Anomaly-based Exploratory Analysis and Detection of Exploits in Android Mediaserver

Guillermo Suarez-Tangil[1 3], Santanu Kumar Dash[1],
Pedro García-Teodoro[2], José Camacho[2], Lorenzo Cavallaro[3]
[1]University College Lonson (UK), [2]University of Granada (Spain)
[3]Royal Holloway University of London (UK)

ABSTRACT

Smartphone platforms are becoming increasingly complex, which gives way to software vulnerabilities difficult to identify and that might allow malware developers to gain unauthorized privileges through technical exploitation. However, we maintain that these type of attacks indirectly renders a number of unexpected behaviors in the system that can be profiled. In this work we present *CoME*, an anomaly-based methodology aiming at detecting software exploitation in Android systems. *CoME* models the normal behavior of a given software component or service and it is capable of identifying any unanticipated behavior. To this end, we first monitor the normal operation of a given exploitable component through lightweight virtual introspection. Then, we use a multivariate analysis approach to estimate the normality model and detect anomalies. We evaluate our system against one of the most critical vulnerable and widely exploited services in Android, *i.e.*, the `mediaserver`. Results show that our approach can not only provide a meaningful explanatory of discriminant features for illegitimate activities, but can also be used to accurately detect malicious software exploitations at runtime.

1. INTRODUCTION

Mobile devices like smartphones and tablets are becoming more and more accepted platforms among users worldwide. The number of tablets and smartphones is expected to be about one order of magnitude higher than the number of PCs and laptops in the next years [1, 2]. The increasing popularity of tablets and smartphones has led to an exponential growth of the number of risks and vulnerabilities in them [3, 4, 5, 6]. Currently, Android is reported as the mobile operating system (OS) most widely affected by these threats with around 97% of the present-day malware designed targeting Android systems.

The primary line of defense in Android is the security architecture of the device where an app runs in an isolated environment with a permission system restricting apps privileges. However, this isolation does not prevent apps from exploiting system or kernel vulnerabilities to bypass platforms protection. This is evidenced by a large number of critical vulnerabilities reported over the last years[1] [7]. Although some works and proposals have been developed in the literature to fight against vulnerabilities and exploits, the topic still remains highly challenging [8, 9].

As malware becomes sophisticated, it makes use of evasive techniques to hide its actions. A well established approach to evading detection mechanisms is to make actions contingent on a rare context. For example, malware which rely on time bombs starts actions only at a specific point in time, while malware that rely on logic bombs remain dormant until the condition guarding the malicious action comes true [10]. In such a setting it is infeasible to eke out the malicious action from the app in a controlled setting. Thus, the best option is to resort to an online monitor that keeps continuously checking for anomalous actions that are indicative of malice [11]. System calls are a widely used feature for this [12, 13]. Anomaly detection systems are highly interesting in this context, specially to fight against zero-day attacks and vulnerability exploitation [14].

The difficulty in designing any anomaly detection system is coming up with a model of normality. This becomes even more challenging while considering systems when there are multitude of actions such as system calls, file I/O actions, network transactions, etc. that may happen in a very short interval. Consequently, the best way to perform anomaly detection with little to no domain knowledge is to monitor all possible events of interests and let the anomaly detector figure out relevance of those events. Unfortunately, as has been observed in [15], this does not scale as the number of features is increased. Therefore, there is a need for an anomaly detection system that scales with the number of features introduced while still being able to pick up the most relevant events of interest corresponding to an anomalous behavior. Additionally, most anomaly detection techniques yield black box models with an uninterpretable linkage between input and output data.

In this paper, we introduce a novel anomaly-based approach aimed at detecting malicious activities intended to cause some harm to the target system. We advocate the use of Multivariate Statistical Network Monitoring (MSNM) to address aforementioned problems with anomaly detection. MSNM was originally proposed by Camacho *et al.* in [15] for anomaly detection in network environments. In this work, we apply it to detecting anomalous actions on Android systems that are indicative of malice. To this end, we show how the proposed anomaly detector effortlessly extracts events and features of interest by using the well-known Principal Component Analysis (PCA) as a building

[1]www.cvedetails.com/product/19997/Google-Android.html

block. We also show that MSNM not only provides a means to easily process a multitude of features but also provides an easily interpretable model of normality and anomaly. When using MSNM, we do not need to perform a feature selection on first place, avoiding the risk of discarding useful information for anomaly detection. In particular, we focus our attention on Android platforms, our main contributions being as follows:

- The proposed anomaly-based methodology relies on the estimation of a *normality model* for a given Android service. For that, the normal expected operation of a given service is first estimated by collecting system information at multiple levels of granularity. CopperDroid [16] will be used as the monitoring system to collect such information.

- The multivariate statistical approach proposed in [15] (MSNM) supports the overall anomaly detection process. Despite the inherent capabilities of such techniques to handle a number of features of diverse nature and origin, they are not commonly used for malware detection for the time being, which constitutes a novelty in the field.

- A relevant and well publicised vulnerability in the Android libraries is used to test our proposal: the *Stagefright* bug[2]. For that, we first estimate the normal operating conditions (NOC) of the system's `mediaserver`—the sub-system responsible for processing media file in Android. Then, we feed the `mediaserver` with both legitimate and crafted files aimed at exposing the vulnerability. By analyzing the associated behavior in all the cases, we shall show the capacity of MSNM in detecting the malicious ones.

The rest of the paper is organised as follows. Section 2 introduces our anomaly detection system, named *CoME* and designed to identify behaviors that are anomalous to normal operating conditions. Section 3 is afterwards devoted to evaluate our proposal. For that, in this section, Android `mediaserver` is first described as our case study, and then the detection experimental results obtained are provided and properly discussed. Section 4 presents some principal works in the field of malware detection in Android, with especial emphasis in vulnerabilities exploitation. Finally, Section 5 summarizes the contributions of the work.

2. CoME: ANOMALY DETECTION IN ANDROID

In this section, we present a novel proposal aimed at detecting anomalies in Android platforms. For that, a usual methodology is followed [17] (Fig. 1):

- *Monitoring*: The target system is monitored in order to collect information regarding the overall activity taking place on it. This information is parameterized to represent (usually in terms of a feature vector) the 'state' or 'behavior' of the system at a given instant. This way, a sequence of observations are disposed as the system operates and evolves over time.

- *Training*: Provided a set of observations corresponding to the 'normal/legitimate' operation of the system, a 'normality' model is first estimated by considering some mathematical theory (*e.g.*, Markov models, fuzzy theory, neural networks, etc.).

