Clustering Android Malware Families by Http Traffic

Marco Aresu, Davide Ariu, Mansour Ahmadi, Davide Maiorca and Giorgio Giacinto
Department of Electrical and Electronic Engineering, University of Cagliari
Piazza d'Armi, 09123, Cagliari, Italy
Email: aresu.ma@gmail.com, {davide.ariu, mansour.ahmadi, davide.maiorca, giacinto}@diee.unica.it

***Abstract*—Due to its popularity and open-source nature, Android is the mobile platform that has been targeted the most by malware that aim to steal personal information or to control the users' devices. More specifically, mobile botnets are malware that allow an attacker to remotely control the victims' devices through different channels like HTTP, thus creating malicious networks of bots. In this paper, we show how it is possible to effectively group mobile botnets families by analyzing the HTTP traffic they generate. To do so, we create malware clusters by looking at specific statistical information that are related to the HTTP traffic. This approach also allows us to extract signatures with which it is possible to precisely detect new malware that belong to the clustered families. Contrarily to x86 malware, we show that using fine-grained HTTP structural features do not increase detection performances. Finally, we point out how the HTTP information flow among mobile bots contains more information when compared to the one generated by desktop ones, allowing for a more precise detection of mobile threats.**

***Keywords*—Android, Botnet, Malware detection, Clustering, HTTP Traffic Network**

I. Introduction

A botnet is a network of compromised machines (bots) commanded and controlled by a bot master for massive attacks such as dispatching unsolicited emails (SPAM), launching Distributed Denial of Service (DDoS) attacks, and performing information theft. Botnets leverage on different approaches such as encrypted HTTP protocol, fast-flux, and domain-flux techniques to be resilient against detection. Botnets may rely on 2 types of command and control (C&C) channels: (i) centralized C&C such as IRC and HTTP; (ii) distributed C&C such as P2P. C&C traffic is hard to identify and to be blocked either at the network level (e.g., by setting appropriate rules on a firewall) or at the DNS level (e.g., by domain blacklisting).

Before the age of smartphones, several studies were performed concerning the detection of malware in mobile networks [1], [2]. Mickens et al. [3] proposed Probabilistic Queuing, a framework that analyzes network nodes to detect infected ones. The prominent work on botnet detection is BotMiner [4], which is a general framework that does not depend on the botnet C&C protocol and structure.

As the source of the user-generated network traffic is moving from desktop computers to mobile devices, mobile malware have become a serious concern that target in particular the Android platform [5]. Android botnets [6] are malware families that take control of Android devices in the same way as malware that are designed to set up a botnet of desktop computers. Commands to mobile bots are sent through different channels such as HTTP, SMS, and the Google Cloud Messaging (GCM) service. Although the Android system itself and mobile anti malware products introduced many security policies and techniques to protect Android devices against malware, mobile botnets are still on the rise.

As large numbers of new mobile malware samples are collected on a daily basis, new techniques are needed for a fast and accurate assessment of the *family* the malware belongs to. We focus our analysis on malware samples that send and receive data using the HTTP channel. We chose HTTP for our study as 70% of the generated network traffic by Android apps is spread through this protocol [7]. In addition, most of the web-based traffic generated by Android apps does not use the HTTPS protocol [8]. In particular, more than 99% of Android botnets use the HTTP-based web traffic to receive bot commands from their C&C servers.

In this paper we show that mobile malware can be effectively clustered and detected on the basis of statistics calculated on the HTTP traffic that is generated by the applications. To do so, we leveraged on a previous work [9] that proposed a technique to cluster and detect malware for desktop architectures. Results show that not only the same rationale is still valid for mobile devices, but also that a simpler system can be used when dealing with mobile malware.

In summary, this paper aims to answer the following research questions:

RQ1 Can we use the HTTP network traffic generated by Android apps to detect malware families?

RQ2 Which features extracted from the HTTP traffic are effective for clustering and detecting Android malware?

RQ3 How well do the performances on Android malware compare to the ones on desktop malware?

Accordingly to such questions, the contributions of the paper are the following:

- We show that the analysis of the HTTP traffic is prominent for the detection of Android malware.
- We propose a malware detection system that is both efficient and effective, as it leverages on seven statistical features that allow for reliably clustering Android malware into families. From such cluster we extract signatures with which it is possible to precisely detect malware belonging to the clustered families.
- The overall performances of the system for mobile malware clustering and detection are better than the ones related to a similar system developed by some of the authors for traditional malware [9]. The reason behind this behavior can be related to the smaller variability in the statistics of the HTTP traffic, as HTTP communications among apps are generally more limited when compared to desktop ones. In addition, mobile botnets have to control less functionalities and applications compared to Desktop ones. In this way, the traffic generated by malware samples

belonging to the same family can be more easily separated from the traffic generated either by benign applications or malicious applications belonging to different families.

The rest of the paper is organized as follows:
Section II provides an overview of the state of the art in mobile malware clustering, as well as the related work on malware clustering; The method used for the experimental evaluation is presented in Section III. Section IV shows the results of the experiments carried out on a significant dataset of Android malware samples. Conclusions are drawn in Section V.

