Knowledge-Based Systems 150 (2018) 198–217

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

A novel pattern recognition system for detecting Android malware by analyzing suspicious boot sequences

Jorge Maestre Vidal*, Marco Antonio Sotelo Monge, Luis Javier García Villalba

Group of Analysis, Security and Systems (GASS), Department of Software Engineering and Artificial Intelligence (DISIA), School of Computer Science, Office 431, Universidad Complutense de Madrid (UCM), Calle Profesor José García Santesmases s/n, Ciudad Universitaria, Madrid 28040, Spain

ARTICLE INFO

Article history:
Received 29 November 2017
Revised 6 February 2018
Accepted 10 March 2018
Available online 16 March 2018

Keywords:
Anomalies
Malware
Mobile devices
Intrusion detection
Pattern recognition
Sequence alignment

ABSTRACT

This paper introduces a malware detection system for smartphones based on studying the dynamic behavior of suspicious applications. The main goal is to prevent the installation of the malicious software on the victim systems. The approach focuses on identifying malware addressed against the Android platform. For that purpose, only the system calls performed during the boot process of the recently installed applications are studied. Thereby the amount of information to be considered is reduced, since only activities related with their initialization are taken into account. The proposal defines a pattern recognition system with three processing layers: monitoring, analysis and decision-making. First, in order to extract the sequences of system calls, the potentially compromised applications are executed on a safe and isolated environment. Then the analysis step generates the metrics required for decision-making. This level combines sequence alignment algorithms with bagging, which allow scoring the similarity between the extracted sequences considering their regions of greatest resemblance. At the decision-making stage, the Wilcoxon signed-rank test is implemented, which determines if the new software is labeled as legitimate or malicious. The proposal has been tested in different experiments that include an in-depth study of a particular use case, and the evaluation of its effectiveness when analyzing samples of well-known public datasets. Promising experimental results have been shown, hence demonstrating that the approach is a good complement to the strategies of the bibliography.

© 2018 Elsevier B.V. All rights reserved.

1. Introduction

Over recent years a significant growth in the popularity of mobile devices was observed, which was empowered by their large capacity of connectivity, accessibility, and versatility. Consequently, users increasingly rely on these technologies to perform activities of special sensitivity, such as e-commerce, sharing assets or management of confidential information. This places smartphones directly in the line of fire for cyber criminals, as has been warned by the European Network and Information Security Agency (ENISA) [36]. This document does not simply estimate the increase of these threats; it also alerts of their sophistication, which make them difficult to be detected by the current defense schemes. In addition, it is also important to highlight the risk of the migration of the classical attacks to the mobile platforms, where the adaptation of malware is one of the most common practices. According to the European Police Office (Europol), behind this laborious task, often complex networks of organized crime are hidden [16]. The most widespread propagation strategies are distribution in application stores, social engineering and exploitation of vulnerabilities at communication protocols. On the other hand, there are several studies pointing that the Android operating system is the main target of the attackers. For example, in [8] it is indicated that 99% of the malware for mobile devices is addressed against this platform. Given the lack of effectiveness of the security methods conducted by the different application markets, as well as the overconfidence of many of the users when granting execution privileges (often fueled by lack of knowledge), criminals are able to propagate Android malware rapidly and indiscriminately.

To combat this threat, the research community has developed different proposals. In [55] many of them are deeply analyzed, and the evolution of the malicious applications is studied. From this paper it is possible to observe the main causes that led to the failure of most of the current solutions, where the limitation of computational resources is one of the most problematic. Consequently, the detection systems tend to issue high false positive rates, have difficulties to operate in real time, incorporate vulnerabilities related with privacy, or penalize the quality of service of the pro-

* Corresponding author.
E-mail addresses: jmaestre@ucm.es (J.M. Vidal), masotelo@ucm.es (M.A.S. Monge), javiergv@fdi.ucm.es (L.J.G. Villalba).

https://doi.org/10.1016/j.knosys.2018.03.018
0950-7051/© 2018 Elsevier B.V. All rights reserved.

tected devices. This implies the necessity to propose novel solutions that take into consideration the Android malware evolution and the limitations of this emerging monitoring scenario, which are mostly related with data processing, memory and energy consumption. In order to contribute to mitigate these drawbacks, this paper introduces a pattern recognition system for malware recognition once downloaded from application stores. Its main goal is to prevent the installation of the malicious software on the victim system. To do this, downloaded applications are analyzed in a safe and isolated environment, prior to their deployment on real devices. Unlike the other publications, this approach focuses on the sequences of actions involved in the boot process of the suspicious programs. Thus the amount of information to be monitored is considerably reduced. Another important contribution is to adopt the sequence alignment methods provided by bioinformatics to assess the similarity between applications. The sequences of the boot system calls are handled as strings of amino acids, in such a way that their alignment allows distinguishing regions of greatest similarity, which facilitates decision-making. To determine if the difference between the scores of two groups of test sequences is representative, the Wilcoxon signed-rank test is applied. If a significant difference between the analyzed applications with the set of sequences associated to their legitimate execution is detected, then the program is tagged as malicious. The reference dataset is constantly updated by information voluntarily shared by legitimate users. The effectiveness of the proposal has been evaluated by testing the approach with malicious and legitimate samples from the collection Genome [68], which nowadays are included into the Drebin [4] dataset. The major contributions of this paper are summarized as follows:

- An In-depth review of the bibliography about malware detection on Android systems, where the evolution of malware and the challenges it entails are studied. The latter considered the different proposals for prevention, mitigation and detection introduced by the research community. They were classified, and their principal advantages/drawbacks were disclosed.
- The dynamic detection of malware in mobile devices based on analyzing boot sequences, which reduces the search space at pattern recognition, and prevents the malicious application from being installed on the victim system. Preliminary experimentation demonstrated that this is an effective first line of defense against Android malware, which is capable of accurately recognizing specimens of different nature.
- A novel pattern recognition system for detecting Android malware based on the study of dynamic behaviors of the analyzed applications. For this purpose, several data processing steps are defined, which combine different analytic methods, among them global sequence alignment, statistical tests or Bootstrap Aggregation.
- The adoption of the global alignment paradigm provided by bioinformatics for dynamic-based malware detection on mobile devices. Note that in this context, the sequences which similarity is assessed consist on actions performed by the suspicious applications at their boot process.
- Comprehensive experimentation that includes an in-depth study of a particular use case aiming on aid the understanding of the proposal, and the description of different evaluation tests. The obtained results are discussed in detail and compared with similar publications.

It is important to highlight that the performed research has multidisciplinary nature, which mainly frames three research fields: information security, pattern recognition and bioinformatics. Because of this, an extensive state of art has been reviewed. The most prominent topics required for the best understanding of our contributions were summarized and discussed in the first pages of this paper, which is organized into eight sections. The first three of them provide in-depth studies of the bibliography, which entail the revision of the concepts and design paradigms required for assimilating the rest of the contents. In particular, Section 1 introduces the problem to be solved, summarizes the contributions and describes the organization of the rest of the paper. Section 2 reviews the problem of malware for mobile devices, both from the perspective of information security and aiming on applying pattern recognition and analytic techniques with detection purposes. Given that the proposal adopts strategies from bioinformatics based on sequence alignment, Section 3 describes them and highlights the advantages and disadvantages of each analytical approach. Section 4 formally enunciates the Needleman–Wunsch algorithm and the Wilcoxon signed-rank test. In Section 5 the scope of the problem to be solved is stablished by defining design principles, assumptions and limitations. Based on this, the proposal is described at each of its levels of information processing. On the other hand, Section 6 introduces the evaluation methodology and the datasets considered throughout the experimentation; Section 7 discusses the observed results; and Section 8 details the conclusions and future lines of research.

2. Background

2.1. Malware against Android

The current state of the malware against Android systems is rooted in the evolution of the malicious software in mobile devices. The first specimens, like Cabir (2004) or CommWarrior (2005), exploited vulnerabilities on these technologies in order to propagate through basic communication protocols, such as Multimedia Messaging Service (MMS) or Bluetooth [3]. Although they were relatively harmless, they could generate money losses related to their propagation messages. This led the attackers to pretend obtaining economic benefit of the intrusions, prompting the appearance of specimens such as RedBrowser (2005) or Yxes (2009), which leverage the premium-rate SMS services [11]. The latter would be the precursor of the botnets of mobile devices. The adaptation of well-known threats for personal computers to smartphones reaches an important landmark in Zitmo (2010), the mobile version of the banking botnet Zeus [21]. By then Android already occupied a large market niche, which gave rise to the discovery of new malware specific for this operative system. These are the cases of Gemini (2010), DroidKungFu (2011) or Plankton (2011). They were distributed both from the official Android application store (Google Play) and from unofficial third party stores [7]. In more recent years, the different organizations for cyber defense warned of a significant growth of this kind of malicious software. Furthermore, new ways to profit these threats are emerging, such as those related with adware, riskware, spyware, etc. A good example is observed in the case of the ransomware. It is a class of malware focused on the extortion of the victims. The ransomware is capable of blocking some system functionalities and asking for money (ransom) in exchange for their release. The first discovered ransomware for Android is FakeDetect (2013), a family of malware that operates imitating antivirus software, and demands a ransom for unlock the system assets [28]. Other examples of ransomware are FakeAV (2013), CryptoLocker (2014), Koler (2014) or Locker (2015) [15].

The malware specific for Android, like any other application developed for this platform, is distributed compacted in the Application PacKage file format (APK). The malicious APK files are mainly distributed in official or third party stores, although there are other less frequent ways, such as social engineering (spear fishing, baiting, etc.) or exploitation of vulnerabilities. Then they acquire the capability of propagation via communication protocols (Bluetooth,

Wi-Fi, NTC, etc.) and by different communication channels (email, instant messaging, social networks, etc.). In order to reduce these threats, the official markets require all applications to be digitally signed and certified before they are installed. Therefore, through hash functions the integrity of the packages is checked. They also implement various intrusion detection methods, such as static and dynamic analysis for the recognition of previously known threats or anomalous contents. But even though a priori they seem a good way to prevent distributing malware and legitimate applications that were previously unpacked and poisoned by intruders, in practice they are not so effective. This is due to various reasons, emphasizing among them the emerging of new vulnerabilities and that the certificates do not need to be signed by a certificate authority or could be forged, as is discussed in [44]. On the other hand, in order to evade such intrusion detection systems, attackers are progressively resorting to obfuscation strategies. The problem on these adversarial attacks is widely discussed in publications like [35], where it is shown that by mutations on the section of the Android executable that contains the infection vector, it is possible to deceive several detection engines.

2.2. Related works

In recent years different surveys about the state of the art on security in smartphones have been published. Some of them propose a general overview, as is the case of [31,55]; others deepen on more specific topics, where the Android operating system is one of the most frequent [17]. Finally, there are surveys focused on specific threats, such as [43], which delve into the problem of malware. Throughout the bibliography, the limitation of the computational resources provided by the mobile devices has determined how the monitored information is processed at the defensive process. One way of committing to this task is to perform the analysis on the devices themselves. This poses the advantage of not relying on external deployments of communications with collaborative systems. In addition, the exploitation of vulnerabilities related with privacy is more difficult [13]. But as stated in [7], they also tend to report more false positives, and they are particularly sensitive against adversarial attacks [35]. This is because in order to properly operate, the most accurate methods require more memory, processing power or battery consumption than they have, as discussed in [57]. As an alternative, several approaches delegate the more complex analysis tasks to external services, even assuming the risks that this practice entails.

Another important aspect with significant relevance to the previous proposals is the decision of the features studied by the intrusion detection systems. This problem is not particularly representative for approaches based on recognition of previously known threats [18]. However, as outlined in [22], when the detection is based on identifying anomalous behaviors, it is crucial to ensure the success of the extracted metrics. For this reason, it is understandable that a large amount of taxonomies considers these features as the main distinction traits between the different anomaly-based analysis methods [19]. On this basis, the proposals can be distinguished into four great groups: analysis considering static features, dynamic features, mixed features or metadata.

2.2.1. Static analysis

The analysis of static characteristics inspects the applications at their execution time looking for malicious contents. For this purpose different information is extracted, such as binary code, privileges requested, hardware resources or connectivity. A good example of this approach is observed in [34], where similar features are studied at the exploration of the various application stores searching for particular specimens. Statistic features are considered in [53,54] for identifying malware obfuscation methods. In [65] the source codes of applications at the Android Package Manager Service (PMS) are analyzed. In [23] the privileges requested by applications are analyzed for intrusion detection. Alternatively, in [4] this distinction is made by querying the *AndroidManifest.xml*, where the hardware to be requested is indicated. In [56] the code structure is analyzed for identifying similarities between the different malware samples. Thus, the relationship between specimens of the same strain can be established, in this way allowing to determine their evolution. In [14] a detection method that enforces carefully-chosen benign properties in trustworthy applications, but not in malware, is proposed. Another interesting approach is [51], where a scalable detection engine for APK inspection is designed based on Multifeature Collaborative Decision Fusion (MCDF). In [60,66] contextual dependency graphs are applied for semantic modeling of Android malware. Repacked malicious applications are detected in [32] by payload mining. In general terms, static analysis poses the advantages of simplicity at data extraction stages and efficiency. However, it is a method liable to be deceived by obfuscation schemes. Furthermore, due to its lack of ability when defining the behavior of the applications at runtime, it often does not properly deals with specimens hidden in repacked applications.