- *Detection*: Further observations gathered from the monitored system are subsequently evaluated by using the 'normality' model in order to estimate a deviation score. From that, we conclude that an anomaly is occurring if the deviation score obtained surpasses a given threshold. Otherwise, the observation (or sequence or observations) analyzed is classified as 'normal'.

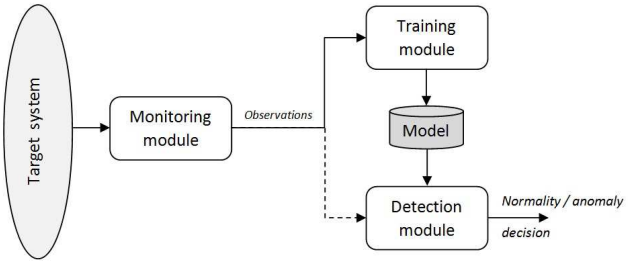


Figure 1: Functional architecture for anomaly detection.

Our anomaly detection proposal is named *CoME*, as it is based on the combined use of two well-known tools: CopperDroid and MEDA/MSNM. The first one is used as a monitoring tool for Android devices [16]. The second one, MSNM, is part of the functionality programmed in the Multivariate Exploratory Data Analysis (MEDA) toolbox [18] as a detection approach proposed and successfully applied to detect anomalies in network environments. In its current version, *CoME* is an offline tool where each of the two component modules (CopperDroid for monitoring; and MEDA/MSNM, for data analysis) is independent of the other.

In what follows, CopperDroid and MSNM are briefly described.

2.1 Android Monitoring with CopperDroid

CoME makes use of CopperDroid [16] as its dynamic analysis component, which runs the Android OS in a sandbox, records system calls and their arguments, and reconstruct high-level behaviors. These reconstructed behaviors have already been shown to be effective in malware classification as they can point out which Android services and state-modifying actions were carried out by the malware [19]. The classes of high-level behaviors include network accesses, file accesses, Binder methods, and file execution as described in Table 1. CopperDroid provides full access to the arguments of all transactions going through the Binder mechanism for inter-process and inter-component communication [20].

[2]http://www.androidcentral.com/stagefright

```
1  setpgid(0, getpid());
2
3  /*set gid*/
4  if (svc->gid) {
5      if (setgid(svc->gid) != 0) {
6          ERROR("setgid failed: %s\n",
7                  strerror(errno));
8          _exit(127);
9      }
10 }
11
12 /*set supplemental gid*/
13
14 if (svc->nr_supp_gids) {
15     if (setgroups(svc->nr_supp_gids,
16                   svc->supp_gids) != 0) {
17         ERROR("setgroups failed: %s\n",
18                 strerror(errno));
19         _exit(127);
20     }
21 }
```

(a) Code from service initialisation

```
1  {
2      "syscall": [ {
3          "parameters": [],
4          "sysname": "getpid" } ]
5  },
6  {
7      "syscall": [ {
8          "parameters": [0, 1076],
9          "sysname": "setpgid" } ]
10 },
11 {
12     "syscall": [ {
13         "parameters": [1],
14         "sysname": "setgid32" } ]
15 },
16 {
17     "syscall": [ {
18         "parameters": [7, "'\\xee\\x03\\x00\\x00'"]
19         "sysname": "setgroups32" } ]
20 }
```

(b) Behavior observed by CopperDroid

Figure 2: Sample trace logged by CopperDroid. Figure 2a shows code from the `init` process which starts services and assigns permissions to the spawned process in Android, while Figure 2b shows the corresponding low-level system calls captured by CopperDroid.

Feature Set	Contained Details
Network Access	IP, port, and network traffic size
File Access	File name/type, name classes
Binder Methods	Method name and parameters
Execute File	File name/type, user permissions
Systemcalls	Low-level syste calls

Table 1: Behavioral classes and details extracted by CopperDroid. A subset
of this information is used as features by MSNM.

CopperDroid relies on Virtual Machine Introspection (VMI) [21] to perform out-of-the-box behavioral analysis. The introspection is lightweight, however, and designed to be directly applicable to all Android OS versions without requiring any modifications to the Android system. A sample trace captured by CopperDroid is shown in Figure 2. The code snippet shown in Figure 2a is an excerpt from the `init` process. In Figure 2a, the `init` process is shown to be setting the process group identifier (ll. 1), effective group identifier of the calling process (ll. 5) and the supplementary group identifiers for the calling process (ll. 15). The corresponding system calls observed for these actions can be seen in Figure 2b as *setpgid* (ll. 9), *setgid32* (ll. 14) and *setgroups32* (ll. 19), respectively.

2.2 MSNM Detection Methodology

The MSNM acronym stands for Multivariate Statistical Network Monitoring, and is related to a novel PCA-based multivariate detection approach proposed by Camacho *et al.* in [15] for anomaly detection in network environments. MSNM is supported on the MEDA toolbox [18] and is a reformulation of the approach by Lakhina et *al.* in [22] following state-of-the-art procedures in PCA-based Multivariate Statistical Process Control (MSPC) [23] in the process industry.

In what follows, the basics of MSNM methodology are introduced.

2.2.1 *Principal Component Analysis*

PCA is intended to linearly transform a given original set of M variables into a new and reduced set of A uncorrelated variables, the so-called *principal components* or PCs. This way, if $\mathbf{X}$ is a data matrix with M variables associated with a given phenomenon and J observations of each variable, PCA reduces its dimension from M variables to A PCs by finding the A-dimensional latent subspace of the most variability. The PCs are the eigenvectors of $\mathbf{X}^T \cdot \mathbf{X}$ for mean centered $\mathbf{X}$ and sometimes after some form of scaling.

PCA follows the next equation:

$$\mathbf{X} = \mathbf{T}_A \cdot \mathbf{P}_A^T + \mathbf{E}_A \tag{1}$$

where $\mathbf{T}_A$ is the $J \times A$ *score* matrix, $\mathbf{P}_A$ is the $M \times A$ *loading* matrix, and $\mathbf{E}_A$ is the $J \times M$ *residual* matrix. The maximum variance directions are obtained from the eigenvectors of $\mathbf{X}^T \cdot \mathbf{X}$, and they are ordered as the columns of $\mathbf{P}_A$ by explained variance. The rows of $\mathbf{T}_A$ are the projections of the original J observations in the new latent subspace. $\mathbf{E}_A$ is the matrix that contains the residual error, which plays a crucial role in anomaly detection, as shown afterwards. The projection (score) on the PCA subspace of a new observation is obtained as follows:

$$\mathbf{t}_{new} = \mathbf{x}_{new} \cdot \mathbf{P}_A \tag{2}$$

where $\mathbf{x}_{new}$ is a $1 \times M$ vector representing a new object and $\mathbf{t}_{new}$ is a $1 \times A$ vector representing its projection to the latent subspace, while

$$\mathbf{e}_{new} = \mathbf{x}_{new} - \mathbf{t}_{new} \cdot \mathbf{P}_A^T \quad (3)$$

corresponds to the associated residuals.