II. Related Work

Many security mechanisms were proposed to detect malicious Android app and protect targets from attacks. Most of the proposed mechanisms are based on analyzing application elements such as permissions, the employed API, or its bytecode. Such mechanisms employ static or dynamic analysis, the latter performed through instrumenting or virtual machine monitoring. Alternatively, applications could be also analyzed by observing the network traffic they generate. In the following, we will briefly review these detection mechanisms.

- **Application analysis** Some static analysis approaches like SEFA [10], ComDroid [11], Epicc [12], Woodpecker [13], and Chex [14] aim to detect Inter-Component Communication vulnerabilities (ICC) by analyzing the application bytecode. Such bytecode is also analyzed by PiggyApp [15] and DroidMOSS [16] to detect repackaged applications. Other approaches such as Drebin [17], DroidMat [18], DroidMiner [19] use learning-based systems to detect anomalies by considering permissions, APIs and bytecode instructions.
  As static analysis can just reveal a subset of malicious content, a number of dynamic analysis systems have been proposed that can be roughly subdivided into two groups: systems based (i) on virtual machine monitoring (VMM), and (ii) on instrumentation. Crowdroid [20], CopperDroid [21], DroidScope [22], and AppsPlayground [23] are VMM-based systems that analyze system calls to detect malicious behaviours. Other examples [24], [25] of these systems profile Android metadata to detect malware. In addition to VMM-based systems, dynamic analysis can be carried out by instrumenting the Android OS (on the source code level) with additional security controls. For example, TaintDroid [26], and Appfence [27] modify the Dalvik virtual machine to perform application taint-analysis.
- **Network traffic analysis** Other approaches explicitly analyze network traffic for different goals. Andromaly [28] is malware detection system that employs machine learning and leverages information such as the CPU usage, active processes, and the amount of transferred data through the network. In another work [29] by the same authors, the analysis was focused on the network traffic generated by Android applications. The main goal of this system was the identification of malicious attacks perpetrated by means of repackaging. The authors showed that applications could be subdivided into a number of categories according to the statistics of their transferred data. They also concluded that deviations from specific normal behaviors could be classified as malicious activity.

  Narudin et al. [30] proposed another detection approach based on both TCP and HTTP protocols. They considered network traffic features such as some basic information from the TCP header (e.g., the frame length), content based features such as the number of HTTP requests, and time-/connection- based features such as the number of frames in a specific time interval or connection. Another nearly similar detection approach [31] was proposed by using a classification system, which is fed just with some time-/connection- based features, and was tested on a small dataset containing 43 malware samples.

  NetworkProfiler [7] is another approach that performs application analysis based on the HTTP header. They generated fingerprints from the network usage of each app, and they were able to use them to detect malicious activities by inspecting the traffic logs produced by a network provider. Subsequently, the same authors resorted to network traces to identify Android applications that use in-app advertisements [32]. Zarras et al. [33] proposed a system that analyzes, among the others, the sequence of headers in HTTP communications to detect malicious traffic generated by botnets. To this end, they extract HTTP traffic generated from both desktop and mobile applications.

  Apart from identifying coarse-grain behaviours such as the presence of maliciousness in a network traffic flow, extracting finer-grain information from the device communications can be interesting for an adversary. Conti et al. [34] designed a system based on network flows analysis and machine learning techniques to identify user actions such as sending an email or posting a message on a friend's wall in online social networks. Wu et al. [35] proposed an approach based on extracting the characteristics of the app from the HTTP traffic to detect repackaged applications on the Android markets.

Approach	Analyzed protocol TCP	HTTP	Purpose
Andromaly [28]	✓		Malware detection
NetworkProfiler [7]		✓	Android fingerprinting
Tongaonkar et al. [32]		✓	Identification of apps with ads
Shabtai et al. [29]	✓		Malware and Repackaging detection
Narudin et al. [30]	✓	✓	Malware detection
Arora et al. [31]	✓		Malware detection
Conti et al. [34]	✓		Identification of user actions
Wu et al. [35]		✓	Detect repackaged apps
The method in this paper		✓	Malware detection

*All the systems use machine learning techniques

Table I. Comparison of different network analysis techniques for Android applications

Table I provides a summary of the aforementioned network-based analysis approaches, and compares them to the characteristics of the method that we employ in this paper. Our method is the first approach that extracts a few statistical features just from the traffic HTTP header for the task of Android malware detection.

III. Android malware clustering by statistical traffic features

We relied on an algorithm that has been proposed by some of the authors for clustering malware that target traditional desktop systems [9], and tested if the proposed scheme was still valid for clustering Android malware on the basis of the HTTP traffic that they generated. The algorithm adopts a multi-step clustering procedure to define the clusters and generate the signatures for malware detection. The multi-step procedure was proposed to speed-up the process, by first using statistical traffic features to perform a coarse- grained clustering, and then by employing a set of structural features (i.e., features that take into account the content of each HTTP connection) to perform a fine-grained clustering. Both the coarse-grained and fine-grained clustering procedures are carried out by resorting to hierarchical clustering techniques, where data is aggregated in nested clusters and the clustering process terminates when further aggregation merges two distant clusters. In the following we will briefly recall these steps:

1) ***Coarse-grained Clustering***: to perform this step, the BIRCH algorithm is employed [36]. The main goal of BIRCH is to perform approximate clustering of arbitrarily large datasets with a guaranteed memory bound and with I/O access costs that grow linearly with the size of the dataset. Whenever the clustering process approaches the preset memory limit, the clustering algorithm will further compress the dataset, thus producing a less fine-grained representation of the data and thus resulting in fewer, larger clusters. The term *coarse grained* refers to the use of simple statistical features extracted from the HTTP traffic to characterize the connections. The seven features that have been used are the following: (i) the total number of HTTP requests, (ii) the number of GET requests, (iii) the number of POST requests, (iv) the average length of the URLs, (v) the average number of parameters in the request, (vi) the average amount of data sent by POST requests and (vii) the average length of the response. The size of a cluster can be measured by its *radius*, whose value can be limited to avoid generating too large clusters that might incorrectly group the connections. After this process, the clusters that are obtained will contain HTTP connections featuring similar statistical features. However, such connections might be related to different malware families. For this reason, each cluster needs to be further refined through a *fine-grained* clustering process, in order to further split the coarse-grained clusters that might contain different families.
2) ***Fine-grained Clustering***: To perform this step, a single-linkage hierarchical clustering algorithm is used and the distance between HTTP requests is computed according to the four parameters: (i) the request method used, (ii) the page, (iii) the set of parameters names, (iv) and the set of parameters values. The reader that is interested in a detailed description of the distance computation employed at this step could check [37].
3) ***Signature Generation***: Whereas clustering allows for grouping together malware samples belonging to the same *family*, further information can be extracted by the samples in the same cluster, leading to the generation of a signature that can be used for detection. To this end, the *Token-Subsequence* algorithm described in [38] can be used to extract a signature from each group of malware. These signatures are then used by a network IDS to perform the detection of malicious traffic generated by malware samples.

IV. Evaluating the effectiveness of HTTP-based clustering for Android malware

This section represents the main contribution of the paper, as we aim to assess if the technique proposed for traditional desktop malware can be used to effectively cluster Android malware. In particular, our experiments had three main goals:

- verifying if the HTTP traffic generated by Android malware can be used to reveal the family they belong to;
- assessing which of the features that can be extracted from HTTP traffic are effective for malware clustering;
- checking if the result of the clustering process allows for extracting malware signatures that could be used by a NIDS.

In the following, we first describe the dataset used in the experiments and we present the measures used to evaluate the results. Then, we report and discuss all the experimental results that we attained.

A. Dataset

We gathered a large number of malicious Android applications to evaluate the effectiveness of the clustering procedure, and a large number of benign Android applications to evaluate the effectiveness of the signatures in terms of false positive rate. For the malicious samples, we focused on Android malware families that were related to botnets. We also analyzed malware families that delivered information to a remote server through HTTP.

The samples were gathered from Malgenome [39], Contagio [40], Drebin [17] and VirusTotal [41]. There were many malware families that interacted with their C&C servers by HTTP, SMS messages, emails, etc. As the majority of malware families use the HTTP protocol, we just considered the families that employ HTTP for their C&C channel. For each sample, we extracted the HTTP information by using CopperDroid [42], or by employing either Anubis [43] or TraceDroid [44] for the cases in which CopperDroid could not generate network traffic. To retain the samples that actually generated network traffic, we removed the ones that did not produce any valid HTTP communication due to the unavailability of their C&C servers.

To avoid inaccuracies in assigning each malware sample to a family, we developed a tool[1] that automatically scans each sample using Virustotal, and creates a naming convention based on the outputs of different anti-malware products. The list of the considered malware families is shown in Table II.

[1] https://github.com/ManSoSec/Auto-Malware-Labeling

Malware family	# Samples	Malware family	# Samples
AndroRat	11	Fjcon	106
AVPass	1	Geinimi	24
BackFlash	2	GoldenEagle	2
BadNews	2	Lien	6
BaseBridge	112	NickiSpy	2
BgServ	45	Obad	1
Chulli	2	Plankton	119
DroidKungFu	86	RootSmart	25
Extension	69	Skullkey	1
FakeAngry	151	SMSpacem	5
FakePlay	8	Tracer	13
FakeTimer	13		
		Total	**806**

Table II. MALWARE FAMILIES USED FOR THE EXPERIMENTAL ASSESSMENT OF THE EFFECTIVENESS OF THE HTTP CLUSTERING PROCEDURE

Benign Dataset	Number of requests
Web Browser	1102237
Android Apps	1037555
Total	**2139792**

Table III. NUMBER OF BENIGN REQUESTS GENERATED BY BROWSING WEB SITES, AND BY ANDROID APPLICATIONS

There are two columns in Table II. The Malware family column refers to the name of the malware variant, and the related number shows the considered malware samples for that variant.

In order to evaluate the false positive rate, we also collected a dataset of legitimate traffic. We collected over 2×10^6 HTTP requests from October 2014 to December 2014. Part of this traffic was achieved by sniffing the HTTP requests generated by crawling 173 most visited web sites (without considering search engines) in an Android emulator. The other part of the benign dataset contains HTTP traffic generated by Android applications. To generate such traffic, we performed two steps: (i) we collected applications that feature the permission *android.permission.INTERNET* by crawling[2] Google Play. We also obtained the thirty top free applications in Google Play under different categories such as Comics, Communication, News and Magazines, Shopping, Social, Sports, and Travel; (ii) we emulated the execution of the Android applications by simulating a real user behavior through the Android testing framework named Uiautomator [45]. Uiautomator lets users test the application user interface (UI) efficiently, and we exploited it to automatically interact with all of the elements in the applications' layouts to generate HTTP traffic. During the interaction, the network traffic is captured by the aid of the tcpdump tool, which collected several Gigabytes of benign HTTP traffic. As shown in Table III, both the web browser procedure and the collection of HTTP traffic from benign applications produced around one million requests.