2.2.2. Dynamic analysis

The analysis based on studying dynamic features aims on monitoring the protected system behavior looking for malicious activities. As referred in [19], the most common features on the dynamic analysis are the sequences of system calls involved at the execution of suspicious applications. A typical example of this approach is illustrated in Crowdroid [6], a local-based intrusion detection system that considers their frequency of occurrence. In other proposals, such as [33] malicious repackaged applications are identified by analyzing the thread-grained system call sequences. Another interesting instance is MADAM [47], which monitors dynamic features belonging to different data processing levels. This group of contributions often carries out the extraction of the required data in isolated and secured environments, which are commonly referred as sandboxes. In this way it is possible to observe the modus operandi of the attacks without compromising the protected system. In [49] a comprehensive state of the art about sandboxing is presented. But it is worth mentioning that the dynamic analysis also considers other features. For example, in [67] the behavior of the applications is modeled based on the permissions requested. In addition, the effectiveness of studying metrics based on power consumption is discussed in [24]. In [50] an intrusion detection system based on recognizing anomalous behaviors in the activities of applications at communication networks is presented. Several classical pattern recognition methods applied to dynamic malware recognition are compared in [1,2]. Concluding, the analysis of dynamic features tends to be very accurate. However it needs to consume a significant amount of computational resources, a situation that could be un-viable in some devices, hence requiring the availability of additional infrastructure.

2.2.3. Mixed analysis

The more complex monitoring environments are usually more susceptible to become compromised. Because of this, it is common to combine both static and dynamic methods. This collaboration is referred as analysis based on mixed or hybrid data. A good example is illustrated in [63], where the statistical analysis of source code and *AndroidManifest.xml* is performed; in addition, different dynamic features are taken into account, such as registers, system calls or network traffic. Another interesting publication is [45], where the problem of the overload caused by dynamic analysis systems is reduced. With this purpose, a first study of the source code of the analyzed applications is performed looking for suspicious sentences. In [27] an intrusion detection system

that adapts the criminal profiling methodology to malware analysis is proposed. It is based on studying heterogeneous data, among them opcodes, metadata of the *AndroidManifest.xml*, digital certificates or the malicious behavioral patterns at runtime. Thereby, the dynamic analysis is simplified by focusing only on the activities arising from the labeled code, greatly reducing the amount of information to be taken into account. In [26] the analysis of intents and permissions for collaborative malware detection are combined. Summarizing, hybrid proposals equilibrate benefits and drawbacks of the static and dynamic detectors, thus providing versatility in the adaptation to the various use cases.

2.2.4. Metadata analysis

The last group of publications is based on metadata analysis. Metadata is defined in [19] as the information of the applications known prior to their download from the various distribution stores. It provides different features, such as the requirements indicated by authors, opinions, comments, reputation, languages or geolocation. A good example of the study of metadata is illustrated in [41], where information related with the software requirements displayed in the application stores is applied. For this purpose Natural Language Processing (NLP) techniques are implemented to identify sentences that explain the need for a given permission in an application specification. In [59] a greater variety of information is taken into account. Depending on the platform, it includes the description of the application, permissions, ratings, or information about the developers. An study of how to conduct effective communication of risks for mobile devices and their assessment is detailed in [42]. The main advantage of the metadata analysis is that it is capable of malware recognition before downloaded. However, the success of these methods depends on information easily manipulated by the attackers. This facilitates evasion and leads to misclassifications.

3. Sequence alignment

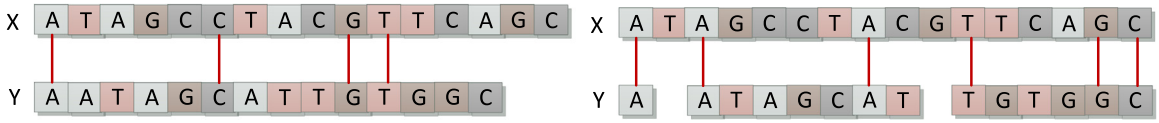
In nature, the crossover of individuals of the same species involves the appearance of different genetic alterations, such as variations on genes (*substitutions*), incorporation of new genes and chromosomes (*insertions*), or their omission (*elimination*). When these mutations pose advantages over other specimens on the same population, the individuals increase their survivability. This is the base of the natural selection. The comparison of the genetic similarities between the genotype of ancestors and offspring is studied in-depth by bioinformatics. With this purpose, this area provides a great collection of strategies, which are able to preprocess, model and measure the various genetic features. The sequence alignment methods are part of this set. They originally determined the degree of similarity between strings of elements (usually DNA, RNA or proteins), which usually correspond to nucleotides or amino acids, and are represented by symbols from a finite alphabet Σ. A wide variety of sequence alignment algorithms has been proposed over the past years. By taking the number of sequences to be compared into account, they are divided into two methodologies: pairwise and multiple alignment. Pairwise sequence alignment methods are used to compare two query sequences. On the other hand, the multiple alignment extends the pairwise methods to analyze more than two sequences at a time, hence entailing a significant increase in computational complexity. Note that these strategies also vary depending on the characteristics of the sample. Consequently, they require observing different aspects of the genotypes, such as their genes, structural features or their phylogenetic trees. Given that in our approach, only the pairwise alignment is applied, hereinafter only this method will be referred. In particular, the efforts carried out adopt the generalization to the problem of find the length of the Longest Common Subsequence (LCS) between two segments, according to the classical solution published in [62]. Therein different operations on the original sequences are performed, such as substitutions, insertions and elimination. They are achieved through the incorporation of new symbols into the chains, which are defined as *gaps*. The similarity of each pair of sequences in these configurations is measured by calculating the best distance between all their possible subsequences.

A practical example of this process is shown in Fig. 1(a) and (b). In the first of them, two sequences are compared without alignment. Let the alphabet $\sum = \{A, C, T, G\}$ where each symbol represents an amino acid: Adenine (A), Cytosine (C), Thymine (T) and Guanine (G). The sequence X is $\{ATAGCCTACGTTCAGC\}$ and the sequence Y is $\{AATAGCATTGTGGC\}$. There are four matches $\{A\}$, $\{C\}$ and $\{GT\}$. In Fig. 1(b) the sequences are aligned by the insertion of a pair of *gaps*. This leads to recognize a greater amount of matches, in particular six: $\{A\}$, $\{A\}$, $\{A\}$, $\{T\}$ and $\{GC\}$. Assuming a simple heuristic that adds $+1$ to the score when a new match is found, and that it does not include penalties, the score at the first example is 4 and the score after alignment is 6.

The definition of appropriate scoring heuristics directly affects the results, which typically consider the impact of matches, non-matches and *gaps*. In bioinformatics they are also built considering the class of amino acids involved in matches and non-matches, which often required specialized knowledge-bases, as discussed in [30]. In addition, the location of *gaps* and the scoring heuristic usually depends directly on the alignment methods. These strategies have traditionally been based on the global, local or hybrid reallocation of their elements, as described below.

- *Global alignment*. This family of algorithms performs alignment considering the complete sequences to be compared. The most important method is well-known as Needleman–Wunsch and it was proposed in [37]. It usually yields better results when the sequences have almost equal length, as it provides an overview of their global features
- *Local alignment*. The algorithms based on local alignment aims on finding the most similar regions within the sequences to be compared. They are commonly implemented as variations of the Smith–Waterman method proposed in [52]. It is based on combining all the partial global alignment scores of the various subsequences within the original data. Because of these characteristics, the local alignment provides high effectiveness when analyzing chains with different lengths.
- *Semi-global alignment*. The hybridization between global and local alignment often is referred as semi-global alignment. It is usually performed by implementing slightly modifications on the Smith-Waterman method; highlight among them not to apply penalizations at the beginning and the end of sequences. They aim on comparing the similarity between complete sequences and subsequences. Because of this, they are usually recommended for the comparison of chains with very different lengths [40].

The sequence alignment algorithms were original implemented by dynamic programming schemes. But to manage large amounts of information, as is the case of the billions of nucleotides within sequences in nature, leads to their simplification by heuristic approaches, which reduces computational cost, but decreases accuracy. Most of these heuristic approaches are studied in [40]. Since the analysis and comparison of sequences is a common problem at different research fields, the sequence alignments has been frequently used at very diverse use cases. For example, in [25] they are invoked to extract out the representative patterns which denote specific daily activities of a person from reference samples. A very different use case is illustrated in [46], where sequence alignment is applied by web crawlers to detect and remove du-



(a) Basic. (b) *Gaps*.

Fig. 1. Example of sequence alignment.

plicate documents without fetching their contents. Despite their popularity, they are unusually considered in order to face the challenges on information security. In this case, they are focused on recognizing actions performed by users of the protected systems. A good example of their contribution is described in [10], where they are applied for detecting masquerade attacks. In particular, the approach aligns sequences of system calls looking for impersonations. In addition, the effectiveness on the implementation of the different alignment algorithms is discussed.

4. Methods

This section formally describes two methods that play an essential role in the rest of the paper: the global alignment approach introduced by Needleman–Wunsch [37] and the Wilcoxon signed-rank test [64]. The first is considered for assessing similarity between legitimate and malicious system call sequences, and the second facilitates to determine when the differences between a sample and the legitimate behavior of an application are significant.

4.1. Needleman–Wunsch algorithm

The global alignment approach published by Needleman–Wunsch [37] was introduced in Section 3. This method originally facilitated the comparison of two sequences of symbols by a dynamic programming approach, where a larger problem is divided into smaller subproblems as follows: let the pair of sequences to be aligned A and B, such as $|A| = m$, $|B| = n$, the matrix of their similarity F has dimension $m \times n$. Within F matrix, the value of each cell $F(i, j)$, $0 < i \leq m$ and $0 < j \leq n$, is the score of the best alignment between sub-segments of the first i elements of A, and the first j elements of B. Therefore F stores the best alignment between the two chains, and $F(m, n)$ contains the optimal alignment. The matrix is recursively constructed from the following base cases:

$$F(0, j) = d \times j \tag{1}$$

$$F(i, 0) = d \times i \tag{2}$$

where d is the *gap* penalty. From this data it is possible to fill the rest of the matrix. The value of each cell $F(i, j)$ is determined by its row, column, or diagonal, defined as the following expressions:

$$F(i, j) = max\{F(i-1, j) + d, F(i, j-1) + d, F(i-1, j-1) + S(A_i, B_j)\} \tag{3}$$

where $S(A_i, B_j)$ indicates the similarity between the elements A_i and B_i. This value is often defined as a scoring matrix, which is the heuristic implemented by the algorithm. The computational cost of this process is $\Theta(n^2)$. This is one of the main reasons that led to use heuristic approaches when analyzing large sequences. Once F is completely filled, the ordering of the sequences and insertion of *gaps* start in the position $F(m, n)$. The matrix is traversed selecting as next position the best value of $F(i-1, j)$, $F(i, j-1)$ and $F(i-1, j-1)$. Thus the optimal solution at each position (i, j) is considered. If the chosen value is $F(i-1, j)$ or $(i, j-1)$, then A_i is aligned with a *gap*. But in the case of $F(i-1, j-1)$, it is aligned with B_i.

4.2. Wilcoxon signed-rank test

The Wilcoxon signed-rank test is a non-parametric test that operates on vectors with paired elements and compares those with the same index [64]. To this end it is assumed that the vectors contain n pairs of observations, each of them referred as (x_i, y_i), $0 < i, \leq n$. Other assumptions are that data comes from the same population, each pair is chosen randomly and independently, and data is measured at least on an ordinal scale (cannot be nominal). The objective of the test is to determine that the values x_i and y_i are equivalent. Hence, the null hypothesis is that the median difference between pairs of scores is zero. The test statistic is W, which is defined as the smaller of W_+ (sum of the positive ranks) and W_- (sum of the negative ranks). Bearing this in mind, when the null hypothesis is satisfied, it is expected to find similar numbers of lower and higher ranks that are both positive and negative, so W_+ is close to W_+. In order to verify this assumption, each paired difference $d_i = x_i - y_i$ is calculated. Every d_i is ranked ignoring the sign (i.e. assign rank 1 to the smallest $|d_i|$, rank 2 to the next, etc.). Then they are tagged according to their sign. The sum of the ranks with positive differences W_+ and negative differences W_- are calculated. With this, the statistic W is calculated according to the following expression:

$$W = min\{W_+, W_-\} \tag{4}$$

From the statistical W it is possible to calculate a p-value (p) that indicates the estimated probability of rejecting the null hypothesis. In case of non-representative n (usually $n < 20$), p is directly calculated from the table that describes the W distribution of critical values as stated in [64]. If n is large enough, p is computed from $P(Z < z)$ on the normal distribution table, were z is expressed as follows:

$$z = \frac{\frac{W - n(n+1)}{4}}{\sqrt{\frac{n(n+1)(2n+1)}{2}}} \tag{5}$$

The test is passed if $p < I$, where I is the confidence interval defined for the evaluation. This is interpreted as the difference between the populations is significant, and therefore it is not due to randomness.

5. Malware recognition in Android

In this section the main features of the proposed detection system are described. Because of the many challenges inherent in malware recognition on mobile devices, prior to its development it is important to stress those aspects that have led to define the design principles of the approach. In order to delimit the situations to be taken into account, the following enumerates the various assumptions that have been considered.

- Android is the most widespread operating system for mobile devices, hence most of the malware for smartphones is directed against this platform. The proposed intrusion detection system assumes that the detection of a larger portion of malware on this environment is possible. Thus, it is aimed on identifying and preventing the execution of malicious software on such devices.

