [7], is an automated framework that extracts unique statistical features from the applications and uses them to build a signature database to compare with known malware signatures. Drebin [8] is another approach that performs geometrical analysis using ML techniques to identify malicious applications based on the features that are extracted from the applications' code. Google supports this type of analysis for all applications that are installed from the play store and any external sources [9]. Nonetheless, this type of analysis, in some cases requires source code access and does not detect usage patterns, or other actions performed by the users.

Dynamic approaches usually rely on the collection of data from the devices (CPU usage, network traffic, battery consumption, etc.) which is examined using data mining methods to extract patterns or behaviours that represent normal or abnormal activities. These patterns are combined and used to create rules that detect known and unknown malicious activities through similarities with the well known existing-patterns. Dynamic approaches can monitor the behaviour of programs during runtime using heuristic methods, therefore the problems related to malware which uses obfuscation methods to bypass the static analysis are solved. Nevertheless, dynamic analysis also fails when malware employs evasion techniques. DroidScope [10] is an off device emulation based analysis framework which uses a taint analysis component to track and monitor how malware obtains and leaks sensitive information at the machine code level. Andromaly [11] proposes a user-configurable framework to help users detect suspicious activities on their devices. It collects processor, network and battery data and analyses it to generate notifications that alert to the possibility of malicious behaviour on the device. This approach only runs on the device and is limited to the simple identification of malware that is installed on the device.

Hybrid approaches were conceived to overcome the limitations of static and dynamic approaches, as hybrid schemes aim to combine and complement the information collected by each to detect malware in a more accurate and robust fashion. DroidRanger [12] has two detection engines, the first uses information collected from malware pre-processing and the second uses heuristics to recognize suspicious behaviours and misused features in applications. A suspicious application is examined manually and its characteristics are added to the first detection engine. Spreitzenbarth et al. [13] present a mobile-sandbox system where the application's Android Manifest is parsed and matched against a virus database. The application's bytecode is examined to detect malicious pieces of code that are used to extract sensitive information with the help of *TaintDroid* [14], and *DroidBox* [15] to trace the native calls.

ML techniques can be used to develop malware detection mechanisms based on data collected from applications running on the device. There are three main categories of algorithms in ML: supervised learning, unsupervised learning, and semi-supervised learning. Unsupervised or semi-supervised learning algorithms are preferred to their supervised counterparts for IDS design since (i) having labelled data for all existing malwares is impractical since there are always some unknown ones that the designer might be unaware about them; and (ii) supervised learning can only detect attacks that have representative examples in the training set.

Our solution, HyIDS is an hybrid approach that supports static and dynamic analysis by collecting information related to applications and their respective resource consumption in an integrated approach that combines ML for malware detection and correlation engine to correlate events to determine if security-policies are followed. Contrary to many dynamic and hybrid approaches, HyIDS does not perform any kind of analysis on emulators, since some malware can detect such and avoid detection by limiting its execution. Also, the operator on the CCC for the PPDR situation management manually responds to the suggestions of the IDS to enforce any management actions on the devices. Furthermore, HyIDS takes into account high level events caused by the device user that may indicate misusage of the device which can compromise a PPDR operation.

III. Hybrid IDS Architecture

The HyIDS consists of several components, shown in Figure 1. Some on the mobile device, and others at the infrastructure side, for instance at the CCC in a PPDR scenario. The former collects and reports data while the latter handles, stores, and performs malicious actions detection on the device. The mobile component for intrusion detection also includes the analysis of files and security configurations, as well as attempts of privilege escalation. The components running on the mobile device and the components in the CCC exchange information over Transport Layer Security (TLS) for the secure transmission of the collected data across the network.

A. Mobile Device Components

The mobile device application consists of three modules: the monitoring; the communication and control; the mobile correlator. For the sake of simplification, from this point on, the mobile device components will be called MDC.

Communication and control module: This component is responsible for periodically gathering and reporting the data collected by the monitoring module to the CCC. It is also responsible for performing control operations on the device that are triggered from either the CCC or the Mobile correlator module. The supported operations aim to enforce the placement of security policies and may include the following actions: installation of applications; removal of unauthorized applications (i.e., not allowed in a PPDR context, represent a security risk); device locking through a randomized PIN that is dynamically modified if brute-force attacks are detected; and device data wiping when disclosure of sensitive information is verified. This module is also used for mobile correlator module rule configuration.