We extracted the seven statistical features from HTTP requests and responses by Jnetpcap [46] Java library. Because the values of the features are in different ranges, we performed feature normalization by re-scaling feature values according to the following equation:

$$x_i^* = \frac{x_i - min(x_i)}{max(x_i) - min(x_i)} \quad (1)$$

[2]https://github.com/egirault/googleplay-api

where the min and max are computed across the whole dataset.

B. Evaluation of the proposed system

To evaluate the performances of the system, the following measures have been used:

(A) `Cohesion`:
The cohesion of a cluster C_i measures the average similarity between two samples in the cluster with a value between zero and one. It will be maximum (one) when the malware samples in a cluster belong to the same family, i.e., when they share the same label. The value zero, on the other hand, indicates that the malware samples are from different families. Whereas the HTTP traffic is similar according to the features employed, the labels of the samples are different. The average value has been computed only for those clusters that contain at least two malware samples. In fact, clusters containing just one malware sample have a cohesion equal to one by definition, and including them would mistakenly bias the final result.

(B) `Separation`:
The separation between two clusters C_i and C_j measures the average family label distance between malware belonging to C_i and malware belonging to C_j. This gives us an indication about whether the malware samples in two different clusters belong to different malware families or not. In order to understand how much the whole of clusters are well separated, instead of calculating the average of the separation between clusters, we calculate the percentage of the number of clusters that have a separation above a threshold. The detailed definitions of the measures of cohesion and separation can be found in [37].

(C) `Detection rate`:
The detection rate measures the percentage of malware that is detected by relying on the signatures extracted by the clusters. The detection rate is calculated for each value of the radius parameters that controls the number of generated clusters in the BIRCH algorithm. The extracted signatures are then included in Snort, [47] which is used to process network traces produced by malware samples and legitimate apps. Then, the alerts are collected and the detection rate is calculated as follows:

$$D.R.(\%) = \frac{N_{malware}}{N} * 100 \quad (2)$$

where $N_{malware}$ is the number of samples for which Snort generates at least one alert, and N is the total number of samples.

(D) `False positive rate`:
The false positive measures the percentages of false alerts that the IDS outputs according to the signatures extracted from the clusters, and it is computed according to the following formula:

$$F.P.(\%) = \frac{N_{alerts}}{N_{requests}} * 100 \quad (3)$$

where N_{alerts} is the sum of all alerts and $N_{requests}$ is the number of all the requests.

C. Experimental Results and Discussion

1) Cohesion vs. Separation: We carried out the experiments by running the complete two-step clustering algorithm. First, we run BIRCH on the statistical features, and we applied the single-linkage hierarchical algorithm to refine the clustering result of the first step. Then, we compared these results with the ones obtained by running only the BIRCH algorithm on the statistical features. The metrics used to compare the results are Cohesion and Separation. We computed these two metrics for 16 values of the R parameter, i.e., the *radius* that controls the number of clusters generated by BIRCH in the range from 10^{-7} to 10^{0}.

Surprisingly, we noticed that using the two-step algorithm leads to the same Cohesion and Separation values that are obtained when using BIRCH stand-alone. Figure 1 shows the values of the average Cohesion of all clusters for different values of radius R used in the experiments. Cohesion exhibits only small variations with different values of the radius. In particular, the average Cohesion is around 0.95 for values of the radius lower than 10^{-3}, and it slightly decreases in the interval between 10^{-3} and 5×10^{-1}. From 0.88, the average Cohesion increases again. Achieving high Cohesion means that the malware inside each cluster are very similar to each other and therefore they are properly clustered in the same family. However, we will see that this is not enough to produce reliable malware signatures. Figure 2 shows the values of the Separation index. The percentage of pairs of clusters with a separation index higher than 0.1 ranges from 84.01% to 96.66%. Attaining high Separation values means that the clusters are better separated. By examining the two figures, it turns out that the best trade-off between Cohesion and Separation is for values of the radius between 0.5 and 1.

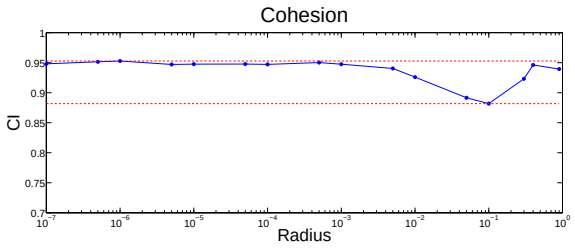

Figure 1. The average value of the cohesion indexes CI.

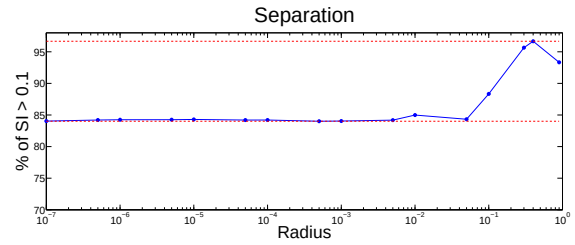

Figure 2. The percentage of pair of clusters with a separation index SI higher than 0.1.