- Given that an important part of Android malware is distributed from application stores, this proposal assumes that its detection once downloaded and before being installed on the victim systems is possible.
- As mentioned in Section 2, the execution of applications in an isolated and secure environment before they perform changes on the real system is possible. This proposal assumes that these methods are effective. Furthermore, it lies on them for extracting the information to be analyzed.
- The accuracy of the dynamic analysis over other approaches is emphasized in the bibliography. As an important part of those publications, this proposal assumes that the dynamic analysis of the sequences of system calls invoked by the monitored applications, is an effective measure for distinguishing malicious contents.
- Unlike many of the previous proposals, this approach assumes that it is possible to recognize malware by studying sequences of actions monitored at the boot process of the applications, which implies taking into account their temporal relationships. This assumption is the principal null hypothesis of our research.
- The proposal also assumes that even when launched on the same device, it is difficult to find two equal boot action sequences of the same application. However, sequences from the same application pose significant resemblance, situation that allows distinguishing between legitimate applications and modifications that contain infection vectors.

Bearing these assumptions in mind, the proposed malware detection system performs dynamic analysis of the behavior of the monitored applications once downloaded from application stores. To this end, the system calls monitored at their boot process are extracted and analyzed when executed in a secure and isolated sandbox. Thus the amount of information to be considered is reduced, since only the system calls related with their initialization are studied. Unlike similar approaches referred to in the bibliography, the temporal relationships of these actions and the order they are initiated are considered. In addition, by the implementation of a sequence alignment algorithm is possible to identify the subsequences with greater similarity and determine their resemblance to the behavior of the legitimate applications. Upon completion of the analysis, only applications labeled as legitimate are allowed to make persistent changes on the mobile device.

The proper functioning of the proposal is defined by two main requirements. The first of them is to be able to accurately identify most of the analyzed threats. This also includes to report a low amount of errors when inspecting legitimate applications, thus reducing the penalization on the quality of experience of users. On the other hand, the detection system must adapt to the protected environment characteristics. This involves operating with good performance, and low consumption of computational resources. Precisely because of the latter requirement, and taking into consideration the limitations of mobile devices, an important part of the analysis tasks are carried out by external infrastructure, as it is described in the following subsections. In general terms, these requirements meet an important part of the needs of the intrusion detection on smartphones. However there are other aspects that have been set aside, highlighting among them the fight against the various evasion strategies [35,61], the interoperability with other security tools or the adaptation to the various data protection policies. They are obviously important to grant its deployment on heterogeneous use cases. But they also add much more complexity to the problem to be solved, not being discussed for the better understanding of this first approach, but being a good aim for future work. The following describes the system architecture, monitorization, analysis and decision-making.

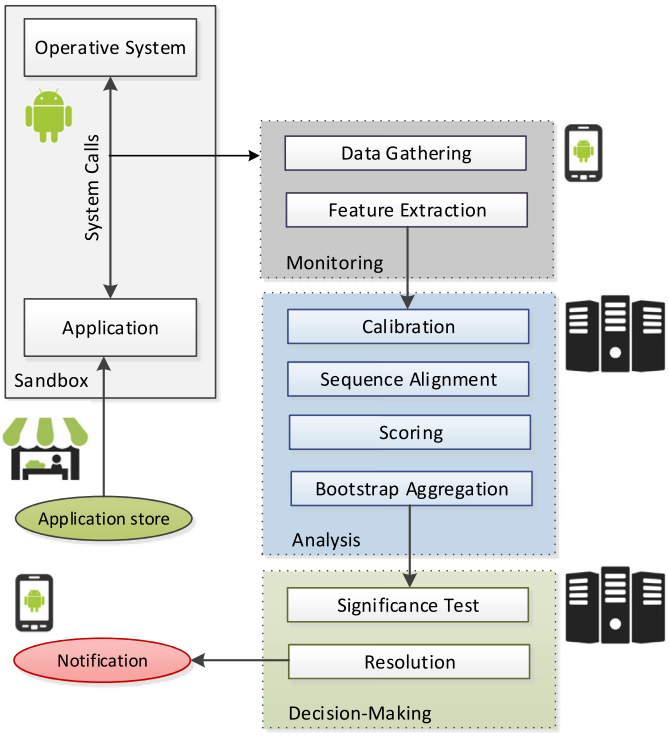


Fig. 2. Architecture of the proposal.

5.1. Architecture

The architecture of our proposal is distributed. This decision implies the addition of complexity to the design, but allows the distribution of the computational resources at different data processing stages. Thus the tasks involving higher costs are performed on dedicated servers, while on mobile devices only behavioral patterns of applications downloaded by users are captured. Furthermore the protected devices must provide access to communication networks; a minor inconvenience considering the enhancement of performance that analyzing data at dedicated servers provide, and that applications are usually downloaded from application markets. As an alternative to the distributed scheme, the downloaded applications may provide information related to the characteristics of their execution as metadata. It should vary according to the features of the device, such as the Android version or software device. In this way the smartphone is able complete the analysis by itself, assuming that it meets all the memory, battery and processing requirements. However the centralized approach has not been implemented in this work, being postponed to future research.

The implemented architecture is summarized in Fig. 2. Three processing layers are defined: monitoring, analysis and decision-making. The monitoring stage is the only one that takes place at the protected device. Detection and decision-making occur in a dedicated server. The proposal performs as follows: the suspicious software reaches the protected device via application stores, and it is installed and executed in an isolated and safe sandbox; therefore, it is not able to make changes to the device, but malware activity can be drawn. At the first boot of the suspicious applications, the sequences of system calls are captured and transmitted to the analysis stage. This information is preprocessed and aligned with sequences of actions observed at their legitimate executions, which were previously shared by trusted users of similar devices. The resulting scores allow decision-making level to determine the nature of the applications. Thus, if an application is labeled as a potential threat, changes in the protected system are not made.

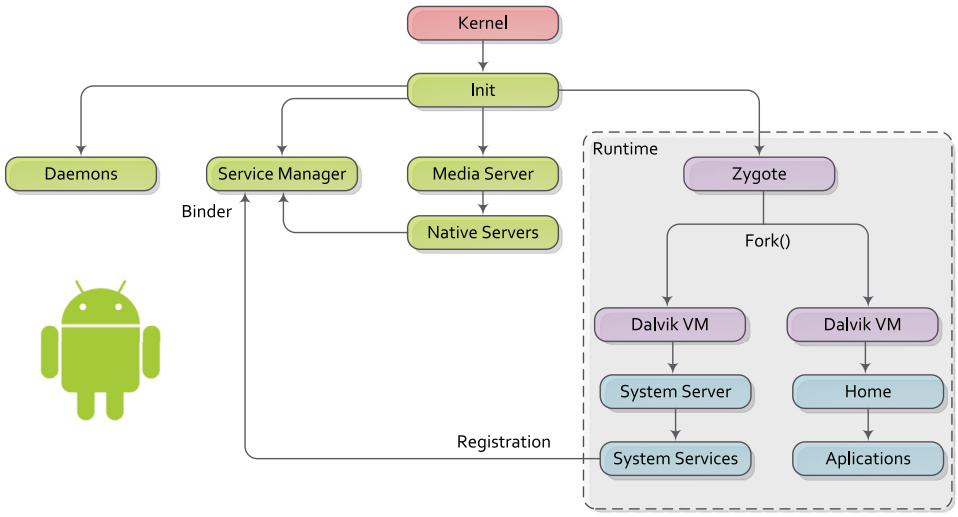


Fig. 3. Android boot sequence.

5.2. Monitoring and feature extraction

At the monitoring stage, the mobile device captures in a sandbox the system calls launched by the suspicious applications at their boot processes. As in the previous works, the monitoring of the executed applications is performed by the diagnostic tool *strace* [6], which is present in most of the GNU/Linux systems, such as Android. This utility facilitates registering the system calls carried out by a program or process by audition of the system call interface, which allows communication between the kernel and the upper layers of the operating system. In order to study all the activities of a particular program from initialization (including those processes derived from it), the parent process (a.k.a Zygote) must be monitored. Zygote is a daemon triggered by the Android *init* process responsible for triggering all the application processes. In Fig. 3 the Android boot sequence and the role of Zygote are summarized. When an application launches Zygote, it creates the first Dalvik virtual machine and calls its main method, which preloads all necessary Java classes and resources, starts System Server and opens a socket to listen for requests of the starting applications, as described in [58]. The monitoring stage implements *strace* in order to capture the sequences of system cells derived from zygote, after the suspicious application is launched. Note that aiming on reducing noise, the sandbox only executes the application to be analyzed.

On the other hand, and with the purpose of bypass the inherent characteristics of the device, the consecutive repetitions of the same actions (i.e. system calls) are simplified in one of them. This further reduces the problems related with errors at the capture tasks, which are usually triggered by redundant data appended by the monitoring software. It should be taken into account that the length of the boot sequence may vary, often observing around 2000 system calls in small applications, and more than 50,000 in the most sophisticated. To simplify treatment, they are preprocessed, so every type of action is associated with a symbol. Hence, if the operating system offers a repertoire of l system calls, the alphabet Σ that identifies every possible action has length l. Ideally, the complete sequences are studied. However, it is possible that because of computing limitations, only the first boot steps are aligned.

All captured sequences are transferred to a dedicated server, where they are compared with legitimate executions shared by other users. If the application is labeled as legitimate, the new sequences are candidates to be included into the knowledge-base, so if it is verified that the applications are clean, their sequences may be used as training samples to evaluate future executions. This prevents the attacker compromising the detection system by poisoning the set of reference samples.

5.3. Analysis

The main purpose of the analysis module is to generate the metrics that allow deciding the nature of the suspicious software. The basic components of these metrics are the scores obtained by aligning the sequence of system calls related with the boot process of the suspicious application, with sequences corresponding with the launch of the legitimate application on similar mobile devices. The implemented sequence alignment algorithm is an adaptation of the global alignment method proposed by Needleman–Wunsch [37]. It is a dynamic programming scheme which correlate both sequences from a matrix F. The decision of implementing Needleman–Wunsch has been addressed keeping in mind that because of computational issues, the maximum length of the monitored sequences is defined. The captured activities often reach such amount, so the alignment algorithm usually operates at the case where both sequences exhibit the same dimension, situation that encourages the use of the global alignment paradigm. This choice is also supported by the fact that the sequences to be compared contain patterns of similar priority. However, if future use cases require to study large sequences, the adoption of heuristic approximations [40] or optimizations based on exploiting parallelism/concurrency [38] are suggested. But given their complexity, and aiming on facilitating the understanding of the performed research, the study of their adaptation and effectiveness when operating on similar problems is out of the scope of this paper. In general terms, the analytic task distinguishes four main data processing steps: calibration, sequence alignment, scoring and bootstrap aggregation. The following subsections describe them in-depth and provide some illustrative examples.

5.3.1. Calibration

Deciding the best sequence length is not a trivial problem, which often involves a trade-off between accuracy and computational cost. This must directly be tackled by the security operators, whom in addition to bear in mind the behavior of the sensor, should consider the characteristics of the use cases in which the proposal is deployed, as well as the adopted risk management policies. But as it was demonstrated in the preliminary experimen-

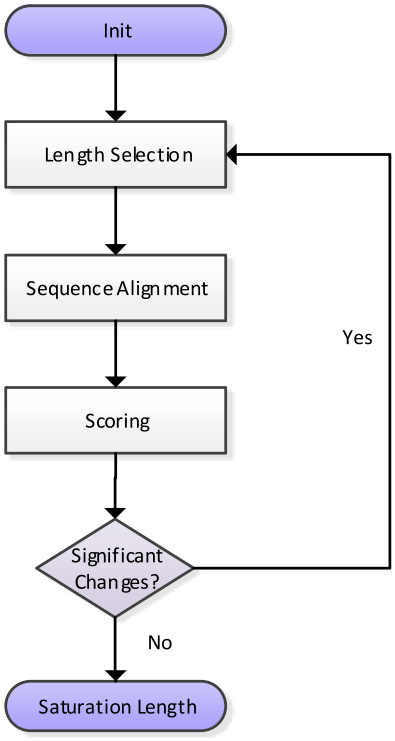


Fig. 4. Sequence length decision-making.

tation (see Section 6), there is a sequence length per reference dataset that when growing, does not provide a significant accuracy enhancement, i.e. the resultant precision gain rate is lower than certain threshold (by default 0.15). Consequently, at an initial training stage it is possible to find the average saturation length of the set of reference samples, as well as the accuracy improvement obtained as it grows (see Fig. 4). With this purpose and starting from a minimum length (in the experiments, 500 actions), cross validation can be performed on different configurations. This process shall be stopped when by increasing the sequence size no significant results in terms of accuracy are observed. So, in addition of calculating the saturation length, the training step provides the list of the different sizes evaluated and the accuracy enhancement they entail, in this way facilitating the decision to be made. As a final remark, it is worth to mention that at experimentation, the saturation length did not entail computation restrictions, so it was always adopted.

5.3.2. Sequence alignment and scoring

The implemented sequence alignment method is driven by the global alignment approach proposed by Needleman–Wunsch [37], which was formaly described in Section 4. It is based on building a F matrix with dimension $n \times m$, where the value of each cell is the score of the best alignment between sub-segments. Note that given that the maximum length of the boot sequences is l, F has dimension $l \times l$ in the worst case. Hence if l is too large, lot of memory for storage purposes is required. This potentially may become a problem if the proposal is not implemented by a distributed architecture. Even in this case, it could be an issue, because the detection system needs to transmit a large amount of information to the server-side, in this way consuming more network bandwidth contracted by the user. But the greater the length of the sequences, the more information they provide, hence potentially allowing more accurate analytics. Consequently, it is recommended to select lengths as large as possible (optimally, saturation

		D	E	B	F	B	C	F	C	E	E
	0	-8	-16	-24	-32	-40	-48	-56	-64	-72	-80
A	-8	-2	-9	-17	-25	-33	-41	-49	-57	-65	-73
B	-16	-10	-3	-4	-12	-20	-28	-36	-44	-52	-60
C	-24	-18	-11	-6	-7	-15	-5	-13	-21	-29	-37
D	-32	-14	-18	-13	-8	-9	-13	-7	-3	-11	-19
E	-40	-22	-8	-16	-16	-9	-12	-15	-7	3	-5
B	-48	-30	-16	-3	-11	-11	-12	-12	-15	-5	2
E	-56	-38	-24	-11	-6	-12	-14	-15	-12	-9	1

Fig. 5. Example of alignment with Needleman–Wunsch.

lengths), according to the computational limitations of the deployment environment.