2.2.2 Detecting Anomalies with MSNM

In PCA-based MSPC, it is usual to monitor a pair of statistics, the D-statistic and the Q-statistic, in a pair of monitoring charts. On the one hand, the D-statistic or Hotelling's T2 statistic, is computed from the projections (scores) of the original observations. On the other hand, the Q-statistic or SPE, represents the compression of the residuals (the quantity remaining from the projections). Although it is widely accepted that the SPE provides of higher detection capability than the D-statistic, both statistics are complementary. The D-statistic and the Q-statistic for observation n can be computed from the following equations:

$$D_n = \sum_{a=1}^{A} \left(\frac{t_{an} - \mu_{t_a}}{\sigma_{t_a}} \right) \quad (4)$$

$$Q_n = \sum_{m=1}^{M} e_{nm}^2 \quad (5)$$

where t_{an} represents the score of the observation in the a-th component, μ_{t_a} and σ_{t_a} stand for the mean and the standard deviation of the scores of that component in the calibration data, respectively, and e_{nm} represents the residual value corresponding to the m-th variable.

The analysis is performed in two successive stages or phases:

1. Calibration. According to the Statistical Process Control theory, calibration data needs to be inspected so that only common causes of variation remain. Typically, this is carried out by removing outliers. After outliers removal, available data are used for the calibration of the system, *i.e.*, matrices $\mathbf{T}_A$, $\mathbf{P}_A$ and $\mathbf{E}_A$ are obtained for 'normal' traffic. From these, the D and Q statistics for normal traffic can be computed, and upper control limits (UCLs) at a certain confidence level α can be established for the monitoring charts:

$$UCL_{D_\alpha} = \frac{A \cdot (N^2 - 1)}{N \cdot (N - A)} \, F_{(A,(N-A)),\alpha} \quad (6)$$

$$UCL_{Q_\alpha} = \theta_1 \cdot \left[\frac{z_\alpha \sqrt{2\theta_2 h_0^2}}{\theta_1} + 1 + \frac{\theta_2 h_0 (h_0 - 1)}{\theta_1^2} \right]^{\frac{1}{h_0}} \quad (7)$$

where $F_{(A,(N-A)),\alpha}$ is the F distribution with A and $N-A$ degrees of freedom, $\theta_n = \sum_{a=A+1}^{rank(\mathbf{X})} (\lambda_a)^n$, $rank(\mathbf{X})$ is the rank of the matrix of data $\mathbf{X}$ and λ_a the eigenvalues of the matrix $\frac{1}{N-1} \cdot \mathbf{E}_A^T \cdot \mathbf{E}_A$, $h_0 = 1 - \frac{2\theta_1\theta_3}{3\theta_2^2}$, and z_α is the $100 \cdot (1-\alpha)$% standardized normal percentile.

When enough calibration data is available, it is usual to replace eqs. (6) and (7) with the UCLs computed to be above $100 \cdot (1-\alpha)$% of the calibration observations. Thus, UCLs are set to values so that only $100 \cdot \alpha$% of the calibration observations surpass the associated values. A typical choice is $\alpha = 0.01$, so that only 1% of the calibration observations surpass the UCL.

2. Detection. Afterwards, subsequent new data are monitored and analyzed using these control limits. Thus, anomalies are detected when the limits are significantly or consistently exceeded. That is, when either a few observations surpass significantly the limits or a set of consecutive observations surpass them.

2.2.3 Diagnosing anomalies

Additionally to the detection, the contribution of the variables (or information gain) can also be abstracted to understand the root-causes of an anomaly. This can be done with oMEDA plots [24].

The oMEDA algorithm was designed to identify the variables related to specific artefacts, like clusters, trends or outliers, found in the projection of the data in a given subspace. This subspace could be determined by PCA or other related methods. In particular, anomalies in the D-st or Q-st are outliers in the PCA subspace and in the residual subspace, respectively.

The oMEDA technique is applied with a dummy variable $\mathbf{d}$ designed to cover the observations of interest. Let us assume the following example: a number of subsets of observations form different clusters of anomalies $\{C_1, \cdots, C_N\}$. If we are interested in identifying, for instance, the variables related to the deviation of C_i from C_j without considering the rest of clusters, a dummy variable $\mathbf{d}$ is created so that observations in C_i are set to 1, observations in C_j are set to -1, while the remaining observations are left to 0. Similarly, C_i can be compared to the center of coordinates (the data average) by setting $\mathbf{d}$ so that observations in C_i are set to 1 and the remaining to 0. Doing this with a cluster of one single observation is similar to issuing a contribution plot of that observation. Finally, values other than 1 and -1 can be included in the dummy variable if desired, which is useful for diagnosing trends in the scores. oMEDA is then performed using this dummy variable as follows:

$$\mathbf{d}_{A,(i)}^2 = 2 \cdot (\mathbf{x}_{(i)}^t - \mathbf{x}_{A,(i)}^t) \cdot \mathbf{D} \cdot \mathbf{x}_{A,(i)} \quad (8)$$

and

$$\mathbf{D} = \frac{\mathbf{d} \cdot (\mathbf{d})^t}{\|\mathbf{d}\|^2}. \quad (9)$$

where $\mathbf{x}_{(i)}$ and $\mathbf{x}_{A,(i)}$ contain the elements for variable i in $\mathbf{X}$ and its projection $\mathbf{X}_A$ in the PCA subspace. If oMEDA is used to compute the contribution of a single observation $\mathbf{x}$, then $\mathbf{x}_{(i)}$ and $\mathbf{x}_{A,(i)}$ are scalars.

The strength of MSNM is that it provides of a procedure to detect and diagnose anomalies in data sets with almost unlimited numbers of features. This comes from: *i)* the ability of PCA to combine and compress a large set

of variables into a reduced number of principal components; *ii)* the sensitivity of D-statistic and Q-statistic to identify anomalies in both model and residuals, *i.e.* to detect changes of behavior in comparison to the calibration data; and *iii)* the capability of oMEDA to point out the features in which the anomalous behavior is apparent. The outstanding capability of the MSNM approach to scale to very large numbers of features is the result of PCA being the simplest multivariate model, in the sense that it considers a linear relationship among the features.