2) Detection Rate vs. False Positive Rate: To compute the detection rate and the false positive rate, we extracted the malware signatures (which are a part of the request URL) from the clusters. In the experiments, we extracted different signatures for three different values of the minimum (Min) length of the signatures, i.e., 5, 10, and 15. Some examples are shown below :

Sig1: *content:"POST"; distance:0; nocase; content:"/ad"; distance:0; nocase;*
The above signature has length 3, which is not in none of Min 5, 10, and 15 categories.

Sig2: *content:"POST"; distance:0; nocase; content:"/aar.do"; distance:0; nocase;*
The above signature has length 7, which is in Min 5 category but not in 10 and 15 categories.

Sig3: *content:"POST"; distance:0; nocase; content:"/api/proxy"; distance:0; nocase;*
The above signature has length 10, which is in Min 5 and 10 categories but not in the Min 15 category.

Sig4: *content:"GET"; distance:0; nocase; content:"/adv/d?t=135782568"; distance:0; nocase;*
The above signature has length 18, which is in all of Min 5, 10, and 15 categories.

The detection rate calculated for the three different signature lengths and for the values of the radius reported in the previous subsection are shown in Figure 3. The values of the detection rate are quite high, and allow for precisely detecting the network traffic generated by malicious applications. The false positive rate accounts for the specificity of the signatures, and it is computed to evaluate the fraction of benign HTTP requests that match with the signatures. The results are shown in Figure 5. Both the detection and false positive rates do not improve if the two step clustering is employed instead of the stand-alone BIRCH (see Figure 4 and Figure 6). Although achieving lower Cohesion and higher number of rules in the range of $10^{-}3$ and 10^{0} could lead to producing more signatures with less integrity, it could significantly worsen the false positive rate.

The appropriate set of signatures are those that allow attaining a high detection rate and low false positives. To this end, we observe that the best values for the detection rate (very close to 100%) were obtained by signatures with minimum length equal to five for all values of the radius. However, if we take into account the corresponding values for the false positive rate, it turns out that the best trade-off is reached when the radius = 0.9, which allows to attain a false positive rate of around 7%.

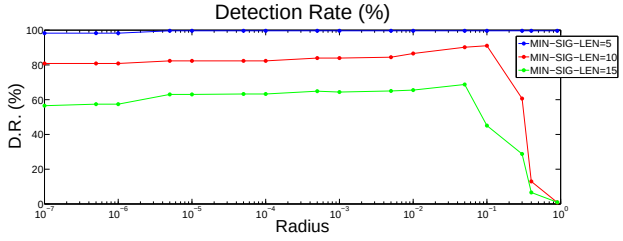

Figure 3. The percentage of detection rate D.R. (%) obtained with different set of signatures and different values of the radius, by only doing coarse-grained clustering.

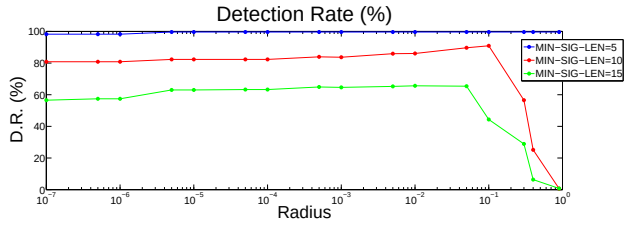

Figure 4. The percentage of detection rate D.R. (%) obtained with different set of signatures and different values of the radius, by doing fine-grained clustering in addition to coarse-grained clustering.

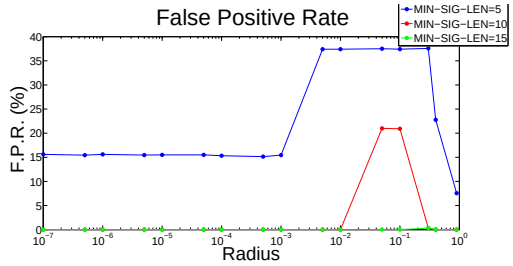

Figure 5. The percentage of false positive rate F.P.R. (%) obtained with different set of signatures and different values of the radius, by only doing coarse-grained clustering.

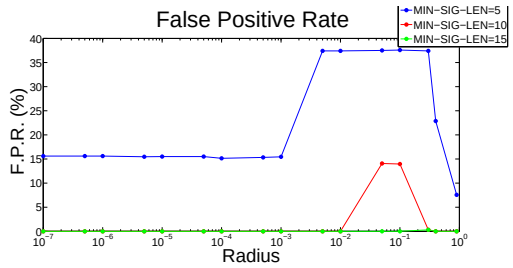

Figure 6. The percentage of false positive rate F.P.R. (%) obtained with different set of signatures and different values of the radius, by doing fine-grained clustering in addition to coarse-grained clustering.

3) Number of Clusters and Number of Signatures: Although the efficiency of the detection system is important, the number of clusters (Figure 7) and signatures (Figure 8) need to be evaluated. The number of clusters is correlated to the value of the radius. If the radius is large, the number of clusters is small, meaning that each cluster is likely to contain malware from different families. In our experiments, the number of clusters for a radius value = 10^{-7} is 501, whereas the number of clusters is 5 for radius = 0.9. As we mentioned, the Snort rules are based on the signatures generated from the clusters, and they are correlated with the number of signatures. A large number of clusters may translate into a redundant number of signatures, as malware belonging to the same family may be grouped in different clusters. Consequently, even if the detection rate does not vary, the false positive rate and the IDS speed exhibit lower performances.