The alignment aims to match representative sets of system calls in a tested block with similar groups in the sequence derived from the suspicious application. It is a very similar situation to that discussed in [9], so an important part of the guidelines that have led to its resolution are taken into account. For example, *gaps* inserted in the sequence to be analyzed are differently penalized than those on the reference executions. In particular, *gaps* in the latter are slightly more penalized, as it is considered less desirable to alter the order of the execution of system calls in the reference samples (whose legitimacy is known a priori) than the order of the sequence to be tested. On the other hands, matches increase the score corresponding to the similarity between both sequences. As is usual in the bibliography, in our proposal mismatches do not influence the score. In related works this only occurs on certain contexts, and assuming particular knowledge-bases [30], which is out of scope. With this in mind, in the analysis stage the following scoring function is considered:

$$S(A_i, B_j) = \begin{cases} 1 & \text{if } A_i = B_j \\ 0 & \text{if } A_i \neq B_j \end{cases} \tag{6}$$

where every single match scores $+1$, mismatches do not penalize, and in addition to $S(A_i, B_j)$, *gaps* are penalized non uniformly by assigning the value $d = -2$ if they are in the sequence to be evaluated, and $d = -3$ in the case of taking part of the reference samples. Finally, it is important to characterize the set of reference sequences. The sequence to be tested gets a score from its alignment with each of the sequences within this collection, and at the end of the analysis stage, a vector s of length n that summarizes all the calculated scores is built. Therefore, the value of n depends on the size of the set of legitimate executions considered as reference. The configuration of this set is relevant, and corresponds to the problem of validating models common in the various machine learning algorithms. The set can be seen as the model of legitimate execution of the application. Because of this, it is easy to realize that the more representative is the dataset, the more accurate analysis. Furthermore, the smaller is the set, the faster the algorithm.

In Fig. 5 an example of F matrix built with the Needleman–Wunsch method is shown. There two sequences are aligned: $\{ABCDEBE\}$ and $\{DEBFBCFDEE\}$. The alphabet is defined as $\sum = \{A, B, C, D, E, F\}$ where each symbol represents a different system call: *create_module* (A), *close* (B), *writev* (C), *waitid* (D), *execv* (E), *fallocate* (F) and *getsockname*(G). At the end of the algorithm the final score is 1, as it is indicated at $F(m, n)$. The best alignment is calculated traversing the matrix following the best score of $F(i-1, j)$, $F(i, j-1)$ and $F(i-1, j-1)$, hence assuming that the current po-

sition is $F(i, j)$. By this way the sequence $\{--A-BC-DEBE\}$ is the most similar to $\{DEBFBCFDEE\}$. From this figure, it can also be clearly seen how the matrix was built. For example, corresponding with the recursive expressions of the method, the position $F(1, 2)$ contains the best score of $F(0,2)+d$, $F(1,1)+d$ and $F(0,1)+S(B_i, D_j)$. Given that *gaps* are penalized ($d=-8$), the mismatch between B and D is also punctuated ($S(B_i, D_j)=-2$) and the related values on the matrix are $F(0,1)=-8$, $F(1,1)=-2$ and $F(0,2)=-16$, then $max\{-10,-10,-24\}=-10$, concluding with $F(1,2)=-10$, as it is indicated in the appropriate cell. A more detailed example based on real data is illustrated in the tables and figures attached in the Annex. They display the appearance of a real log of system calls considered during the experimentation (see Fig. A.11), the first 120 actions registered when executing the legitimate application *Monkey Jump 2* twice (Tables A.2 and A.3), and the results of executing *Monkey Jump 2* when triggering the *GinMaster* malware infection vector (Table A.4). The alphabet Σ considered for their alignment is summarized in Table A.5, and the results of their alignment are detailed in Table A.6. Assuming a scoring system similar to Eq. (6), it is observed that by global aligning the couple of legitimate samples a greater score is obtained (84/136, normalized in 0.62) than that for comparing one of them with the harmful execution (68/149, normalized in 0.46).

5.3.3. Bootstrap aggregation

In the implementation, the amount of executions taken as reference is predefined. It is assumed that the number of available legitimate sequences is greater or equal to the set applied in the analysis. Because of the construction of different reference sets, the classifications could vary, situation that derives on irregular behaviors of the system. Fortunately, this is a problem widely discussed by the machine learning community [29]. Among the various existing solutions, Bootstrapping Aggregation was implemented [5]. The main goal of this method is to build m training datasets C_i bagging from a larger training set C, uniformly and with replacement. In this way the system generates different datasets from the original group of sequences of legitimate executions, and then it uses each of them on a different instance of the sequence alignment process. In particular, all the possible combinations of n legitimate executions are considered. From these analytics, m scoring vectors S_i, $0<i\leq m$ are obtained. Then the results are pooled, and from them the final classification provided by the detector is inferred after decision-making. The aggregation of scores is performed by arranging the elements of each S_i from lowest to highest. The value of every position j on the final vector is the average of the values in j of each S_i. Because of this, at the end each element on the final scoring vector contains the average score of the partial results in the same position.

In Fig. 6 the management of the dataset of boot sequences gathered when analyzing suspicious applications is summarized. When a new request of analysis reaches the intrusion detection system at server-side, the subsets considered from the instances of the detector are bagged. Consequently, the different analytics are carried out in concurrency. Each of them provides a particular scoring vector, thus representing the similarity of the tested sequence with all the samples within the reference sets. Finally, the elements on the scoring vectors are sorted. This information is pooled in a final vector by calculating the average score of each position. In this way the metric for the decision-making stage is generated.

Note that the main drawback of bagging reference sets is that computationally, involves a very expensive strategy, so it is not suited for centralized architectures. Moreover, its scalability is limited, so if the database grows too much, the method would lack efficiency. A common way to reduce this problem is by randomly selecting the group of samples to be divided into the training sets. But in order to avoid randomness, the number of samples for each

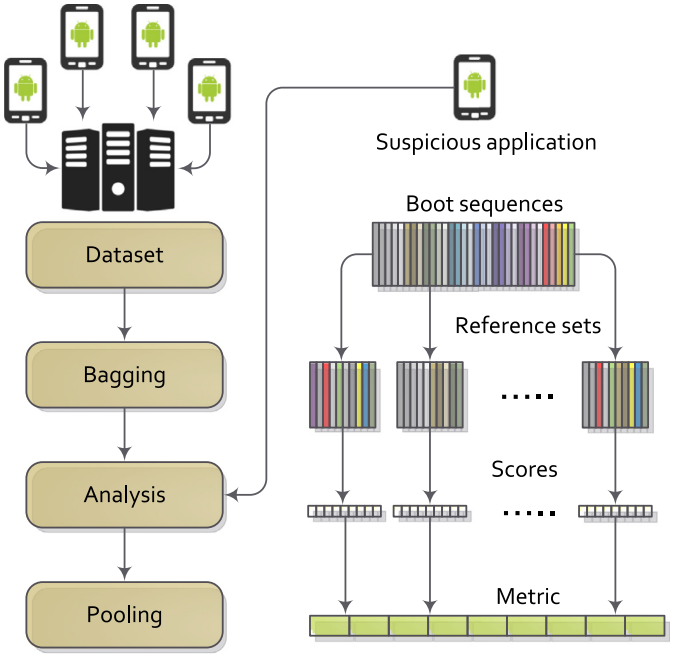


Fig. 6. Bootstrap aggregation when analyzing applications.

application in the database is fixed. This could affect the process of upgrading C by adding the latest sequences tagged as legitimate and verified. To reduce this problem, the database is updated each time a new sequence is added, but replacing some of the older entries. In this way it also preserves consistency. To keep update the data in the knowledge-base, the information reported by mobile devices that request analytic tasks is taken into account. Therefore, they provide the new reference samples for future detection processes, which alternatively may be modified by experts or security operators. This reduces the *concept drift* problem inherent in non-stationary monitoring environments [12], and adapts the defensive deployment for being able to recognize the mutations and evolution of the various malware families [3]. But this solution also raises challenges related to privacy, data protection and poisoning the knowledge-base for hindering malicious pattern recognition, that in order to facilitate the understanding of the performed research are not discussed in-depth.

5.4. Decision-making and classification

At decision-making stage, the suspicious applications are labeled as legitimate or malicious considering the information provided by the analysis module. In the first case, they are allowed to act on the protected environment, and their boot sequences may be added to the dataset. But if they contain malware, the intrusion detection system reports the incidences, and the potential threats are blocked. In particular, the labeling step has as input, the vector of scores obtained after aligning the sequence of system calls gathered by monitoring the Zygote process at the boot of the suspicious applications, with the bootstrapped sets of references. Apart from this data, it also considers the vector of scores obtained after aligning all the legitimate sequences with each of them. With all this information the Wilcoxon signed-rank test is performed [64], which was formally described in Section 4. The adoption of this statistical test is justified by two premises. First, it is not possible to assume the distribution of the scoring vectors, which leads to apply a non-parametric test. This reduces the search space, but there are still many statistical evaluation schemes framed within this family. Nevertheless, and unlike the other non-parametric tests

(such as the Mann–Whitney *U*-test), samples are not independent, leading us to treat data pairs (analogue of dependent *t*-test for paired samples), where the Wilcoxon signed-rank test is the best solution. Note that the test aims on determining if the values x_i and y_i are equivalent. In the proposed system, these coincide with the elements *j, i* of the scoring vectors to be compared.

The test is passed if $p < I$, where I is the confidence interval defined for assessment purpose, and p is the resultant p-value that indicates the estimated probability of rejecting the null hypothesis. Formally, this is interpreted as the difference between the populations is significant, and therefore it is not due to randomness. Consequently, at the proposed strategy is considered that the sequence of system calls to be analyzed does not resemble the executions of legitimate applications within the reference datasets, since the monitored observations were summarized at x_i and the legitimate booting sequences obtained those at y_i. In the first case the booting process is classified as *anomalous*, so a potential malicious application is detected. On the contrary, it is classified as *legitimate*.

6. Experimentation

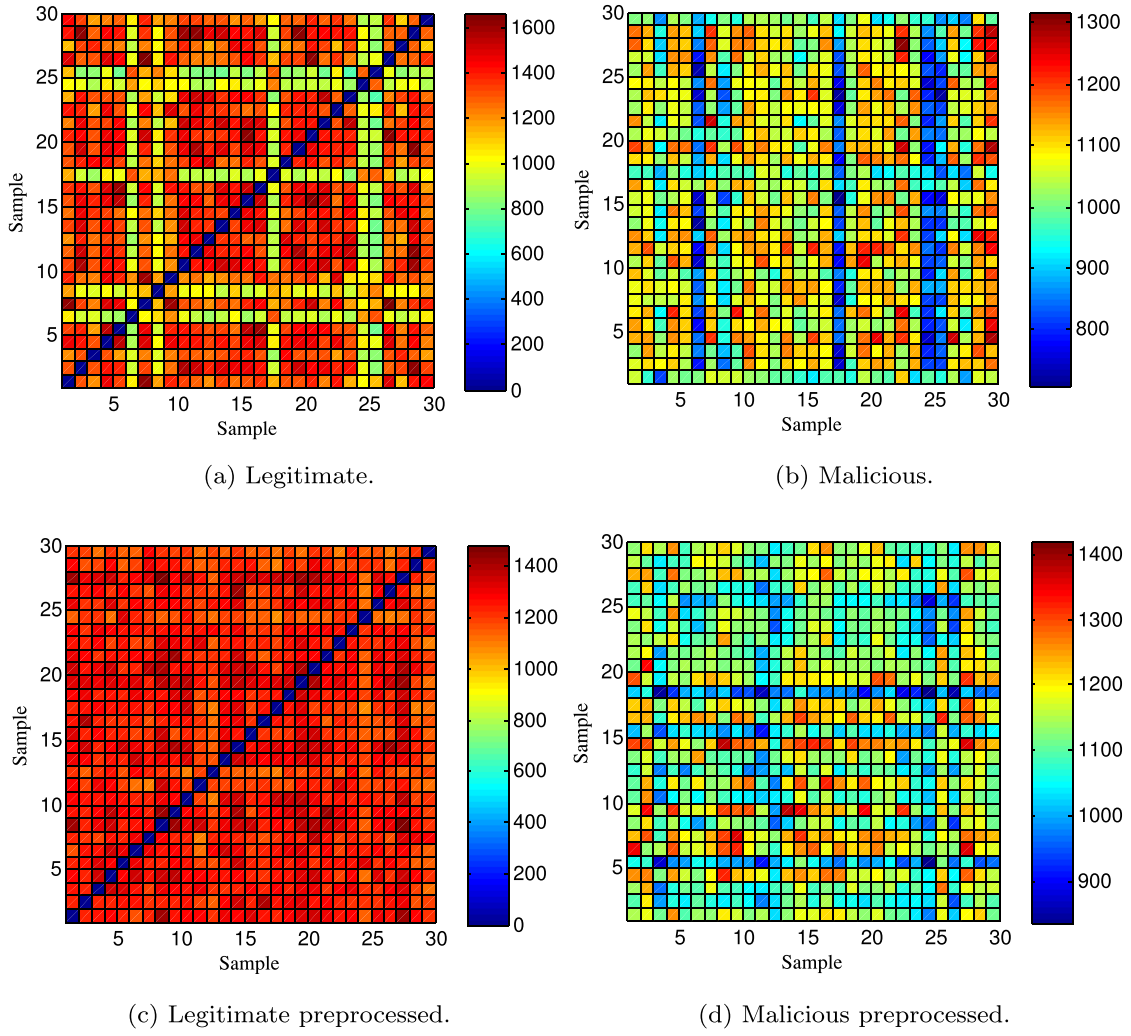
In the experimentation, the proposed system has been deployed on different mobile devices with various versions of the Android operating system. As no significant differences have been found in terms of effectiveness, the device setting is not considered in the group of sensitivity parameters of the tests. The experimentation has been conducted through a cloud computing server built on Openstack [39] and an instantiated virtual web service running Ubuntu server 16.04-x64 with 8 CPU cores and 8GB of RAM. The communication with the mobile devices was carried out through REST (Representational State Transfer) web services written in Flask [20]. Note that the latter has been selected from among other solutions for simplicity and efficiency, with REST being the predominant API model built on HTTP methods [48], which also facilitates the accurate measurement of the CPU processing times per request. At the performed test, only the initial subsequences of the boot processes have been considered, which length has varied between 500–2500 actions. In the worst case, their computation delayed few seconds (usually milliseconds), being this the reason why the impact of this feature has not been studied in-depth. The analyzed collection of applications contains malicious specimens of the public dataset Genome [68], nowadays included into the Drebin [4] dataset. The following 19 legitimate applications from Drebin were studied: *Diner Dash 2, Jaro, Mash, Plumber, Fruits Matching, Scrambled Net, Solitaire, Tap and Furious, Robotic Space Rock, Basketball shot, Monkey Jump 2, Whites out, Super touch down, Tilt Mazes, Helix, DailyMoney, Sanity, Best Voice Changer* and *Z-test*. Their malicious versions contain specimens of the following 9 malware families: *DroidKungFu, Plankton, Geinimi, GinMaster, Cogos, jSMSHider, VdLoader, Gapev* and *Gamex*. The monitoring was distributed at different intervals throughout a time period of 2 months. A total of 300 different boot sequences per application were considered (150 legitimate and 150 malicious, 3 per device) assuming variations in mobile devices and operative system. So the complete dataset contains 2850 executions of the legitimate applications and 2850 sequences of attacks. It is worth mentioning that all the data gathering tasks were performed at real mobile devices owned by different users. The evaluation methodology involved to study the behavior of the proposed strategy when analyzing both, malicious and legitimate applications. In the first case, the intrusion detection system adopted the legitimate collection of samples as reference set, and the malicious collection as test set. Then the malware was dissected one by one, and the analytic results were observed. On the other hand, in order to measure the sensitivity of the approach to report false positives, the legitimate samples were needed at both, reference and test sets. With the purpose of further exploiting the dataset and avoiding collisions (i.e. occurrences of an identical sample within the reference and test sets, on a particular configuration), the experimentation with only legitimate samples implemented a cross-validation scheme. The different boot sequences related with the execution of each application were divided into three disjointed groups: *A, B*, and *C*. From them three different scoring vectors were generated (V_1, V_2, V_3):