3. EXPERIMENTAL EVALUATION

In this section, we first describe the case study used to evaluate *CoME*. Then, we present the experimental results and the use of our anomaly detector. A discussion is finally performed.

3.1 Case Study: Media Server

The Android `mediaserver` is one of the most critical and widely exploited services in Android. Although in this work we focus on this service, we emphasize that our framework is designed to monitor and analyze any given service or app running on the device.

3.1.1 Monitoring Important Features

The `mediaserver` in Android is typically invoked from the system media player or any app that uses the system media player.

Android provides a native-level framework called *Stagefright* for playing media files. *Stagefright* comes packaged with popular media codecs necessary to play audio and video files. It achieves this through two modules known as `audioflinger` and `surfaceflinger`. An app which uses the media player typically invokes the `mediaserver` through Android's bespoke inter-process communication scheme: the Binder protocol. Our dynamic analysis platform CopperDroid offers the unique advantage of logging these binder transactions, which enables deeper visibility into how the `mediaserver` is invoked.

For our experiments on anomaly detection, we invoke the `mediaserver` with both benign and crafted media files. For opening the files and playing the media, we used Android's stock media player. If the file is a crafted one, the `mediaserver` is known to crash and is restarted by the `init` process. A restarted `mediaserver` points towards a crafted or corrupt media file opened by the `mediaserver`. While opening a corrupt file does not necessarily amount to malicious activity, our anomaly detector errs on the side of caution and flags these as potentially dangerous.

The stock player is identified by the `com.android.music` process and is used to communicate with the `mediaserver` process, which further invokes the `surfaceflinger` process which is responsible for low-level actions to display or play media content. For monitoring events that point to anomalous behaviour, we shortlist three processes directly responsible for media playback: `mediaserver`, `android.process.media` and `surfaceflinger`. In addition to these processes, we also monitor the `debuggerd`—the debugger daemon on Android—and the `init` process, which is the parent of the `mediaserver`. The actions of these two processes are crucial for two reasons: firstly, `debuggerd` is responsible for logging `mediaserver` crashes and `init` is reponsible for restarting the `mediaserver`, as already mentioned.

Type of media	Size range
ISO Media, MPEG v4 system, 3GPP ISO Media, MPEG v4 system, iTunes AVC-LC ISO Media, MPEG v4 system, version 1 ISO Media, MPEG v4 system, version 2 ISO Media, Apple QuickTime movie Ogg data, Theora video	1KB - 10MB

Table 2: Type of files and size range for *goodware* used in our study.

3.1.2 Dataset used for Evaluation

As pointed out in [7], a number of critical vulnerabilities have been reported for Android platforms[3]. Particularly, the Android `mediaserver` has been repeatedly targeted recently through the *Stagefright* media playback engine. These flaws can be exploited remotely with the aid of maliciously crafted multimedia files, so that the device is compromised without requiring the action of the victim.

The `mediaserver` is an important system service which is used to play media files on Android platforms. Given its central role in mobile and handheld devices as well as the seriousness of the vulnerabilities reported for it, we focus our attention on this service as a case study to deploy and test the detection capabilities of *CoME*. Recent vulnerabilities for Android `mediaserver` are grouped into four main categories: Denial of Service (DoS), code execution, overflow and memory corruption. In fact, these four groups together constitute more than 75% of the total vulnerabilities for Android, and they all can be exploited by using some of a well-know set of techniques: stack overflow, heap corruption, format string and race condition.

For our evaluation, we collected a dataset of normal media files (*goodware*) and a dataset of crafted media files (*malware*). For the *goodware*, we queried GooglePlay and retrieved about 15,000 apps from which we extracted all media files and we randomly selected a total number of 264 MP4 files. Our dataset contains a wide range of different MPEG-4 visual and audio encoded files with a variety of sizes. Table 2 shows an excerpt of the formats and summarizes the range of sizes.

For the *malware*, we obtained a number of crafted media files released by Zimperium [25] exploiting recent vulnerabilities affecting most Android versions before Lollipop 5.1.1. These files were all crafted to expose the Stagefright media library vulnerabilities. Table 3 describes the type of vulnerability exploited by each crafted file selected. As the current version of our emulator is enhanced with the CopperDroid plugin, which runs Android Kitkat 4.4.2, we used only those files for which the associated vulnerability was not patched and which could be used to run remote code execution attacks with Kitkat 4.4.2. Although the Android version used to extract features is Android 4.4 and the crafted media files were tailored to exploit Android 5.1

[3]http://www.cvedetails.com/product/19997/Google-Android.html?vendor_id=1224

ID	CVE	Description
SF-01		*ctts*
SF-02	CVE-2015-1538	*stts*
SF-03		*stss*
SF-04	CVE-2015-3827	*covr*
SF-05	CVE-2015-3824	*tx3g*
SF-06	CVE-2015-3829	*covr*

Table 3: Crafted MP4 files, vulnerability identifier and name of the MP4 atom name that leads to the Stagefright exploitation. All the files can enable remote code execution through integer overflow vulnerabilities.

or below, it is worth noting that vulnerabilities based on the ones tested in this paper have been recently used to run remote code execution attacks in the Android media framework against versions up to Oreo (Android 8.0[4]).

3.2 Performance Results

CopperDroid generates a set of features at a constant sampling rate during the execution of media files. For the experimentation of this paper, we consider a total of 692 features. These features contained all behaviors reported after monitoring the `mediaserver` process. We refer the reader to Table 1 for a summary of all behaviors captured by CopperDroid. Our data consisted of a total of 71,336 time observations of the 692 features derived from the execution of 298 files (264 goodware and 34 malware samples). This execution was of variable duration. So, the number of logged observations per file was also variable. For the 298 files monitored during our evaluation, we mainly observed: (i) *File Access* to different libraries and data files, (ii) low-level *System Calls*, and (iii) different *Binder* transactions. We also captured an additional set of features from other related processes such as *debuggerd* or *system_server* (see Section 3.1.1). We treated behaviors reported by each process as individual features. This way, a call to *sendto* executed by the `mediaserver` process constitutes a different feature than a call from the *system_server*.

Note that, unlike MSNM, most anomaly detection techniques cannot handle a big number of features [15]. When using MSNM we do not need to perform a feature selection on first place, avoiding the risk of discarding useful information for anomaly detection.