Platform	Get	Post	Url	Param	Sent	Response
Windows	19.1(183.1)	1.1(10.3)	51.9(58.0)	1.4(3.3)	84.5(201.5)	81E+4(88E+5)
Android	3.8(6.1)	2.8(3.3)	87.9(98)	4.4(6.5)	73.7(97.0)	88E+2(15E+4)

Table IV. COMPARISON ON AVERAGE (STANDARD DEVIATION) FOR EACH STATISTICAL FEATURE

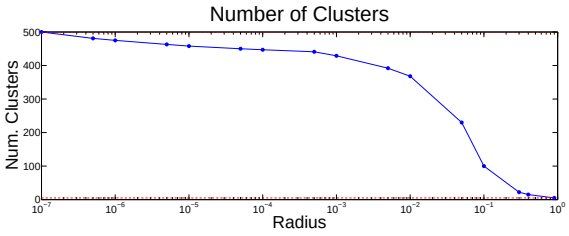

Figure 7. The total number of clusters (Num. clusters) for different values of the radius.

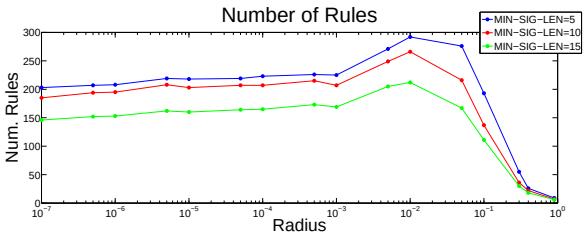

Figure 8. The total number of the Snort rules (Num. rules) obtained from the signatures for different radius values.

D. Comparisons with HTTP based clustering for traditional desktop malware

As Android has been largely adopted only recently compared to traditional desktop systems, the number of available Android malware samples that generate malicious HTTP traffic are fewer than the analogous desktop malware samples. In general, compared to the work on traditional desktop malware [9], clustering Android malware samples by HTTP traffic traces shows that: (i) the value of cohesion is higher, (ii) the value of separation is lower, (iii) the detection rate is higher, (iv) and the false positive rate is lower. Thus, grouping malware samples according to the generated HTTP traffic they produce is effective not only to detect malware families, but also to produce effective malware signatures. In order to show the basic motivation behind this behavior, we computed the mean and the variance of each statistical feature for malware samples belonging to both platforms. We show this comparison in Table IV.

Windows malware exhibit a higher mean value on features such as the number of Get requests, and the length of sent and response data. Conversely, the number of Post requests and parameters, and the length of URL are larger for the Android malware. The limited capability of mobile devices compared to desktop PCs can be the reason for the low interaction of mobile malware with their C&C servers. In other words, desktop applications tend to produce multiple HTTP requests to perform an action, and Android apps tend to produce one long HTTP request containing all the information. Another possibility is that mobile botnets have to control less

functionalities and applications compared to Desktop ones. For example, the higher value of the average Get requests for Desktop malware shows that such malware resort to more computational resources to generate as many requests as possible. Consequently, they do not need to create long URLs with different parameters. Conversely, as mobile devices have limited computational capabilities, malicious applications are developed to keep the number of requests as small as possible. Thus, they are forced to enrich their requests by using a larger number of parameters. In addition, the analysis of the standard deviation values allows for a better understanding of the diversity of requests. As an example, the diversity of the number of requests and of the amount of transferred data is much higher in Desktop malware compared to Android malware. This examples provide reasons for the effectiveness of the clustering by just using the statistical traffic features, and for the higher performances in malware detection for mobile devices compared to desktop systems.

V. Conclusions and Future Work

In this paper, we performed an analysis of Android botnets that employ HTTP traffic for their communications. By clustering the generated network traffic of different Android malware with the usage of an algorithm originally developed for grouping desktop malware, we showed that the samples belonging to the same malware family have similar HTTP traffic statistics. Moreover, a small number of signatures can be extracted from the clusters, allowing to achieve a good trade-off between the detection rate and the false positive rate. The causes of this behavior can be related to the higher uniformity of the HTTP traffic generated by Android botnet malware compared to traffic generated by Desktop botnet malware.

Acknowledgements

This work has been partly supported by the project "Computational quantum structures at the service of pattern recognition: modeling uncertainty" [CRP-59872] funded by Regione Autonoma della Sardegna, L.R. 7/2007, Bando 2012. Davide Maiorca gratefully acknowledges Sardinia Regional Government for the financial support of his PhD scholarship (P.O.R. Sardegna F.S.E. Operational Programme of the Autonomous Region of Sardinia, European Social Fund 2007-2013 - Axis IV Human Resources, Objective l.3, Line of Activity l.3.1.).

References

[1] A. Bose and K. G. Shin, "Proactive security for mobile messaging networks," in *Proceedings of the 5th ACM Workshop on Wireless Security*, ser. WiSe '06. New York, NY, USA: ACM, 2006, pp. 95–104. [Online]. Available: http://doi.acm.org/10.1145/1161289.1161307

[2] R. Racic, D. Ma, and H. Chen, "Exploiting mms vulnerabilities to stealthily exhaust mobile phone's battery," in *Securecomm and Workshops, 2006*, Aug 2006, pp. 1–10.