- V_1 contains the scores obtained by aligning all the samples within *A* and *B*.
- V_2 contains the scores obtained by aligning all the samples within *A* and *C*.
- V_3 contains the scores obtained by aligning all the samples within *B* and *C*.

Summarizing, each vector contains the scores required when analyzing the remaining group. For example, to calculate the false positive rate when studying boot sequences within group *C*, the scoring vector V_1 is taken as reference. This is because it is built from legitimate samples within *A* and *B*, thus taking advantage of the complete dataset. Note that in this case, it would not be honest to analyze sequences within *A*, which already were considered at calibration stage. The evaluation of the proposal is divided into three stages:

1. *In-depth study of a particular use case*. The first test is an example that aims on facilitating the comprehension of the proposed method and the importance of the sequence preprocessing tasks. During its progress, the application *DailyMoney* was analyzed at both, legitimate and malicious executions. Therefore, the results obtained by aligning the various boot sequences were studied, and the relationships between score vectors were discussed. Although the study focused on a simple application, the results were very similar to those of the others, plainly illustrating the problem surrounding the data gathering process. In addition, the advantages on information preprocessing were proven.
2. *Variations on the confidence interval*. At this experiment the accuracy of the proposal in function of the rigor with which decisions were made, is studied. All the samples were analyzed, and effectiveness metrics were calculated for each adjustment value. It is expected that the higher is the confidence interval, the lower True Positive Rate (TPR) and False Positive Rate (FPR). The confidence interval that minimizes precision errors was applied for the remainder of the experimentation.
3. *Variations on the length of the boot sequences*. In this test the behavior of the approach when modifying the length of the boot sequences was measured. Its interest lies in the possibility that when dealing with short sequences, the boot activities of the applications are not completely represented in the captures. This situation makes the detection tasks difficult, because the payload of the threat could not be represented. In the opposite scenario, if the sequences are too long they could include redundant information, which also worsens the quality of the decisions. At this experiment all the samples within the dataset were analyzed, and effectiveness metrics were calculated and discussed for each length.

The proposed detection system behaves satisfactorily if the tests demonstrate that the requirements extensively described in Section 5 are met: accuracy when identifying threats, to report low amount of detection errors when inspecting legitimate applications and to adaptat to the protected environment characteristics.



(a) Legitimate. (b) Malicious.

(c) Legitimate preprocessed. (d) Malicious preprocessed.

Fig. 7. Alignment scores in *DailyMoney*.

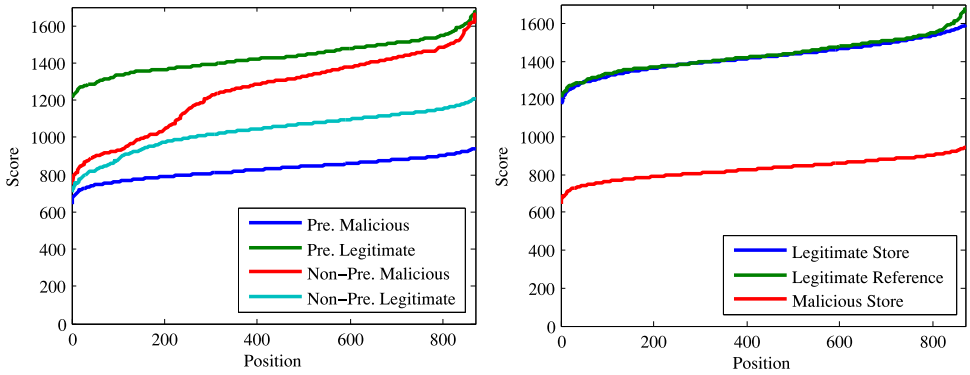
7. Results

7.1. In-depth study of a particular use case

For this experiment, a vector of reference scores was built considering 30 samples of executions of the application *DailyMoney*. The length of their boot sequences was 2500 system calls, and the confidence interval of the Wilcoxon signed-rank test was $I = 0.001$. In a first evaluation step, the data preprocessing capabilities previously described in Section 5 (i.e. reducing the consecutive repetitions of system calls into one action) were not implemented. The various alignments performed for the construction of the vector of reference scores are shown in Fig. 7(a), where rows and columns indicate the legitimate boot sequences of the reference sample set, so each cell contains the score on each partial alignment. The scores are displayed according to the color chart which is shown in the legend. On the other hand, in Fig. 7(b) a similar structure is shown, but this time the rows that indicate the boot sequences of the same application include harmful payload. To the naked eye it is possible to realize that the difference in scores is fairly representative. Analytically, the average score in the first structure approached 1248.9 while in the second was 1033.4. With this scheme it was possible to successfully detect 100% of the inspected malicious samples. However, the accuracy obtained when analyzing legitimate applications was not satisfactory enough; the FPR was close to 32%, thus hindering the deployment of the proposal on real monitoring environments.

Although the example is shown for a specific application, the false positive problem has persisted throughout the experimentation. It only has been reduced by data preprocessing as indicated in Section 5. In this way a large part of the observed noise is mitigated during the capture tasks. The method leads to discover important differences between sequences from different devices. To illustrate this, in Fig. 7(c) the scores of the alignment of legitimate sequences (analogous to Fig. 7(a)) are shown, and in Fig. 7(d) the malicious executions (similar to Fig. 7(b)) are displayed, all of them after applying preprocessing. At first glance, it can be observed that at the legitimate matrix of the latter case, the darker colors dominate (and hence, the higher scores). In general terms, the scores are higher, with less variation, and reached an average score of 1431.6. In addition, for the harmful boot sequences the results were also lower, averaging 829.6. In this case the TPR approached 98% and FPR was close to 2%. Therefore their difference of means is greater, which facilitates the decision-making orchestrated by the Wilcoxon test.

On the other hand, the increases of the paired scores after ordering their mean values are shown in Fig. 8(a). As can be seen after preprocessing the score vectors, they pose a more homogeneous distance, which less frequently may lead to labeling mistakes. But without preprocessing, a significant convergence region



(a) Preprocess vs non-preprocess. (b) Range of scores.

Fig. 8. Score vectors in *DailyMoney*.

appeared. Its location has changed along the experimentation depending on the application. Finally, in Fig. 8(b) different vectors of average and sorted scores, related to the analysis of *DailyMoney* are shown: the vector of reference scores built during training (*Legitimate Reference*), the vector of mean scores obtained by analyzing instances of the legitimate application once download from the application store (*Legitimate Store*) and the vector of average scores obtained when analyzing compromised instances (*Malicious Store*). There the similarity between the first two and their significant distance with the third vector is obvious.

7.2. Variations on the confidence interval

At this test, the system accuracy has been measured by analyzing its reactions to variations on the confidence interval in range 0.002-0.0000004 and assuming a fixed sequence length of 1000 system calls. In Fig. 9(a) the resulting average hit rates are shown. When the confidence interval has lowest error tolerance (i.e. close to cero), the TPR value is nethermost. In particular, at the displayed range the worst case implies approximate TPR of 80%. On the contrary, the higher is the error tolerance, the greater precision. The system behaves with better accuracy when TPR is near 99.6%.

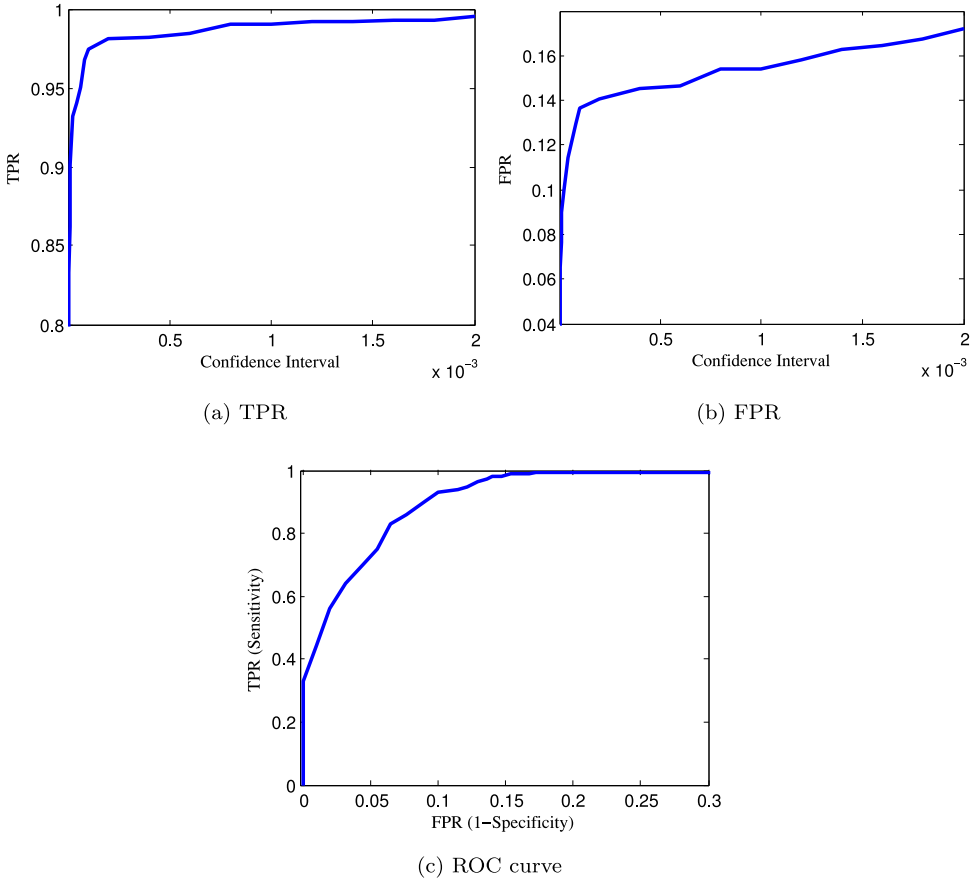
On the other hand, in Fig. 9(b) the relationship between false positives and the value of the confidence interval is shown. The lower is the error tolerance, the better the result. So when the confidence interval is close to zero, the FPR approximates 4%. Analogously, the higher is the error tolerance; the worst is the system behavior. The worse TPR observed is close to 17.19%. At this test, the optimal balance between TPR and FPR was obtained when the confidence interval approached 0.03, where the TPR was 95.01% and the FPR was 10.03%. The obtained results are summarized in Fig. 9(c), when they are displayed on the Receiver Operating Characteristic (ROC) space. The obtained Area Under Curve (AUC) is 96.70%, which was calculated via trapezoidal estimation.