Monitoring module: This component is responsible for mobile device data collection from system APIs (CPU, memory usage, network traffic input/output, GPS), content observers and broadcast receivers (installation/uninstallation of applications, outgoing and incoming calls, sent and received messages, visited websites, battery status). It also monitors the file access and usage namely to detect unwanted modifications or attempts of privilege escalation. All the collected information is used afterwards by the ML module and correlator modules on the CCC to assess if the user or any of mobile device's applications represent a security threat, either by performing data leaks, or by performing malicious actions.

Mobile correlator module: This component is responsible for performing the correlation of simple events according to rules and application permissions. This way, this module assures that the most important security assets can be analysed, considering the trade-off of the impact of the analysis in the overall mobile device resource usage.

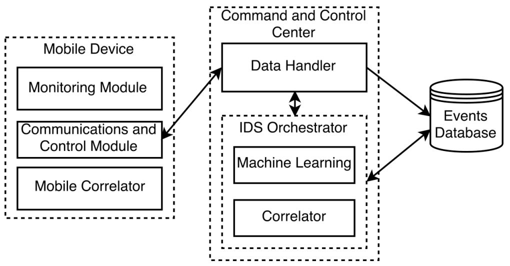

Fig. 1: Architecture of Hybrid HIDS

B. Command Control Center Components

The HyIDS architecture at the Command Control Center or server side, includes a data handler component to support a scalable process of data collection (e.g., from the order of thousands to millions of devices); an Orchestration module that combines the ML analysis algorithm and correlation to detect malicious behaviour.

Data Handler: This module stores the data gathered from all devices and also malicious application signature and behaviour information. This dataset is used by the IDS Orchestrator and other components to detect malicious actions. Such process was decoupled from the orchestrator to enable scalable deployments of the HyIDS solution, for instance to support private cloud deployments by PPDR organizations.

Correlation Engine: The correlation engine on the CCC is based on the Esper Engine [16]. Esper is an event series analysis and Complex Event Processing (CEP) system that combines and examines information from several data stream sources to detect events or patterns indicating the occurrence meaningful events. Correlation is used, for instance in forensics analysis [17] to help analysts sift through large amounts of data in order to gather evidence of malicious action posthumously. The correlator module in HyIDS leverages from the advantages of CEP to process several events and detect correlations that may correspond to security attacks. The events that are correlated include: unauthorized calls (i.e., calls to numbers that are not assigned to PPDR organizations or health organizations - hospitals), call duration and frequency, unauthorized messages, installation of malware flagged applications, wrongful device unlock attempts, removal of applications, and geographical location of users, especially in an emergency scenario. For instance, an SMS sent from an application different from the default SMS application may indicate the presence of malware. Whenever the events correlation detects suspicious behaviour, an alert is generated, considering the level of severity that is determined.

Machine Learning: We use two learning algorithms for IDS. The first one is called Anomaly Detection algorithm, which is a semi-supervised learning algorithm and is based on statistical methods. More specifically, it models the normal behaviour of the system as a Gaussian Mixture model, and then detects outliers that occur in the tail of the distribution [18]. The second algorithm that we use is the K-means clustering algorithm, which is an unsupervised learning algorithm [19,20]. The rational behind using Anomaly Detection algorithm is that most of the collected data represents the *normal* behaviour although we have few training examples representing the anomalous behaviour. On the other hand, K-means clustering algorithm is suitable since (i) there is no need for labelled data; and (ii) we know a priori that we have only two clusters: normal and malicious.

Feature Selection: Representative features are memory usage, CPU consumption, battery consumption, and network traffic per device or per each application as well as *system calls* that the Operating System receives from each application. Let dataset $\{x^{(1)}, ..., x^{(m)}\}$ represent the normal behaviour of the device sampled in regular intervals, say every 2s.

Anomaly Detection Algorithm: Following are four steps for Anomaly Detection algorithm.