[3] J. W. Mickens and B. D. Noble, "Modeling epidemic spreading in mobile environments," in *Proceedings of the 4th ACM Workshop on Wireless Security*, ser. WiSe '05. New York, NY, USA: ACM, 2005, pp. 77–86. [Online]. Available: http://doi.acm.org/10.1145/1080793.1080806

[4] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection," in *Proceedings of the 17th Conference on Security Symposium*, ser. SS'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 139–154. [Online]. Available: http://dl.acm.org/citation.cfm?id=1496711.1496721

[5] Sophos, "Security threat report," 2014. [Online]. Available: http://www.sophos.com/en-us/medialibrary/PDFs/other/sophos-security-threat-report-2014.pdf

[6] R. Nigam, "A timeline of mobile botnets," *Virus Bulletin*, March 2015. [Online]. Available: https://www.virusbtn.com/virusbulletin/archive/2015/03/vb201503-mobile-botnets

[7] S. Dai, A. Tongaonkar, X. Wang, A. Nucci, and D. Song, "Networkprofiler: Towards automatic fingerprinting of android apps," in *INFOCOM, 2013 Proceedings IEEE*, April 2013, pp. 809–817.

[8] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos, "Profiledroid: Multi-layer profiling of android applications," in *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking*, ser. Mobicom '12. New York, NY, USA: ACM, 2012, pp. 137–148. [Online]. Available: http://doi.acm.org/10.1145/2348543.2348563

[9] R. Perdisci, D. Ariu, and G. Giacinto, "Scalable fine-grained behavioral clustering of http-based malware," *Computer Networks*, vol. 57, no. 2, pp. 487–500, 2013. [Online]. Available: http://dx.doi.org/10.1016/j.comnet.2012.06.022

[10] L. Wu, M. Grace, Y. Zhou, C. Wu, and X. Jiang, "The impact of vendor customizations on android security," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ser. CCS '13. New York, NY, USA: ACM, 2013, pp. 623–634. [Online]. Available: http://doi.acm.org/10.1145/2508859.2516728

[11] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner, "Analyzing inter-application communication in android," in *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '11. New York, NY, USA: ACM, 2011, pp. 239–252. [Online]. Available: http://doi.acm.org/10.1145/1999995.2000018

[12] D. Octeau, P. McDaniel, S. Jha, A. Bartel, E. Bodden, J. Klein, and Y. L. Traon, "Effective inter-component communication mapping in android: An essential step towards holistic security analysis," in *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)*. Washington, D.C.: USENIX, 2013, pp. 543–558. [Online]. Available: https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/octeau

[13] M. C. Grace, Y. Zhou, Z. Wang, and X. Jiang, "Systematic detection of capability leaks in stock android smartphones," in *19th Annual Network and Distributed System Security Symposium, NDSS*. San Diego, California, USA: The Internet Society, February 2012.

[14] L. Lu, Z. Li, Z. Wu, W. Lee, and G. Jiang, "Chex: Statically vetting android apps for component hijacking vulnerabilities," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12. New York, NY, USA: ACM, 2012, pp. 229–240. [Online]. Available: http://doi.acm.org/10.1145/2382196.2382223

[15] W. Zhou, Y. Zhou, M. Grace, X. Jiang, and S. Zou, "Fast, scalable detection of "piggybacked" mobile applications," in *Proceedings of the Third ACM Conference on Data and Application Security and Privacy*, ser. CODASPY '13. New York, NY, USA: ACM, 2013, pp. 185–196. [Online]. Available: http://doi.acm.org/10.1145/2435349.2435377

[16] W. Zhou, Y. Zhou, X. Jiang, and P. Ning, "Detecting repackaged smartphone applications in third-party android marketplaces," in *Proceedings of the Second ACM Conference on Data and Application Security and Privacy*, ser. CODASPY '12. New York, NY, USA: ACM, 2012, pp. 317–326. [Online]. Available: http://doi.acm.org/10.1145/2133601.2133640

[17] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, K. Rieck, and Siemens, "Drebin: Efficient and explainable detection of android malware in your pocket," in *Proceedings of the ISOC Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, 2014.

[18] D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.-P. Wu, "Droidmat: Android malware detection through manifest and api calls tracing," in *Information Security (Asia JCIS), 2012 Seventh Asia Joint Conference on*, Aug 2012, pp. 62–69.

[19] C. Yang, Z. Xu, G. Gu, V. Yegneswaran, and P. Porras, "Droidminer: Automated mining and characterization of fine-grained malicious behaviors in android applications," in *19th European Symposium on Research in Computer Security (ESORICS'14)*, ser. Lecture Notes in Computer Science. Wroclaw, Poland: Springer Berlin Heidelberg, 2014.

[20] I. Burguera, U. Zurutuza, and 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*, ser. SPSM '11. New York, NY, USA: ACM, 2011, pp. 15–26. [Online]. Available: http://doi.acm.org/10.1145/2046614.2046619

[21] A. Reina, A. Fattori, and L. Cavallaro, "A system call-centric analysis and stimulation technique to automatically reconstruct android malware behaviors," in *Proceedings of the 6th European Workshop on System Security (EUROSEC)*, Prague, Czech Republic, April 2013.