7.3. Variations on the length of the boot sequences

With the aim of determining the impact of the sequence length at the system accuracy, the range of 50–2500 system calls per boot sequence was analyzed. In Fig. 10(a) the dependence of the obtained precision to this parameter is described by the following three curves: the best hit rate per length (*Max TPR*), the average accuracy (*Average TPR*) and the worst cases (*Min TPR*). There, it can be seen that the detection system requires monitoring a minimum number of actions in order to work properly. In particular, in *Max TPR* this occurs from 500 system calls, and the hit rate approached 100%. In the case of *Average TPR* it happens after 750 actions, with 95.37% accuracy. Finally, in *Min TPR* stability is observed from 1400 system calls, with 59.33% precision. Despite the fact that *Min TPR* presented discouraging results, these occur very infrequently. Indeed, the average difference between *Max TPR* and *Average TPR* is less representative (approximately 0.14427) than that observed between *Average TPR* and *Min TPR* (about 0.35785). In general terms, the trend observed throughout the experiment has much to do with the sequence alignment algorithm and the distribution of the malicious contents on the boot processes of the applications. The payload of the attack is not released until the beginning of the basic functionality of the application, situation that affects the required amount of actions to be monitored in order to success. Only in this case the local sequence alignment algorithm is capable of identifying regions of divergence between sequences, thus allowing the presence of anomalies to be reported.

Similarly, in Fig. 10(b) the relationship between the false positive rate and the length of sequences is shown. Unlike the previous experiment, from the outset regularity has been observed (except for *Max FPR*). *Min FPR* is stabilized before analyzing the first 500 system calls, with a false positive rate of 1.7%. *Average FPR* is also stabilized before the first 500 activities; however its false positive rate is close to 11.7%. Finally, *Max FPR* displays less regularity. Its values vary continuously in an interval ranging from 24.44% to 47.61%. As in the previous test, *Average FPR* has greater similarity to the optimal values, but in this case they are described by *Min FPR*; their average difference was 0.1105. However their mean difference with *Max FPR* is considerably higher, approaching 0.2356. The early saturation of these curves is mainly due to the lack of divergence regions within the monitored sequences, contrary to what occurs when they include malicious content. Therefore, differences between sequences are global and constant throughout the boot process. They are mainly due to the noise caused by the context in which the applications are launched on the sandbox. Note that because there is not a significant trade-off between true positives and false positives related with the length of the monitored sequences, the results of varying the reference sequence lengths were not analyzed based on the ROC space.

In Fig. 10(c) the worst response times measured per length since the mobile devices transmit the system call sequences to the dedicated server, until the results of their analysis are received, are summarized. Therefore, the time displayed on the graph is the addition of the delays observed at feature extraction, analysis, decision-making and notification to the protected system. In general terms they grow exponentially, being 0.45 s the worst registered delay when operating at saturation length. In the opposite, when the monitored sequences included 2500 actions, the worst observed delay was 1.67 s. Bearing in mind that the implemented global alignment method potentially involves the greater computa-



(a) TPR (b) FPR

(c) ROC curve

Fig. 9. Average variations on confidence interval

tional impact, i.e. $\Theta(n^2)$, it is reasonable to fit the measurements to a polynomial curve of degree two. This has been accomplished by assuming an error lower than 0.015, from which it is possible to estimate the impact in terms of performance involved at processing sequences of larger dimensions. According to this calculation, the inferred delay when length 5000 is 7.25 s, when 25,000 is almost one minute, and when 50,000 approaches 11 min. This demonstrated the feasibility of the proposal when properly calibrated, but it also indicates the high computational cost derived from not applying filtering measures or reductions of the search space, being the second a common drawback of the dynamic-based analysis approaches in the bibliography.

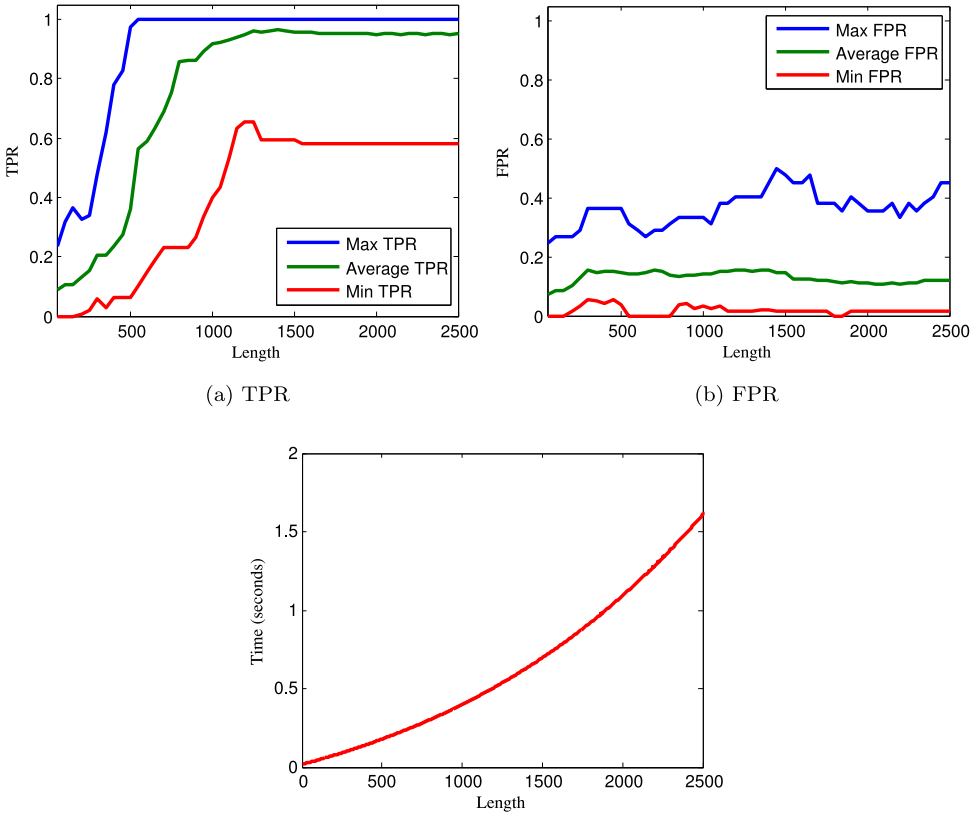
In view of the obtained results, it is possible to conclude that both confidence interval and sequence length, have a direct impact on the system sensitivity. Because of this, they must adjust to each device at every use case. Once stabilized the detection capability of the sensor (in this, case the saturation length was 2000), the average hit rate when studying applications with malicious content was 98.61%, and the average false positive rate was 6.88%. The AUC in ROC space was 96.2%

7.4. Discussion and comparison

Keeping the results obtained in mind, the following items are highlighted:

1. If the system is properly configured, the results in terms of accuracy demonstrate high precision when dealing with malicious applications (TPR=0.9861), and the false positive rate is 6.88%.
2. When preprocessing the score vectors, they pose a more homogeneous distance, which less frequently may lead to labeling mistakes.
3. The lower is the error tolerance of the approach (i.e. lower confidence interval), the better the result in terms of identification of malicious samples. But this penalizes the false positive rate.
4. The length of the sequences should be large enough to reach the saturation lenght. If it must be fixed because of computational limitations, the larger sequences, the greater true positive rate. The false positive rate is independent of the length, and it has much to do with the noise generated during the extraction of the sequences.

These statements demonstrate that by studying sequences of actions at the booting process of the applications, and hence taking into account their temporal relationship, it is possible to effectively recognize malware. The proposed pattern recognition method has proven accuracy at this task. But it demonstrated sensitivity to the adjustment parameters and data preprocessing. On the other hand, the comparison of the observed accuracy with the effectiveness of the most precise publications in the bibliography, states that the malware recognition capabilities of the approach are similar (see Table 1). This is a very important point considering that our proposal requires less information, and that its search space is reduced to only consider boot actions of the applications. However, the main disadvantage on the results is a trend to issue a higher rate of false positives than the proposals that perform the analysis of the complete actions [18,55], were the result ranges on 0.1–14.8%. It is noteworthy to mention that our method does not compete directly with these publications. This paper introduces a novel first line of defense based on analyzing only boot sequences, which pose a different monitoring scenario, with a priori less likely to recognize malware. Therefore, it is perfectly able to complement other general-purpose schemes. On the other hand, it is important



(a) TPR (b) FPR

(c) Worst response time

Fig. 10. Average variations on length.

Table 1
Comparison of results with previous publications.

Proposal	Method	TPR (%)	FPR (%)	Accuracy (%)	AUC (%)	Dataset
[4]	Drebin	93	1	93.9	N/A	Varios App. distribution markets, forums, security blogs and Genome
[47]	MADAM	96.9	0.2	N/A	N/A	Contagio Mobile, VirusShare and Genome
[53]	DroidSieve	N/A	N/A	98.12	N/A	McAfee, Drebin, Genome and Marvin
[58]	Patronus	N/A	N/A	87	N/A	Google Play, Contagio Mobile and Genome
[66]	DroidSIFT	93	5.15	N/A	N/A	McAfee and Genome
[54]	AlterDroid	N/A	N/A	97	N/A	RAMP Competition and Genome
[14]	User-trigger dependences	97.9	2	N/A	N/A	Google Play, VirusShare and Genome
[2]	STEAM	97.6	14.8	N/A	N/A	VirusTotal, Google Play and Genome
[42]	Ranking Risks	N/A	0.4	N/A	0.94	VirusTotal, Google Play and Genome
[1]	DroidAPIminer	97.8	2.2	99	N/A	Google Play, McAfee and Genome
[32]	Payload Mining	75	N/A	90	N/A	Genome
[60]	Dependence graphs	99.65	2.97	N/A	N/A	Google Play, VirusShare and Genome
[26]	PIndroid	98	0.2	99.8	N/A	Drebin and Genome
This proposal	Boost sequences	98.61	6.88	95.8	96.20	Drebin and Genome

to highlight that in the bibliography, the evaluation methodology has significantly varied. Table 1 compiles some of the most relevant publications that mainly considered the Genome dataset in their evaluation. However, samples within Genome have been usually repacked or mixed with applications from other sources (both legitimate and malicious instances), among them different distribution markets (official and third-party), forums, blogs, competitions or published by private information security organizations. Because of this, we have not found publications which evaluation were based on exactly the same dataset.

On the other hand, the performance of the defensive deployment varied depending on the data processing stage. At monitoring, the sandboxing capabilities provided by the Android operating system have been exploited. Based on these technologies, the data collection time entailed a delay of few seconds, which depends on the length of the action sequences and the resources available in the protected device. But given that this drawback lies on hardware, implementation and Android version (where the sandboxing strategy differs), and that it is common in every publication based on dynamical analysis, it has not impact on the comparison of the performed contribution with similar approaches in the bibliography. On the other hand, the transmission of information is driven by an API REST that communicates the protected devices with a dedicated server, hence sending logs as that illustrated in Fig. A.11. The longest log files contained several KB (usually between 100 KB and 300 KB), so the transmission time delayed in the order of milliseconds (note that this parameter may greatly vary depending on the characteristics of the network, which is out of the scope of this study). Finally, and as discussed in Section 5, due that complete sequences have not been considered, and that at Bootstrap

Aggregation they are aligned in concurrency, the execution time of the analytic tasks is of little significance (usually in the order of milliseconds). Therefore, by adding all the aforementioned delays, in the worst case the total time delay is a few seconds (normally, in the order of milliseconds). It has mainly focused at the monitoring stage due to the interaction of the suspicious applications with the Android sandbox, as is common in detection strategies based on dynamic analysis. In addition, the communication with the server and the sequence analysis stage seem to pose a slightly cost in terms of Quality of Experience (QoE).

From an empirical point of view, our experimentation allows us to be optimistic about the effectiveness of the approach. On theoretical grounds, other interesting details should be taken into account. For instance, the introduced strategy inherits the characteristics of the sequence alignment algorithms. Because of this, it is capable of identifying high-level patterns within the alignment elements [10]. This is a relatively unexplored field to address malware recognition on mobile devices, which fits very well with the characteristics of the data that is usually studied. It also should be borne in mind that in particular, this proposal takes advantage of the global alignment methods, which usually yield better results when it is supposed that the sequences should be almost equal, thus providing an overview of the startup features. In return, the proposal is penalized in terms of resource consumption: the Needleman–Wunsch method is implemented by dynamic programming, so the memory of the system must store large data arrays. The problem of limited resources is present throughout the bibliography, and in our approach it is addressed in the same way as in most of the related works: by performing tasks involving higher costs on dedicated servers [6,57]. Hence no analytical tasks are carried out in the device, but data extraction is performed. As discussed in Section 5 , the protected devices must provide access to the network; a minor inconvenience considering that the programs to be analyzed are downloaded from application stores, and that the extracted information is preprocessed and aligned with sequences of actions steadily updated by other trusted users, both of them involving connectivity.

8. Conclusions

A novel intrusion detection system for malware recognition on Android mobile devices has been proposed. Its main goal is to prevent the installation of the malicious software on the victim system once downloaded from application stores. To do this, the boot sequences of system calls performed by the suspicious applications are studied in-deep. It involves an elaborate data processing approach that includes the capture of system call sequences in an isolated and secure environment, their analysis by sequence alignment methods, and the decision of their nature by statistical hypothesis testing. The experimentation performed three different tests, where the analyzed samples of both, legitimate and malicious applications, were provided by the datasets Genome and Drebin. The preliminary results demonstrated that to study boot sequences is a lightweight and efficient complement to the conventional dynamic analysis schemes, that poses an unused first line of defense.