1) Choose features x_i that might be indicative of anomalous examples.
2) Fit parameters $\mu_1, ..., \mu_n, \sigma_1^2, ..., \sigma_n^2$ as follows:
$$\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)} \; ; \; \sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(x_j^{(i)} - \mu_j)^2 \quad (1)$$
3) Given a new example x, compute $p(x)$:
$$p(x) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2) = \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi}} e^{-(x_j-\mu_j)^2/2\sigma_j^2} \quad (2)$$
We assume that x_js are iid (independent and identically distributed) random variables with distribution $\mathcal{N}(\mu_j, \sigma_j^2)$.
4) Flag anomaly if $p(x) < \epsilon$.

K-means Clustering Algorithm: Clustering and the K-Means algorithm are common ML practices for anomaly detection. Since we classify the behaviour of any malware as either benign or malicious, the clustering algorithm will end up in two clusters, so K=2 [11,21]. In general, K-Means algorithm randomly initializes K points, called cluster centroids, $\mu_1, ...\mu_n \in \mathbb{R}^n$. Then, it iteratively performs the following two steps until the cluster centroids do not change any more:

1) assigns every training example to the cluster whose centroid has the closest distance to it; i.e., $c^{(i)} \leftarrow \arg \underset{k}{min} \; \| \; x^{(i)} - \mu_k \; \|^2$, where $c^{(i)}$ is the index of the associated cluster;
2) move each cluster centroid to the sample mean of its assigned examples.

IDS Orchestrator: This component includes the ML and correlator components, previously described, and additionally includes an orchestration component, mainly for synchronization regarding the action to perform considering the suggested security levels for malicious behaviour either from the ML algorithm or from the correlator engine. For such, the *LOW, MEDIUM, HIGH* security levels are defined, which lead to different actions. For instance, the MEDIUM and HIGH levels might lead to a lock action if suspicious behaviour from the user is detected, or a wipe action is suggested if malware is being installed to avoid information leakage.

IV. Hybrid IDS Training and Evaluation

This section presents the methodology followed to perform the ML component training and to assess the Hybrid IDS architecture MDC performance.

A. Machine Learning Training

To assess the Hybrid IDS efficiency to detect malware, four scenarios were devised: The first without malware – **noMalware**, the second with the first malware sample – **withMalware1**, the third with the second malware sample – **withMalware2**, and finally the fourth with the combination of both malware samples – **withMalware1and2**. The malware samples that were installed in the mobile phone include Pincer [22] and Hehe [23], these are known to have the following effects: Pincer is able to forward SMS messages to a server with the device's phone number, device serial number, device model, and OS version; Hehe registers itself on a server, then monitors incoming phone calls and SMS messages. It is able to intercept and block calls and messages. These can be forwarded to a malicious server, after which the malware cleans up any trace of the communications from the device.

The scenarios use the same applications, which include the Google Chrome web browser, the GMail email client, and the Patience game application. Each scenario involves several data collection runs with a period of 5 minutes, on which such applications are used. In each of these runs, the data collection mechanism retrieves information for all the applications on the device that are using the CPU, memory or network interface.

The automation process for the data collection tests has been performed using UI Automator [24]. It allowed us to stimulate the Android system and its applications by triggering touches on specific application components, system buttons, and screen locations. Specific stimulation scenarios with the applications were developed to replicate the actions performed on a mobile device (e.g., sending an e-mail, visiting a website, playing a card game). This way, by making the device behave like it is being used, we collect data similar to a real world usage pattern to use it for training the ML algorithm.

B. Evaluation Scenarios

The HyIDS was evaluated on a performance level, with the goal to assess the resource usage impact.

Two scenarios were used to assess the HyIDS application performance on the mobile phone: **OHyIDS** - where the HyIDS is the only running application on the device; **AHyIDS** - where the HyIDS app is running on the device with other applications running in the background. Such applications are equal to those used in the ML training (cf. IV-A).

Two mobile phones with different characteristics were used to assess the impact of the data collection process. The first device is a Google Nexus 5 with 2 GB of RAM with a quad-core (Snapdragon 800) processor clocked at 2.3 GHz, and 16 GB of internal memory. The second device is a One Plus X with 3 GB of RAM, a quad-core (Snapdragon 801) processor clocked at 2.3 GHz, and 16 GB of internal memory. The 6.0.1 Android version was used on both devices.