In this section, we illustrate two applications for *CoME*. First, an off-line exploratory data analysis on average data is illustrated. A second application is *o*n-line monitoring. MSNM can be used to identify anomalies on a timely manner during the file execution, with the goal of swiftly thwarting the execution of the malware, hopefully minimizing the harm caused.

3.2.1 *Off-line analysis*

The off-line analysis is useful to gain data understanding. For instance, to identify the features that best discriminate goodware and malware in the problem at hand, or those features that may lead to undesired properties in the anomaly detector. To perform this analysis, we average the observations of a file, so that each file is represented by a single feature vector with 692 values. The total dataset is a matrix of 298×692, from which the last 34 rows correspond to malware samples. We call this an *o*ff-line or forensic analysis because it is performed after the execution of the media files.

[4]See https://www.cvedetails.com/cve/CVE-2017-0809/

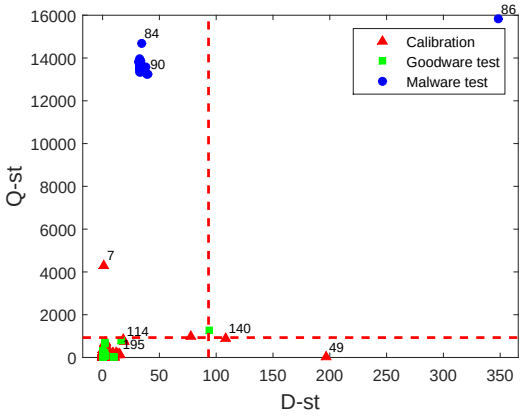


Figure 3: MSNM monitoring chart: Q-st vs D-st. Each point represents a file. Calibration files, test goodware and test malware are shown as triangles, squares and circles, respectively. Control limits are estimated from calibration files.

Calibration of the Anomaly Detector.

The MSNM related model is firstly calibrated with 'normal' data corresponding to goodware samples. Afterwards, this system can be used to identify and diagnose anomalies. We split the total dataset of 298×692 into a calibration part, of size 200×692 and solely composed of goodware, and a test part with 98×692, with 65% of goodware and 35% of malware. The calibration part was used to select the number of PCs, identify the PCA model, and compute the monitoring statistics and their corresponding control limits. All these steps were performed following the recommendations in [15], which for the sake of brevity are not repeated here. Then, the test dataset was analyzed by the MSNM system.

The result obtained is shown in Figure 3. We can see that the Q-st is highly efficient in discriminating goodware and malware. Control limits are adjusted at a 99% confidence level, so that 99 out of 100 calibration goodware files yield statistics below the limits. It can be seen that this adjustment holds for the tested goodware, since only one test goodware file out of 64 exceeds the limits, and they are exceeded by a reduced margin. Although the results are satisfactory, we may consider that calibration goodware files number 7 and 49 are outliers and deserve further study.

Explaining Prominent Features.

The MSNM analysis in Figure 3 shows that there is representative information for discrimination in the 692 features provided by CopperDroid. We can take advantage of the multivariate tools in the MSNM approach to investigate which are the most valuable features for discrimi-

Feature	Process	Action	Details
$f49$	mediaserver	syscall	`mprotect`
$f508$			`access`
$f9$			`advise`
$f308$			`mmap2`
$f479$			`close`
$f72$		filesystem	`/system/lib/libbinder.so`
$f108$			`/system/lib/libaudioutils.so`
$f1$			`/system/lib/libcorkscrew.so`
$f128$	debuggerd	filesystem	`/dev/log/system`
$f202$			`/dev/log/main`
$f194$	system_server	syscall	`sigprocmask`
$f199$			`sendto`
$f80$		binder	`IPhoneStateListener.onChanged`[5]
$f18$	com.android.music	binder	IAudioService.setPlaybackInfoForRcc[6]

Table 4: Excerpt of characteristic features distinguishing goodware/malware.

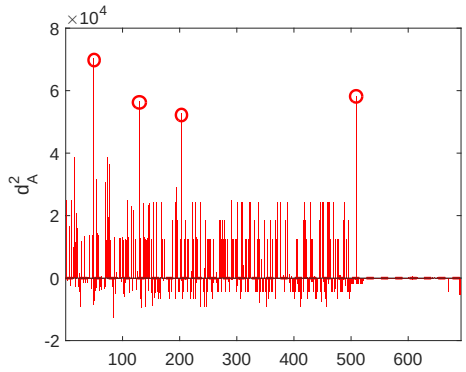
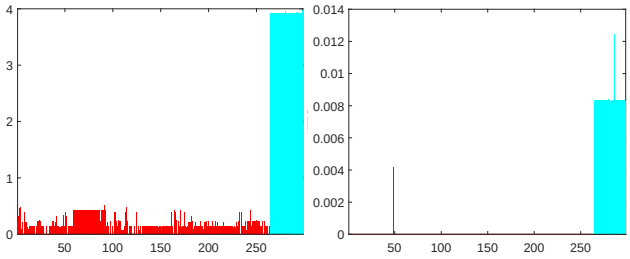


Figure 4: Bar oMEDA plot to determine the relevance of features to discriminate goodware and malware. Values of higher magnitude represent relevant features out of the complete set of 692. The four most relevant features are highlighted with circles.

nation. Note that there is no actual need in reducing the number of features in the MSNM module. Rather, the goal here is to improve our understanding on the data and the results. With this understanding, we may in turn elucidate ways to further improve the anomaly detection and/or to assess to what extent the achieved detection performance can be generalized to new types of malware and goodware. This is a main advantage of MSNM over other anomaly detection methods: MSNM not only provides high performance results but also an interpretable model of the data. In comparison, most machine learning techniques yield black box models with an uninterpretable linkage between input and output data.

Figure 4 shows the oMEDA plot to assess the specificities of malware according to MSNM. In other words, this plot shows the features that better help distinguish malware from goodware, for the malware samples considered. Although the discrimination fingerprint affects many features, we can see that there are four of them that stand out of the others. These are highlighted with a red circle in the figure. Figure 5 shows the values of the complete set of files (calibration and test) for two of the features. The values for the other two features follow a similar trend and are shown in Appendix A. Recall these values are averaged along the duration of the files execution. According



(a) f49 (b) f128

Figure 5: Scores in the four variables highlighted in Figure 4 for the complete set of files: goodware in dark red color, malware in light blue color.

to the figures, any of the four features, by itself, could be used to calibrate an anomaly detection system, since the values for malware samples (light blue color) exceed those for goodware samples (dark red color) by a large margin, at least one order of magnitude. Notice that we only need a pair of multivariate plots (Figs. 3 and 4) to assess the discrimination capability for malware detection of a total of 692 features. Clearly, the exploratory ability of this set of multivariate tools simplifies the analysis of complex and highly dimensional datasets. Note that results provided in Figure 4 claim for a multivariate discrimination between malware and goodware, *i.e.*, the analyzed malware shows high values in the four variables at the same time, and also on many others according to oMEDA. This multivariate discrimination is more robust than its univariate counterpart, which may lead to a higher number of false positives.