However, the experimentation also exposed sensitivity to changes in the major adjustment parameters (i.e. sequence length and confidence interval). In addition, a slightly trend towards the emission of higher false positive rates is observed. This is not a surprising fact, since the performed analysis comprised a smaller amount of information (only system call sequences collected from the booting activities of the monitored applications) than most of the publications in the bibliography. Therefore, the proposed malware detection system is limited to the set of specimens that trigger its infection vector when the application is first started. The best approaches for tackling this drawback seem to be the improvement of the information provided by the reference datasets, and to directly deepen into the analytic methods looking for effectiveness enhancement. The first may be achieved by implementing advanced data preprocessing techniques that allow sample weighing, in this way taking greater account of the actions that are most likely part of the intrusion processes. This also facilitates the adoption of less generalist scoring systems and makes it possible to manage the discarding of the redundant actions that entail a greater quantity of noise. On the other hand, the upgrades on the analytic steps could be driven by exploiting concurrency at the pattern recognition tasks. Thus, subsequences of the original booting sequence may be studied in parallel, which are selected when they contain suspicious elements (e.g. system calls related with memory modifications, privilege gain, searches for certain file extensions, etc.). Along with the article, different alternatives for future research have been proposed, including, straightening against adversarial attacks, the implementation of a centralized version of the approach, or the discussion about how to obtain/share the information required by the analytic tasks, which may adapt reputation-based schemes, and raise solutions related with data protection. It is important to highlight that the main contributions of this paper are integrated into the RAMSES framework (H2020-FCT-2015/700326) for tracking the money flow of financially-motivated malware. In particular, the approach is part of a solution for detecting malicious applications on mobile devices (mainly different ransomalware families) based on identifying sequences of actions related to asset encryption, hidden communication processes and fraudulent monetary operations.

Acknowledgments

This work was funded by the European Commission Horizon 2020 Programme under Grant Agreement number H2020-FCT-2015/700326-RAMSES (Internet Forensic Platform for Tracking the Money Flow of Financially-Motivated Malware).

Appendix A. Examples of sequence alignment

```
27155 <... ioctl resumed> , 0x7898dc48) = 0
27155 write(33, "27155", 5)             = 5
27155 getpriority(PRIO_PROCESS, 27155)  = 20
27155 getpid()                          = 27144
27155 clock_gettime(CLOCK_MONOTONIC, {5586, 650596065}) = 0
27155 write(33, "27155", 5)             = 5
27155 ioctl(38, 0xc0186201 <unfinished ...>
27156 <... ioctl resumed> , 0x78a92c48) = 0
27156 write(33, "27156", 5)             = 5
27156 getpriority(PRIO_PROCESS, 27156)  = 20
27156 clock_gettime(CLOCK_MONOTONIC, {5586, 651749170}) = 0
27156 write(33, "27156", 5)             = 5
27156 ioctl(38, 0xc0186201 <unfinished ...>
27144 <... ioctl resumed> , 0xbfa631ec) = 0
27144 ioctl(38, 0xc0186201, 0xbfa631ec) = 0
27144 ioctl(38, 0xc0186201, 0xbfa631ec) = 0
27144 ioctl(38, 0xc0186201, 0xbfa631ec) = 0
27144 ioctl(38, 0xc0186201 <unfinished ...>
27158 <... ioctl resumed> , 0x78cd19bc) = 0
27158 getpriority(PRIO_PROCESS, 27158)  = 20
27158 ioctl(38, 0xc0186201, 0x78cd17fc) = 0
27158 ioctl(38, 0xc0186201 <unfinished ...>
27155 <... ioctl resumed> , 0x7898dc48) = 0
27155 write(33, "27155", 5)             = 5
27155 getpriority(PRIO_PROCESS, 27155)  = 20
27155 futex(0x7100e60c, FUTEX_WAIT_PRIVATE, 0, NULL <unfinished ...>
27144 <... ioctl resumed> , 0xbfa631ec) = 0
27144 futex(0x7100e60c, FUTEX_WAKE_PRIVATE, 1) = 1
27144 gettimeofday( <unfinished ...>
27155 <... futex resumed> )             = 0
27155 clock_gettime(CLOCK_MONOTONIC, {5586, 681311899}) = 0
27155 write(33, "27155", 5 <unfinished ...>
27144 <... gettimeofday resumed> {1433878441, 895602}, NULL) = 0
27144  mprotect(0x4343d000,  4096,  PROT_READ|PROT_WRITE  <unfinished
...>
27155 <... write resumed> )             = 5
```

Fig. A1. Example of log of system calls generated when monitoring *Monkey Jump 2*.

Table A1
Example (L1) of first 120 actions monitored when booting *Monkey Jump 2*.

No.	Action	No.	Action	No.	Action	No.	Action
1	futex	31	futex	61	set_thread_area	91	futex
2	futex	32	futex	62	futex	92	futex
3	futex	33	futex	63	mmap2	93	prctl
4	select	34	futex	64	prctl	94	mmap2
5	ioctl	35	futex	65	madvise	95	prctl
6	recvmsg	36	futex	66	prctl	96	madvise
7	ioctl	37	munmap	67	sigaltstack	97	prctl
8	futex	38	sigaltstack	68	prctl	98	mmap2
9	futex	39	munmap	69	prctl	99	prctl
10	futex	40	sigprocmask	70	prctl	100	madvise
11	futex	41	munmap	71	gettid	101	prctl
12	futex	42	open	72	prctl	102	mprotect
13	futex	43	getdents64	73	futex	103	prctl
14	futex	44	getdents64	74	prctl	104	clone
15	futex	45	close	75	futex	105	prctl
16	sigaltstack	46	sigaction	76	futex	106	futex
17	futex	47	mmap2	77	prctl	107	futex
18	munmap	48	gettid	78	futex	108	prctl
19	futex	49	madvise	79	futex	109	set_thread_area
20	futex	50	getuid32	80	prctl	110	prctl
21	futex	51	mmap2	81	futex	111	mmap2
22	sigprocmask	52	open	82	futex	112	prctl
23	futex	53	madvise	83	getpriority	113	madvise
24	futex	54	write	84	setpriority	114	prctl
25	munmap	55	mprotect	85	clock_gettime	115	sigaltstack
26	sigaltstack	56	close	86	futex	116	prctl
27	munmap	57	clone	87	prctl	117	prctl
28	sigprocmask	58	prctl	88	prctl	118	prctl
29	munmap	59	futex	89	futex	119	gettid
30	munmap	60	prctl	90	clock_gettime	120	prctl

Table A2
Example (L2) of first 120 actions monitored when booting *Monkey Jump 2*.

No.	Action	No.	Action	No.	Action	No.	Action
1	select	31	sigprocmask	61	mmap2	91	prctl
2	futex	32	futex	62	open	92	futex
3	futex	33	munmap	63	madvise	93	prctl
4	futex	34	futex	64	write	94	futex
5	ioctl	35	futex	65	mmap2	95	prctl
6	recvmsg	36	futex	66	close	96	futex
7	ioctl	37	munmap	67	madvise	97	prctl
8	futex	38	sigaltstack	68	prctl	98	futex
9	futex	39	munmap	69	mprotect	99	prctl
10	futex	40	sigprocmask	70	prctl	100	getpriority
11	futex	41	munmap	71	clone	101	setpriority
12	futex	42	futex	72	futex	102	clock_gettime
13	futex	43	futex	73	prctl	103	futex
14	futex	44	futex	74	set_thread_area	104	prctl
15	futex	45	futex	75	prctl	105	futex
16	sigaltstack	46	futex	76	mmap2	106	clock_gettime
17	futex	47	futex	77	prctl	107	futex
18	munmap	48	futex	78	madvise	108	futex
19	futex	49	munmap	79	prctl	109	prctl
20	sigprocmask	50	sigaltstack	80	sigaltstack	110	mmap2
21	futex	51	munmap	81	prctl	111	prctl
22	sigprocmask	52	sigprocmask	82	prctl	112	madvise
23	futex	53	open	83	prctl	113	prctl
24	munmap	54	getdents64	84	gettid	114	mmap2
25	futex	55	getdents64	85	prctl	115	prctl
26	futex	56	close	86	futex	116	madvise
27	futex	57	sigaction	87	prctl	117	prctl
28	munmap	58	fork	88	futex	118	mprotect
29	sigaltstack	59	gettid	89	prctl	119	prctl
30	munmap	60	getuid32	90	futex	120	clone

Table A3
Example (M1) of first 120 actions monitored when booting *Monkey Jump 2* (binded to *GinMaster*).

No.	Action	No.	Action	No.	Action	No.	Action
1	futex	31	munmap	61	mmap2	91	futex
2	select	32	futex	62	close	92	mmap2
3	futex	33	sigprocmask	63	madvise	93	prctl
4	futex	34	futex	64	prctl	94	madvise
5	ioctl	35	munmap	65	mprotect	95	prctl
6	recvmsg	36	futex	66	prctl	96	mprotect
7	ioctl	37	munmap	67	clone	97	prctl
8	futex	38	open	68	prctl	98	clone
9	futex	39	sigaltstack	69	futex	99	futex
10	futex	40	getdents64	70	prctl	100	futex
11	futex	41	munmap	71	prctl	101	set_thread_area
12	futex	42	getdents64	72	prctl	102	futex
13	futex	43	sigprocmask	73	prctl	103	mmap2
14	futex	44	close	74	prctl	104	madvise
15	futex	45	munmap	75	set_thread_area	105	sigaltstack
16	sigaltstack	46	clock_gettime	76	prctl	106	prctl
17	futex	47	clock_gettime	77	mmap2	107	prctl
18	munmap	48	futex	78	prctl	108	gettid
19	sigprocmask	49	open	79	madvise	109	prctl
20	futex	50	getdents64	80	prctl	110	futex
21	munmap	51	getdents64	81	sigaltstack	111	prctl
22	futex	52	close	82	prctl	112	futex
23	futex	53	sigaction	83	prctl	113	prctl
24	futex	54	fork	84	prctl	114	futex
25	futex	55	gettid	85	gettid	115	prctl
26	futex	56	getuid32	86	prctl	116	futex
27	munmap	57	mmap2	87	futex	117	prctl
28	futex	58	open	88	futex	118	getpriority
29	sigaltstack	59	madvise	89	futex	119	prctl
30	futex	60	write	90	futex	120	setpriority

Table A4
Alphabet for building sequences related with the examples L1, L2, M1.

Symbol	Action	Symbol	Action	Symbol	Action
A	*futex*	J	*getdents64*	S	*prctl*
B	*select*	K	*close*	T	*set_thread_area*
C	*ioctl*	L	*sigaction*	U	*gettid*
D	*recvmsg*	M	*mmap2*	V	*getpriority*
E	*munmap*	N	*madvise*	W	*setpriority*
F	*sigaltstack*	O	*write*	X	*clock_gettime*
G	*sigprocmask*	P	*mprotect*	Y	*fork*
H	*lock*	Q	*linkat*	Z	*getuid*
I	*open*	R	*clone*	–	gap

Table A5
Example of global alignment considering booting sequences (L1, L2 and M1) from *Monkey Jump 2*.

ID	Content	Action sequences
L1	Legitimate	L1(001-060):AAABCDCAAAAAAAAEAFAEAAAGAAEFEGEEAAAAAAEFEGEHIIJKlHMNOJPQAQR
		L1(061-120):ALQMQFQQQSQAQAAQAAQAATUVAQQAVAAQLQMQLQMQOQPQAAQRQLQMQFQQQSQ
L2	Legitimate	L2(001-060):BAAACDCAAAAAAAAEAFAEAGAGAEAAAEFEGAEAAAEFEGEAAAAAAAEFEGHIIJK
		L2(061-120):WSZLHMNLPMQOQPAQRQLQMQFQQQSQAQAQAQAQAQTUVAQAVAAQLQMQLQMQOQJ
M1	Malware	M1(001-060):ABAACDCAAAAAAAAEAFAEGAEAAAAAEAFAEAGAEAEHFIEIGJEVVAHIIPKWSZL
		M1(061-120):HMNLJMQOQPQAQQQQQRQLQMQFQQQSQAAAAALQMQOQPAARALMFQQSQAQAQTQU
A1	Align.(L1:L2) Match:84/136 Rate:0.62 Gaps:18	L1(001-060):AAABCDCAAAAAAAAEAFAEAAAG------------AAEFEGEEAAAAAAEFEGEHIIJ
		L2(001-060):BAAACDCAAAAAAAAEAFAEAGAGAEAAAEFEGAEAAAEFEGEAAAAAAAEFEG-HIIJ
		L1(061-120):K---lHMNOJPQ---AQRALQMQFQQQSQAQAAQAAQAATUVAQQAVAAQLQMQLQMQO
		L2(061-120):KWSZLHMNLPMQOQPAQRQLQMQFQQQSQAQAQAQAQAQTUVA-QAVAAQLQMQLQMQO
		L1(121-136):QPQAAQRQLQMQFQQQSQ
		L2(121-136):QJ----------------
A2	Align.(L1:M1) Match:68/149 Rate:0.45 Gaps:31	L1(001-060):AAABCDCAAAAAAAAEAFAEAAAGAAEFEGEEAAAAAAEF-----EGE----------
		M1(001-060):ABAACDCAAAAAAAAEAFA---------EGAEAAAAAEAFAEAGAEAEHFIEIGJEVVA
		L1(061-120):HIIJK---lHMN----OJPQA----QRALQMQFQQQSQAQAAQAAQAATUVAQQAVAAQ
		M1(061-120):HIIPKWSZLHMNLJMQOQPQAQQQQQRQLQMQFQQQSQAAAA----------------
		L1(121-149):LQMQLQMQOQPQAAQRQL---QMQFQQQSQ-
		M1(121-149):---ALQMQOQP--AARALMFQQSQAQAQTQU

References

[1] Y. Aafer, W. Du, H. Yin, DroidAPIMiner: mining API-level features for robust malware detection in android, in: Proceedings of the International Conference on Security and Privacy in Communication Systems (SecureComm), Guangzhou, China, 2013, pp. 86–103.