During the battery performance evaluation, the devices' displays were kept on to more closely represent a situation where the device is being used. Also, the brightness setting was set to 50%, and the HIDS application was open on foreground. This was done to ensure that both devices are in similar conditions throughout the performance evaluation.

C. *Performance Evaluation metrics*

The performance evaluation aims to determine the impact that the MDC component has on the device's resources and battery consumption. Table I depicts evaluation metrics.

TABLE I: Evaluation metrics

Metric	**Unit**	**Description**
CPU usage	%	Percentage of CPU used by the monitoring component
RAM usage	MB and %	Amount of memory used by the monitoring component
Battery	%	Amount of battery consumption
Sent/Recv packets	MB	Amount of data transferred during communication

To collect the CPU and RAM usage metrics, the *Another-Monitor* [25] tool was used. This tool can monitor the CPU and RAM usage of the HyIDS components and record the measurements into log files and was used to validate the measurements taken by the MDC. The network traffic statistics are collected by the monitoring component.

V. Results

This section presents the results of the performance evaluation, which comprise the execution of 5 runs for each scenario.

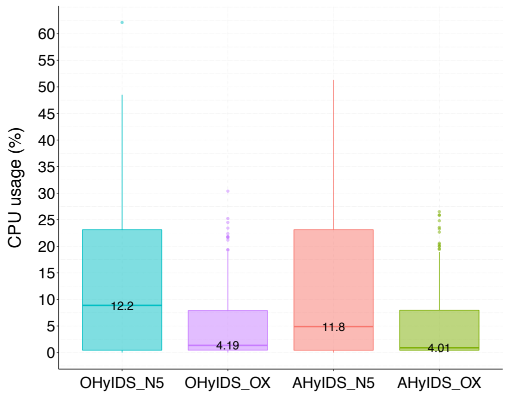

Fig. 2: Average CPU usage

The CPU performance analysis results of the application are depicted in Figure 2. The impact of the MDC is low in terms of CPU consumption. Such impact is even lower in more recent devices, such as the ONE Plus X, since the CPU consumption for the MDC is ≈4% for both of the evaluation scenarios. On the other hand, the CPU consumption in the Nexus 5 device is higher, increasing to values of ≈12%. In comparison with the *AnotherMonitor* program, which performs the same functionality in terms of collecting metrics, the *AnotherMonitor* consumes ≈15% of CPU on the Nexus 5 device, while the MDC's CPU usage is lower, with ≈12% of CPU usage.

The memory analysis results are presented in Figure 3. The impact of the MDC is low in terms of memory consumption, inline with the CPU results. The memory impact represents a

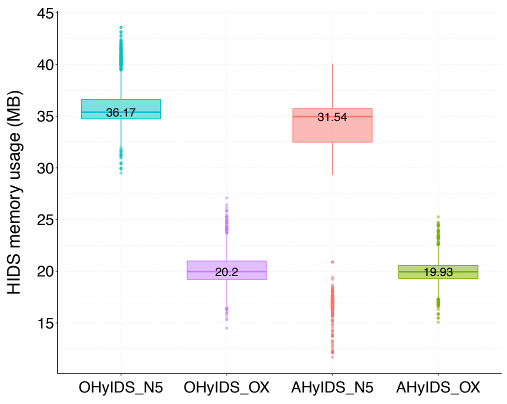

Fig. 3: Average memory usage

≈0,648% memory usage for both scenarios on the One Plus X. On the Nexus 5, the memory usage increases to ≈1,653%, this behaviour is expected since the Nexus 5 has only 2GB of memory opposed to the 3GB of the One Plus X. As depicted in Figure 3, the Nexus 5 device reports more memory usage (in MB). The devices have different memory capacities, despite the same Android base, they have different systems, Nexus 5 has installed Cyanogen Mod, while the One Plus X has the official system from One Plus, the Oxygen OS. Such fact justifies the different memory usage behaviour in the devices.