[22] L. K. Yan and H. Yin, "Droidscope: Seamlessly reconstructing the os and dalvik semantic views for dynamic android malware analysis," in *Proceedings of the 21st USENIX Conference on Security Symposium*, ser. Security'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 29–29. [Online]. Available: http://dl.acm.org/citation.cfm?id=2362793.2362822

[23] V. Rastogi, Y. Chen, and W. Enck, "Appsplayground: Automatic security analysis of smartphone applications," in *Proceedings of the Third ACM Conference on Data and Application Security and Privacy*, ser. CODASPY '13. New York, NY, USA: ACM, 2013, pp. 209–220. [Online]. Available: http://doi.acm.org/10.1145/2435349.2435379

[24] B. Amos, H. Turner, and J. White, "Applying machine learning classifiers to dynamic android malware detection at scale," in *Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International*, July 2013, pp. 1666–1671.

[25] F. Li, N. Clarke, M. Papadaki, and P. Dowland, "Behaviour profiling on mobile devices," in *Emerging Security Technologies (EST), 2010 International Conference on*, Sept 2010, pp. 77–82.

[26] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–6. [Online]. Available: http://dl.acm.org/citation.cfm?id=1924943.1924971

[27] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall, "These aren't the droids you're looking for: Retrofitting android to protect data from imperious applications," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, ser. CCS '11. New York, NY, USA: ACM, 2011, pp. 639–652. [Online]. Available: http://doi.acm.org/10.1145/2046707.2046780

[28] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, ""andromaly": A behavioral malware detection framework for android devices," *J. Intell. Inf. Syst.*, vol. 38, no. 1, pp. 161–190, Feb. 2012. [Online]. Available: http://dx.doi.org/10.1007/s10844-010-0148-x

[29] A. Shabtai, L. Tenenboim-Chekina, D. Mimran, L. Rokach, B. Shapira, and Y. Elovici, "Mobile malware detection through analysis of deviations in application network behavior," *Computers & Security*, vol. 43, no. 0, pp. 1 – 18, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167404814000285

[30] F. Narudin, A. Feizollah, N. Anuar, and A. Gani, "Evaluation of machine learning classifiers for mobile malware detection," *Soft Computing*, pp. 1–15, 2014. [Online]. Available: http://dx.doi.org/10.1007/s00500-014-1511-6

[31] A. Arora, S. Garg, and S. Peddoju, "Malware detection using network traffic analysis in android based mobile devices," in *Next Generation Mobile Apps, Services and Technologies (NGMAST), 2014 Eighth International Conference on*, Sept 2014, pp. 66–71.

[32] A. Tongaonkar, S. Dai, A. Nucci, and D. Song, "Understanding mobile app usage patterns using in-app advertisements," in *Passive and Active Measurement*, ser. Lecture Notes in Computer Science, M. Roughan and R. Chang, Eds. Springer Berlin Heidelberg, 2013, vol. 7799, pp. 63–72.

[33] A. Zarras, A. Papadogiannakis, R. Gawlik, and T. Holz, "Automated generation of models for fast and precise detection of http-based malware," in *2014 Twelfth Annual International Conference on Privacy, Security and Trust (PST)*, July 2014, pp. 249–256.

[34] M. Conti, L. V. Mancini, R. Spolaor, and N. V. Verde, "Can't you hear me knocking: Identification of user actions on android apps via traffic analysis," *Fifth ACM Conference on Data and Application Security and Privacy (CODASPY)*, 2015.

[35] X. Wu, D. Zhang, X. Su, and W. Li, "Detect repackaged android application based on http traffic similarity," *Security and Communication Networks*, pp. n/a–n/a, 2015. [Online]. Available: http://dx.doi.org/10.1002/sec.1170

[36] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: an efficient data clustering method for very large databases," in *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996.*, 1996, pp. 103–114. [Online]. Available: http://doi.acm.org/10.1145/233269.233324

[37] R. Perdisci, W. Lee, and N. Feamster, "Behavioral clustering of http-based malware and signature generation using malicious network traces," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 26–26. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855711.1855737

[38] J. Newsome, B. Karp, and D. X. Song, "Polygraph: Automatically generating signatures for polymorphic worms," in *2005 IEEE Symposium on Security and Privacy (S&P 2005), 8-11 May 2005, Oakland, CA, USA*, 2005, pp. 226–241. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/SP.2005.15

[39] X. J. Y. Zhou, "Android malware genome project," http://www.malgenomeproject.org, 2012.

[40] M. Parkour, "Contagio mobile - mobile malware mini dump," http:///contagiominidump.blogspot.it/, 2012.

[41] "Virustotal - free online virus, malware and url scanner," https://www.virustotal.com/, 2014.

[42] "Copperdroid," http://copperdroid.isg.rhul.ac.uk/copperdroid/, 2014.

[43] "Anubis - free online malware analysis for unknown binaries (windows executable or android apk)," https://anubis.iseclab.org, 2014.

[44] "Tracedroid - free online dynamic android app nalysis," http://tracedroid.few.vu.nl, 2014.

[45] "Uiautomator - android framework," http://developer.android.com/tools/help/uiautomator/index.html, 2014.

[46] "jnetpcap opensource — protocol analysis sdk," http://jnetpcap.com/, 2014.

[47] "Snort - open source network intrusion prevention system," https://www.snort.org, 2014.