[2] B. Amos, H. Turner, J. White, Applying machine learning classifiers to dynamic Android malware detection at scale, in: Proceedings of the 9th International Wireless Communications and Mobile Computing Conference (IWCMC), Sardinia, Italy, 2013.

[3] A. Apvrille, The evolution of mobile malware, Comput. Fraud Secur. 8 (2014) 18–20.

[4] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, Drebin: effective and explainable detection of android malware in your pocket, in: Proceedings of the 21th Annual Symposium on Network and Distributed System Security (NDSS), San Diego, CA, US, 2014, pp. 1–12.

[5] L. Breiman, Bagging predictors, Mach. Learn. 24 (2) (2006) 123–140.

[6] I. Burguera, U. Zurutuza, S. Nadjm-Tehrani, Crowdroid: behavior-based malware detection system for android, in: Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, Chicago, IL, US, 2011, pp. 15–26.

[7] J. Chen, M. Alafi, Detecting android malware using clone detection, J. Comput. Sci. Technol. 30 (5) (2015) 942–956.

[8] Cisco, Cisco 2014 annual security report, 2015, (http://www.cisco.com).

[9] S. Coull, J. Branch, B. Szymanski, E. Breimer, Intrusion detection: a bioinformatics approach, in: Proceedings of the 19th Annual Computer Security Applications Conference (CSAC), Las Vegas, NV, US, 2003, pp. 24–33.

[10] S. Coull, B. Szymanski, Sequence alignment for masquerade detection, Comput. Stat. Data Anal. 52 (8) (2008) 4116–4131.

[11] S. Coursen, The future of mobile malware, Netw. Secur. 2007 (8) (2007) 7–11.

[12] G. Ditzler, M. Roveri, C. Alippi, R. Polikar, Learning in nonstationary environments: a survey, IEEE Comput. Intell. Mag. 10 (4) (2015) 12–25.

[13] Q. Do, B. Martini, K.K.R. Choo, Exfiltrating data from android devices, Comput. Secur. 45 (2015) 74–91.

[14] K.O. Elish, X. Shu, D. Yao, B.G. Ryder, X. Jiang, Profiling user-trigger dependence for android malware detection, Comput. Secur. 29 (2015) 255–273.

[15] Europol, Police ransomware threat assessment, 2014, (http://www.europol.europa.eu).

[16] Europol, The internet organised crime threat assessment (iOCTA), 2017, (http://www.europol.europa.eu).

[17] P. Faruki, A. Bharmal, L. V., Android security: a survey of issues, malware penetration, and defenses, IEEE Commun. Surv. Tutor. 17 (2) (2015) 998–1022.

[18] P. Faruki, V. Laxmi, M.S. Bharmal, A. Gaur, V. Ganmoor, Androsimilar: robust signature for detecting variants of android malware, J. Inf. Secur. Appl. 22 (2015) 66–80.

[19] A. Feizollah, N.B. Anuar, R. Salleh, A.W.A. Wahab, A review on feature selection in mobile malware detection, Digit. Invest. 13 (2015) 23–37.

[20] Flask, Flask: a python microframework, 2018, (http://flask.pocoo.org).

[21] T. Gaffney, Following in the footsteps of windows: how android malware development is looking very familiar, Netw. Secur. 2013 (8) (2013) 7–10.

[22] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, V. E., Anomaly-based network intrusion detection: techniques, systems and challenges, Comput. Secur. 25 (1–2) (2009) 18–28.

[23] D. Geneiatakis, I.N. Fovino, I. Kounelis, P. Stirparo, A permission verification approach for android mobile applications, Comput. Secur. 49 (2015) 195–205.

[24] J. Hoffmann, S. Neumann, T. Holz, Mobile malware detection based on energy fingerprints – a dead end? in: Proceedings of the 16th International Symposium of Research in Attacks, Intrusions, and Defenses (RAID), Rodney Bay, St. Lucia, 2013, pp. 348–368.

[25] P.C. Huang, S.S. Lee, Y.H. Kuo, K.R. Lee, A flexible sequence alignment approach on pattern mining and matching for human activity recognition, Expert Syst. Appl. 37 (1) (2010) 298–306.

[26] F. Idrees, M. Rajarajan, M. Conti, T.M. Chen, Y. Rahulamathavan, PIndroid: a novel android malware detection system using ensemble learning methods, Comput. Secur. 68 (2017) 36–46.

[27] J.W. Jang, H. Kang, J. Woo, A. Mohaisen, H.K. Kmin, Andro-autopsy: anti-malware system based on similarity matching of malware and malware creator–centric information, Digital Invest. 14 (2015) 17–35.

[28] D. Kim, P. Yan, J. Zhang, Detecting fake anti-virus software distribution webpages, Comput. Secur. 49 (2015) 95–106.

[29] S.B. Kotsiantis, I.D. Zaharakis, P.E. Pintelas, Machine learning: a review of classification and combining techniques, Artif. Intell. Rev. 26 (3) (2003) 159–190.

[30] C. Kuiken, H. Yoon, W. Abfalterer, B. Gaschen, C. Lo, B. Korber, Viral genome analysis and knowledge management, Methods Mol. Biol. 939 (2013) 253–261.

[31] M. La Polla, F. Martinelli, D. Sgandurra, A survey on security for mobile devices, IEEE Commun. Surv. Tutor. 15 (1) (2013) 446–471.

[32] Y. Li, J. Jang, X. Hu, X. Ou, Android malware clustering through malicious payload mining, in: Proceedings of the 20th International Symposium on Research in Attacks, Intrusions and Defenses (RAID), Atlanta, GA, US, 2017, pp. 192–214.

[33] Y.D. Lin, Y.C. Lai, C.H. Chen, H.C. Tsai, Identifying android malicious repackaged applications by thread-grained system call sequences, Comput. Secur. 39 (2013) 340–350.

[34] M. Lindorfer, S. Volanis, A. Sisto, M. Neugschwandtner, E. Athanasopoulos, F. Maggi, C. Platzer, S. Zanero, S. Ioannidis, AndRadar: fast discovery of android applications in alternative markets, in: Proceedings of the 11th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA), Egham, UK, 2014, pp. 57–71.

[35] D. Maiorca, D. Ariu, I. Corona, G. Giacinto, Stealth attacks: an extended insight into the obfuscation effects on android malware, Comput. Secur. 51 (2015) 16–31.

[36] L. Marinos, A. Sfakianakis, Threat landscape 2014, 2014, (http://www.enisa.europa.eu/).

[37] S.B. Needleman, C.D. Wunsch, A general method applicable to the search for similarities in the amino acid sequence of two proteins, J. Mol. Biol. 48 (3) (1970) 443–453.

[38] C. Oehmen, J. Nieplocha, Scalablast: a scalable implementation of BLAST for high-performance data-intensive bioinformatics analysis, IEEE Trans. Parallel Distrib. Syst. 17 (8) (2006) 740–749.

[39] Openstack, Openstack: open source sotware for creating private and public clouds, 2018, (http://www.openstack.org).

[40] S. Pabinger, A. Dander, M. Fischer, R. Snajder, M. Sperk, M. Efremova, B. Krabichler, R. Speicher, J. Zschocke, Z. Trajanoski, A survey of tools for variant analysis of next-generation genome sequencing data, Brief. Bioinform. 15 (2) (2013) 256–278.

[41] R. Pandita, X. Xiao, W. Yang, W. Enck, T. Xie, WHYPER: towards automating risk assessment of mobile applications, in: Proceedings of the 22nd USENIX Conference on Security, Washington, D.C, US, 2013, pp. 527–542.

[42] H. Peng, C. Gates, B. Sarma, N. li, Y. Qi, R. Potharaju, C. Nita-Rotaru, I. Molloy, Using probabilistic generative models for ranking risks of Android apps, in: Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS), Raleigh, NC, US, 2012, pp. 241–252.

[43] S. Peng, S. Yu, A. Yang, Smartphone malware and its propagation modeling: a survey, IEEE Commun. Surv. Tutor. 16 (2) (2013) 925–941.

[44] A.M. Retenaga, Android malware situation 2015, 2015, (https://www.incibe.es/CERT_en/publications/Studies/Android_malware_situation_en).

[45] B.P. Rocha, M. Conti, S. Etalle, B. Crispo, Hybrid static-runtime information flow and declassification enforcement, IEEE Trans. Inf. Forensics Secur. 8 (8) (2013) 1294–1305.

[46] K. Rodrigues, M. Cristo, E.S. de Moura, A. da Silva, Removing DUST using multiple alignment of sequences, IEEE Trans. Knowl. Data Eng. 27 (8) (2015) 2261–2274.

[47] A. Saracino, D. Sgandurra, G. Dini, F. Martinelli, MADAM: effective and efficient behavior-based android malware detection and prevention, IEEE Trans. Dependable Secur. Comput. (99) (2016) 1.

[48] J.L. Schnase, D.Q. Duffy, G.S. Tamkin, D. Nadeau, J.H. Thompson, J.H. Grieg, M.A. McInerney, W.P. Webster, MERRA analytic services: meeting the big data challenges of climate science through cloud-enabled climate analytics-as-a-service, Comput. Environ. Urban Syst. 61 (2017) 198–211.

[49] Z.C. Schreuders, T. McGill, C. Payne, The state of the art of application restrictions and sandboxes: a survey of application-oriented access controls and their shortfalls, Comput. Secur. 32 (2013) 219–241.

[50] A. Shabtai, L. Tenenboim-Chekina, D. Mimran, L. Rokach, B. Shapira, Y. Elovici, Mobile malware detection through analysis of deviations in application network behavior, Comput. Secur. 43 (2014) 1–18.

[51] S. Shenn, R. Anitha, V. Natarajan, Android based malware detection using a multifeature collaborative decision fusion approach, Neurocomputing 151 (2) (2015) 905–912.

[52] T.F. Smith, M.S. Waterman, Identification of common molecular subsequences, J. Mol. Biol. 147 (1) (1981) 195–197.

[53] G. Suarez-Tangil, S. Dash, M. Ahmadi, L. Cavallaro, DroidSieve: fast and accurate classification of obfuscated android malware, in: Proceedings of the of the 7th ACM on Conference on Data and Application Security and Privacy (CODASPY), Scottsdale, AZ, US, 2017, pp. 309–320.

[54] G. Suarez-Tangil, J.E. Tapiador, F. Lombardi, R. Di Pietro, Thwarting obfuscated malware via differential fault analysis, Comput. (Long Beach Calif.) 47 (6) (2014) 24–31.

[55] G. Suarez-Tangil, J.E. Tapiador, P. Peris-Lopez, Evolution, detection and analysis of malware for smart devices, IEEE Commun. Surv. Tutor. 16 (2) (2014) 961–987.

[56] G. Suarez-Tangil, J.E. Tapiador, P. Peris-Lopez, J.B. Alis, Dendroid: a text mining approach to analyzing and classifying code structures in android malware families, Expert Syst. Appl. 41 (4) (2014) 1104–1117.

[57] G. Suarez-Tangil, J.E. Tapiador, P. Peris-Lopez, S. Pastrana, Power-aware anomaly detection in smartphones: an analysis of on-platform versus externalized operation, Pervasive Mob. Comput. 18 (2015) 137–151.

[58] M. Sun, M. Zheng, J.C.S. Lui, X. Jiang, Design and implementation of an android host-based intrusion prevention system, in: Proceedings of the 30th Annual Computer Security Applications Conference (ACSAC), New Orleans, LA, US, 2014, pp. 226–235.

[59] P. Teufl, M. Ferk, A. Fitzek, D. Hein, S. Kraxberger, C. Orthacker, Malware detection by applying knowledge discovery processes to application metadata on the android market (google play), Secur. Commun. Netw. 9 (5) (2016) 389–419.

[60] K. Tian, D.D. Yao, B.G. Ryder, G. Tan, G. Peng, Detection of repackaged android malware with code-Heterogeneity features, IEEE Trans. Dependable Secur. Comput. (99) (2017), doi:10.1109/TDSC.2017.2745575.

[61] T. Vidas, N. Christin, Evading Android runtime analysis via sandbox detection, in: Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security, ASIA CCS'14, New York, NY, US, 2014, pp. 447–458.

[62] R.A. Wagner, M.J. Fischer, The string-to-string correction problem, J. ACM 21 (1) (1974) 168–173.

[63] X. Wei, L. Gomez, I. Neamtiu, M. Faloutsos, ProfileDroid: multi-layer profiling

of Android applications, in: Proceedings of the 18th Annual International Conference on Mobile Computing and Networking (Mobicom), Istambul, Turkey, 2012, pp. 137–148.
[64] F. Wilcoxon, Individual comparisons by ranking methods, Biom. Bull. (1945) 80–83.
[65] L. Xing, X. Pan, R. Wang, K. Yuan, X. Wang, Upgrading your Android, elevating my malware: privilege escalation through mobile OS updating, in: Proceedings of the 35th IEEE Symposium on Security and Privacy (SP), San Jose, CA, US, 2014, pp. 393–408.
[66] M. Zhang, Y. Duan, H. Yi, Z. Zhao, Semantics-aware android malware classification using weighted contextual API dependency graphs, in: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CSS), Scottsdale, AZ, US, 2014, pp. 1105–1116.
[67] Y. Zhang, M. Yang, Z. Yang, G. Gu, Permission use analysis for vetting undesirable behaviors in android apps, IEEE Trans. Inf. Forensics Secur. 9 (1) (2014) 1828–1842.
[68] Y. Zhou, X. Jiang, Dissecting android malware: characterization and evolution, in: Proceedings of the 33rd IEEE Symposium on Security and Privacy (SP), San Francisco, CA, US, 2012, pp. 95–109.