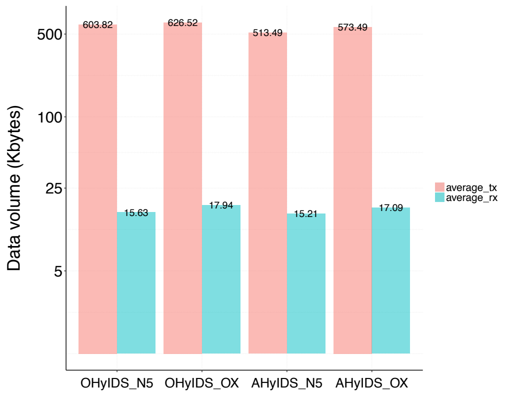

Fig. 4: Average data volume

The transmitted data analysis results are depicted in Figure 4. Similarly to the CPU and memory results, the network data results show that the MDC generates a low volume of data, considering the features that are collected. The results of the received data are similar for both devices with ≈15 and ≈18 Kbytes of data for the Nexus 5 and One Plus X devices, respectively. Such behaviour is expected, since the MDC is receiving acknowledgement messages from the server to confirm the reception of the messages containing information of the collected metrics. Regarding transmitted data, there are some differences. The One Plus X transmits on average more data than the Nexus 5 which is explained by the greater processing capacity of the former which in turn results on more collected samples. On average, the One Plus X collects ≈19% more samples than the Nexus 5.

The battery usage analysis results are depicted in Figure 5. The results show that the battery usage is low on both devices. Also, on the Nexus 5 device the battery usage is higher than

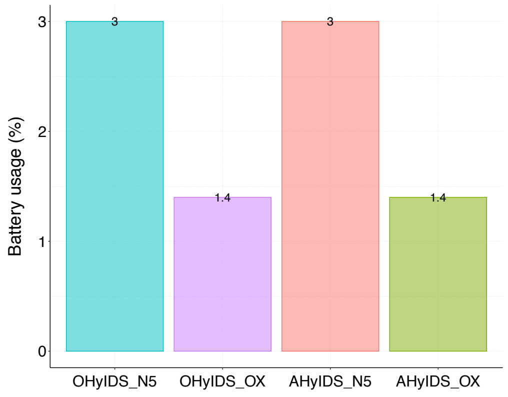

Fig. 5: Average battery usage

on the One Plus X (≈3% versus ≈1,4%). Per the CPU usage results, the MDC uses more CPU on the Nexus 5 device than on the One Plus X; therefore, the device that uses more CPU has a higher battery draw. There is no clear distinction between regarding the the battery consumption when there are no applications running and applications running (e.g., GMail), since they are running in background.

One important result is that the MDC's performance is not hindered by the number of applications that are running in the background on the device. This is supported by the performance evaluation results that show that the CPU and memory usage are similar in both scenarios.

VI. Conclusion & Future work

This paper presents and validates the Hybrid IDS designed to support the BYOD paradigm in PPDR scenarios. HyIDS supports static and dynamic IDS analysis by performing automated permission and behaviour analysis for detecting and preventing malware, and with the aid of an operator at the CCC identify and isolate the risk of user actions that negatively impact the safety of PPDR operations. The validation results demonstrate that the monitoring component of HyIDS has a low impact on the device in terms of CPU and memory usage, and battery consumption. Indeed, such results highlight the design choices of HyIDS, where the collected information is not stored in the RAM of the mobile device, leading to low memory consumption (i.e., not interfering with other applications) and increasing resilience, by persistently storing the collected information. The impact on the CPU usage is low, where measured values are below 15% on a device with a quad core 2.3 GHz processor. The low impact in the network infrastruture is also accomplished, even in collection intervals of 2s, with low volume of data transfer from devices to CCC. Thus, enabling the collection of thousands of devices without upgrading existent PPDR infrastructures.

The mobile correlator design in the mobile device component is being improved. Our plans include optimizations to use selective monitoring (e.g. collect more information, reduce frequency intervals) when suspicious events are detected. Additionally, policy synchronisation is also required between the mobile device and the CCC.

Acknowledgment

This work was carried out with the support of the Centro 2020 Project Mobitrust (CENTRO-01-0247-FEDER-003343).