The top features highlighted by oMEDA (including the 4 highlighted in Figure 4), are listed in Table 4. Anomalous behaviors stemming from opening a crafted media file leads to the mediaserver being restarted. Therefore, it is unsurprising that the top features for distinguishing benign behavior from potentially dangerous ones are derived from the Android debugger; both */dev/log/system* and */dev/log/main* are log files that are typically written after a service has crashed. When the mediaserver restarts, it loads various libraries and sets the memory segment permissions for loaded libraries. Therefore, we observe that a number of system calls related to memory manipulation such as `mprotect`, `mmap2`, `advise` and `access` are also among the distinguishing features. Additionally, it can also be observed that the access to these libraries—after the mediaserver is restarted—is observed in the list of features discriminating benign from dangerous actions.

Understanding Outliers.

The MSNM approach is also powerful to identify and diagnose outliers. We will illustrate this with the two outliers previously identified in Figure 3: calibration goodware files number 7 and 49, with high Q-st and high D-st, respectively. For diagnosing the special behavior in each of them, we follow the same approach than with malware, that is, we use oMEDA plots (not shown). The oMEDA for file number 7 identified a number of features in which this

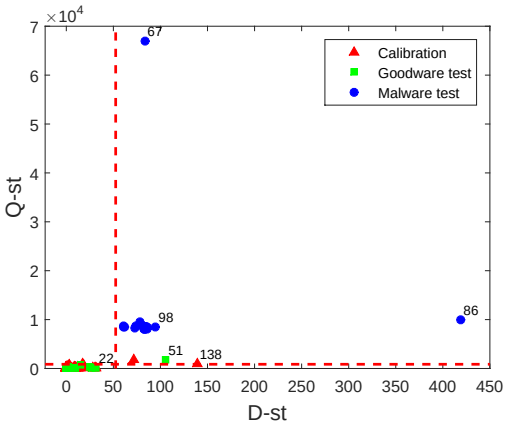
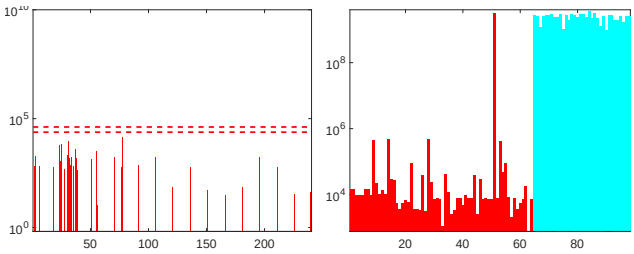


Figure 6: MSNM monitoring chart after removal of outliers 7 and 49: Q-st vs D-st. Each point represents a file. Calibration files, test goodware and test malware are shown as triangles, squares and circles, respectively. Control limits are estimated from calibration files.



(a) Q-st (b) Max Q-st

Figure 7: The online monitoring Q-st charts for a goodware file in the MSNM system is shown in the figure on the left; and the maximum Q-st value of the MSNM statistics in test files is shown in the figure on the right. Both figures are displayed in logarithmic scale.

file obtained very low values. Inspecting those features, we found that it is the only file in the complete set of files with 0 value in features $f18$ and $f80$ and $f199$ (see Table 4). On the contrary, the oMEDA analysis for file 49 showed that this sample presents a behavior similar to the one observed in malware (for at least many observed features). An example is feature $f1$.

There are different approaches to handle outliers, depending on the information found in the diagnosis and also on the context. One typical approach is to discard the outliers. The effect of outliers removal is illustrated in Figure 6, where the MSNM system is re-calibrated after discarding both calibration goodware files number 7 and 49. In the new system, the D-st is also useful for detection, and the level of detection in the Q-st is augmented in one order of magnitude.

3.2.2 *On-line analysis*

For our experiments on on-line analysis, we considered the complete set of observations. As already mentioned, our dataset consists of 71,226 observations of 692 features (71336×692). The last 8161 observations are derived from the 34 malware files. This dataset was divided into two parts: one for calibration and another for testing. The calibration part consisted of size $47,818$ observations derived from the first 200 goodware files. The test part with $23,518$ observations derived from the rest of goodware and malware files.

Figure 7a illustrates the online MSNM system applied over a goodware file for the Q-st. We refer the reader to Figure 11a in Appendix B for the D-st charts. As it can be seen, statistics remain below the control limits. Although in this illustration the monitoring charts contain the complete evolution of the file execution, in a production monitoring system the statistics computed for a specific sampling time would be shown in the charts as soon as the information is obtained. That way, the execution can be stopped right after the control limits are exceeded by a large margin or on a consistent basis.

Figure 7b and Figure 11b (in Appendix B) show the performance of the online MSNM system. Here, the maximum value of the statistics computed per file is shown for test files. Again, we can see that the Q-st makes a clear discrimination between malware and goodware, except for test goodware file number 51. Interestingly, this false positive was also highlighted in our offline analysis assessment in Figure 6. Although the D-st has a lower discrimination capability (see Appendix B), malware files tend to show higher D-st values than goodware. We can assess the discrimination quality of the monitoring charts with ROC curves obtained by varying the α parameter in the control limits. Results are shown in Figure 8, where we included the results by a one-class support vector machine (OCSVM) [26, 27] computed with the LIBSVM library [28]. The OCSVM is a classification-based network anomaly detection method [29] reported to provide excelent results [30]. Following [31] we used default values for the metaparameters. Then, the ROC curve was obtained by varying the bias term [32]. According to these ROC curves, the detection performance of the online MSNM system is impressive. While the D-st is outperformed by the one class SVM classifier, the Q-st clearly outperforms the latter.

Figure 9 compares the monitoring charts in logarithmic scale of the outlier (goodware file number 51) and one malware file. There is a time interval of maximum activity around sampling time 30. It is during this interval that the malware presents much higher activity than the goodware. Recall that we are using a logarithmic scale to improve visualization. The outlier goodware presents an anomalous behavior (one single observations) right at the beginning of the execution. If future false positives follow the same pattern, we can always distinguish them from malware for their temporal specificities.

Detections in both the outlier and the malware can be diagnosed with oMEDA. This is shown in Figure 9c. We can see that the outlier can also be discriminated from malware with the diagnosis, as very different features are highlighted. This is the case, for instance, of feature *f339* detected while running the goodware 51. This feature is

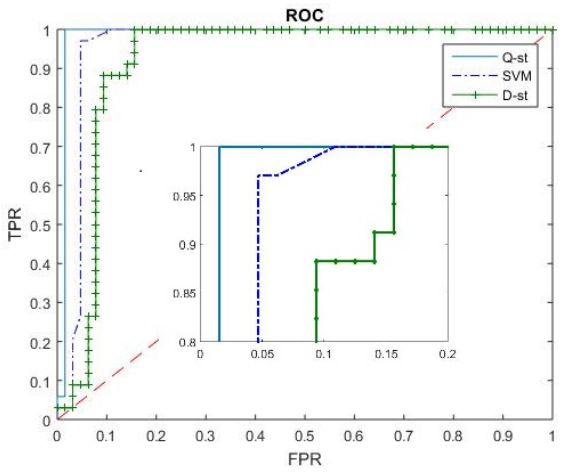


Figure 8: ROC curves for the MSNM statistics in test files. One-class support vector machine results are presented as a reference. The ROC curves are obtained by varying the value of parameter α in the control limits of the MSNM system and the bias term in the SVM [32].

related to the Android Volume Manager and it is a system call (*openat*) triggered by the the *vold* daemon. This type of behavior should only be attributed to good behaviors as diagnosed by oMEDA.

3.3 Discussion

To provide on-line detection of malicious intents on devices, it is essential to understand normal operating conditions (NOC) and identify sequence of actions that deviate from the NOC. Many machine learning techniques that perform malware analysis manage to detect malware. However, in most cases, this detection is done off-line and often with a black-box model where it is not possible to meaningfully explain the linkage between input and output data.

We should also take into account that MSNM is an anomaly detector based on a model of normality (NOC), and malware is detected because it shows a different behavior to modeled NOCs. Thus, the calibration dataset should be as representative of NOC as possible. If during the MSNM operation one goodware file is mistaken by malware, that is, in the event of a false positive, we should diagnose with oMEDA the reason for this detection and update the MSNM system to yield the correct outcome for this event in the future.

The MSNM methodology provides an interpretable model of the data which makes explanation of the results easier. Furthermore, it contributes to the identification of root-cause problems even when the source of the anomaly can only be derived after observing multiple variables across time. For instance, for the case of the Android `mediaserver`, we could observe that the NOC of different features such as *mprotect*, and *access* syscalls maintain a very low profile. Contrary, when any of the Integer overflows are triggered, the `mediaserver` crashes. Only right after being restarted, the `mediaserver` repeatedly invokes *access* and *mprotect* to load all the different native components of the *audioflinger* and *surfaceflinger* such as *libaudioutils.so* (see Table 4). This usually happens together with the dump of the crash report to any of the files listed in */dev/log/*. Similarly, we can observed that the absence of number of features such as those related to the Binder (*com.android.internal.telephony* and *android.media* actions) only happen during the NOC. All in all, this means that *CoME* is able to list all those features related to the crash of the `mediaserver` and correlates them with the absence of several binder transactions that should occur during the normal operation of the service.

One key feature of *CoME* is that it does not require an understanding of the domain or prior knowledge about the monitored component. This makes our approach suitable for both explaining and detecting unknown malicious activities or zero-day exploits. Our evaluation over four different CVE showed that even when confronted with a complex system, *CoME* is capable of modeling those conditions needed to detect unexpected behaviors. Contrary to other anomaly-based systems, *CoME* is capable of dealing with systems containing a big number of features.

Most of the existing attacks are tailored at vendor-specific Android OSes. In our work, we instrument vanilla Android emulator using CopperDroid. A limitation of our evaluation stems from the lack of exploits targeting, in general, vanilla Android systems, and in particular CopperDroid. Although CopperDroid is engineered to automatically generate system call and inter-component communication introspection stubs, the system has not been evaluated on vendor-specific Android platforms. For this reason, we limit the scope of our work to the most prevalent set of vulnerabilities affecting all vanilla systems (including stock hardware smartphones) from Android 4.4.4 to 8.0.

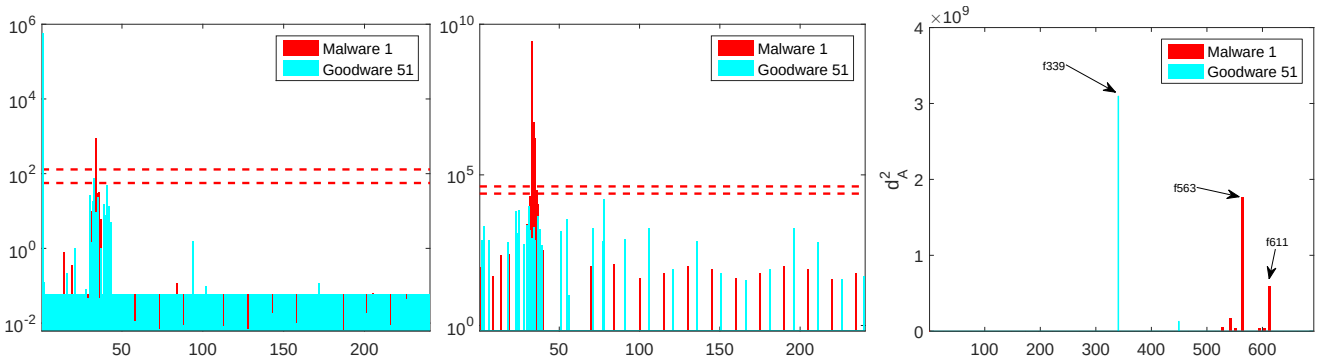
According to Google[7], at the time of writing this paper about 99.70% of the users run an Android device between Android 4.4 and 8.0. This gives a rough intuition of the scope of this threat. Novel but less popular exploits are also relevant to our system. A vendor-specific version of our monitoring system would grant us access to a larger set actionable exploits, and would allow us to test novel exploits.

4. RELATED WORK

The majority of the approaches proposed in the literature for malware detection focus on general detection strategies using either anomaly or misuse detection for both static and dynamic analysis [33]. However, due to the diversity of malware goals and incentives [34], it is often desired to narrow down the complexity towards detecting specific classes of malware such as privileged escalation [35], battery-depletion attacks [36], or information leakage [37]. In this work, we shall focus on a privileged escalation attack because of its impact.