References

[1] ITU ICT, "The World in 2015 ICT Facts & Figures," Tech. Rep., 2015.
[2] Idc, "IDC_ Smartphone OS Market Share 2015, 2014, 2013, and 2012," 2015. [Online]. Available: http://www.idc.com/prodserv/smartphone-os-market-share.jsp
[3] H. M. et al., *Next Generation Communication Systems for PPDR - The SALUS Perspective*. Wiley-ISTE, 2015.
[4] M. S. L. Alcatel-Lucent, "Malware Report H1 2015," pp. 1–17, 2015.
[5] G. S. et al., "NIST Special Publication 800-30 Revision 1," *Risk Management Guide for Information Security*, 2012.
[6] P. Faruki, A. Bharmal, V. Laxmi, V. Ganmoor, and M. S. Gaur, "Android Security : A Survey of Issues, Malware Penetrarion and Defenses," vol. 17, no. 2, pp. 998–1022, 2015.
[7] P. Faruki, V. Ganmoor, V. Laxmi, M. S. Gaur, and A. Bharmal, "AndroSimilar: Robust Statistical Feature Signature for Android Malware Detection," *Proceedings of the 6th International Conference on Security of Information and Networks*, pp. 152–159, 2013.
[8] D. Arp, M. Spreitzenbarth, H. Malte, H. Gascon, and K. Rieck, "Drebin: Effective and Explainable Detection of Android Malware in Your Pocket," *NDSS*, pp. 23–26, 2014.
[9] Google, "Protect against harmful apps." [Online]. Available: https://support.google.com/accounts/answer/2812853?hl=en
[10] L. Yan and H. Yin, "Droidscope: seamlessly reconstructing the os and dalvik semantic views for dynamic android malware analysis," *Proceedings of the 21st USENIX Security Symposium*, p. 29, 2012.
[11] A. e. a. Shabtai, "Andromaly: a behavioral malware detection framework for android devices," *Journal of Intelligent Information Systems*, vol. 38, no. 1, pp. 161–190, 2012.
[12] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, "Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets," *19th NDSS*, no. 2, pp. 5–8, 2012.
[13] M. Spreitzenbarth, F. C. Freiling, F. Echtler, T. Schreck, and J. Hoffmann, "Mobile-Sandbox: Having a Deeper Look into Android Applications," *ACM SAC*, pp. 1808–1815, 2013.
[14] 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," *Osdi '10*, vol. 49, pp. 1–6, 2010.
[15] P. Lantz, "DroidBox," 2012. [Online]. Available: https://www.honeynet.org/node/940
[16] EsperTech, "Esper Correlation Engine." [Online]. Available: http://www.espertech.com/products/esper.php
[17] D. Kasiaras, T. Zafeiropoulos, N. Clarke, and G. Kambourakis, "Android forensics: Correlation analysis," *9th ICITST 2014*, pp. 157–162, 2014.
[18] V. Hodge and J. Austin, "A survey of outlier detection methodologies," *Artificial Intelligence Review*, vol. 22, no. 2, pp. 85–126, 2004. [Online]. Available: http://dx.doi.org/10.1023/B:AIRE.0000045502.10941.a9
[19] Z. Muda, W. Yassin, M. N. Sulaiman, and N. I. Udzir, "Intrusion detection based on k-means clustering and naive bayes classification," in *2011 7th International Conference on Information Technology in Asia*, July 2011, pp. 1–6.
[20] M. Jianliang, S. Haikun, and B. Ling, "The application on intrusion detection based on k-means cluster algorithm," in *2009 International Forum on Information Technology and Applications*, vol. 1, May 2009, pp. 150–152.
[21] A. Shabtai, L. Tenenboim-Chekina, D. Mimran, L. Rokach, B. Shapira, and Y. Elovici, "Mobile malware detection through analysis of deviations in application network behavior," *Computers & Security*, vol. 43, pp. 1 – 18, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167404814000285
[22] F-Secure, "Threat Report H1 2013," Tech. Rep., 2013.
[23] Symantec, "Android.Hehe," 2014. [Online]. Available: https://www.symantec.com/security_response/writeup.jsp?docid=2014-012211-0020-99
[24] Google, "Testing Support Library (UI Automator) ." [Online]. Available: https://developer.android.com/topic/libraries/testing-support-library/index.html
[25] A. Redondo, "AnotherMonitor," 2016. [Online]. Available: https://github.com/AntonioRedondo/AnotherMonitor