There are two common types of privilege escalation attacks according to whether the exploitation strategy focuses on inter-process capability leakage or system vulner-

[7]https://developer.android.com/about/dashboards/index.html



(a) D-st (b) Q-st (c) oMEDA for goodware 51 and malware 1

Figure 9: Online monitoring charts for different files in the MSNM system (in logarithmic scale): a) D-statistic, b) Q-statistic and c) Diagnosis based on oMEDA, highlighting the relevance of the features in the detected anomalous sampling times.

abilities. Analyzing inter-process capability leakage has gained a lot of attention over the last years with approaches such as XManDroid [38], Woodpecker [39], Elish et al. [40] or CHEX [41]. However, system vulnerabilities pose a higher risk to users because they can enable full escalation of privileges, simply called root access [42].

There are several recent works aimed at systematically studying and characterizing root vulnerabilities in Android [43, 44, 45]. Authors in [43] present a static-based framework to detect apps that could potentially use root exploits. Likewise any other static analysis technique, their approach is vulnerable to code obfuscation and other more sophisticated forms of evasion. In addition to static analysis, authors in [44] present a system to detect both app-level and kernel-level vulnerabilities in device firmware images using dynamic analysis. Contrary to our approach, authors do not explore framework-level vulnerabilities which constitute a fundamental part of the Android OS. Furthermore, our approach is driven by multivariate statistical theory supporting anomaly detection. This constitutes a novel perspective to deal with the problem.

Similar to our approach, authors in [45] present a root exploit containment system for Android called PREC. PREC is a dynamic-based anomaly detection system that monitors system calls from both benign and malign apps. After extracting and labeling system calls, they build a learning scheme based on self-organizing maps (SOM). One key different with respect to our approach is that PREC only monitors behaviors produced from the applications being executed, whereas in our approach we holistically consider the entire system. Specifically, PREC only monitors system calls from third party libraries used within the process of the app. Contrary, we monitor all processes related to a given exploitable service. Additionally, we not only monitor system calls, but we are also able to extract other high level behaviors such as Binder transactions in a transparent fashion. This ultimately allows us to perform more fine-granular detection and to target a broader type of exploits. Let's take for instance a malware manages to run a reverse shell after exploting an Integer Overflow in the media service with a crafted MP4 file. Except for the trigger of an Intent action asking the Android OS to play the MP4 file, PREC would miss the actual trace of system calls leading to the exploitation.

In this context, multivariate techniques have shown to be promising when dealing with complex observations of diverse nature and origin [46]. However, to the best of our knowledge, very few multivariate-based malware detection proposals are deployed in the literature at present. So far, as a single example of this we can cite [47], which is poor from the perspective of the actual capabilities argued for multivariate methodologies.

5. CONCLUSION

In this paper, we have presented a multivariate statistical approach to identify abnormal uses of general components in Android. Our approach, called *CoME*, uses lightweight introspection to perform transparent behavioral analysis at runtime. As a key novelty of our approach, we showed how anomaly-based multivariate systems can effectively be used to identify technical software exploitations based on unknown vulnerabilities. More precisely, we use *CoME* to estimate the normal operating conditions of the Android `mediaserver`. We have illustrated two applications: (i) an off-line system capable of aiding security experts on the identification of relevant discriminant features; and (ii) on-line monitoring and analysis system for accurate detection at runtime.

6. ACKNOWLEDGMENTS

This work is partialy supported by the Spanish *Ministry of Economy and Competitiveness* and FEDER funds through projects TIN2014-60346-R and TIN2017-83494-R, and the UK EPSRC grant EP/L022710/1.

7. REFERENCES

[1] Worldwide Quarterly Smart Connected Device Tracker, Tech. rep., International Data Corporation (2015).

[2] Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2015-2020, Tech. rep., Cisco (2015).

[3] G. Suarez-Tangil, G. Stringhini, Eight years of rider measurement in the android malware ecosystem: Evolution and lessons learned, arXiv preprint arXiv:1801.08115.

[4] J. Lyne, Security Threat Trends 2015. Predicting what Cybsersecurity will Look Like in 2015 and Beyond, Tech. rep., Sophos (2015).

[5] Kaspersky Security Bulletin 2015. 2016 Predictions, Tech. rep., Kaspersky (2015).

[6] 2016 Internet Security Threat Report, Tech. rep., Symantec (2016).

[7] C. Cao, N. Gao, P. Liu, J. Xiang, Towards Analyzing the Input Validation Vulnerabilities associated with Android System Services, in: Proceedings of the 31st Annual Computer Security Applications Conference (ACSAC), 2015, pp. 361–370.

[8] M. Jimenez, M. Papadakis, T. F. Bissyandé, J. Klein, Profiling Android Vulnerabilities, in: Software Quality, Reliability and Security (QRS), 2016 IEEE International Conference on, IEEE, 2016, pp. 222–229.

[9] V. van der Veen, Y. Fratantonio, M. Lindorfer, D. Gruss, C. Maurice, G. Vigna, H. Bos, K. Razavi, C. Giuffrida, Drammer: Deterministic Rowhammer Attacks on Mobile Platforms, in: 23rd ACM Conference on Computer and Communications Security (CCS), 2016.

[10] Y. Fratantonio, A. Bianchi, W. Robertson, E. Kirda, C. Kruegel, G. Vigna, TriggerScope: Towards Detecting Logic Bombs in Android Apps, in: Proceedings of the IEEE Symposium on Security and Privacy (S&P), San Jose, CA, 2016.

[11] L. S.-C. A. Ruiz-Heras, P. García-Teodoro, ADroid: Anomaly-based Detection of Malicious Events in Android Platforms, International Journal of Information Security (2016) 1–14.

[12] S. Forrest, S. Hofmeyr, A. Somayaji, The Evolution of System-call Monitoring, in: Computer Security Applications Conference, 2008. ACSAC 2008. Annual, IEEE, 2008, pp. 418–430.

[13] D. Mutz, F. Valeur, G. Vigna, C. Kruegel, Anomalous System Call Detection, ACM Transactions on Information and System Security (TISSEC) 9 (1) (2006) 61–93.

[14] S. F. O. Pieczul, Runtime Detection of Zero-Day Vulnerability Exploits in Contemporary Software Systems, in: Data and Applications Security and Privacy XXX, DBSec 2016, LNCS 9766, 2016, pp. 347–363.

[15] J. Camacho, A. Pérez-Villegas, P. García-Teodoro, G. Maciá-Fernández, PCA-based Multivariate Statistical Network Monitoring for Anomaly Detection, Computers & Security 59 (2016) 118–137.

[16] K. Tam, S. Khan, A. Fattori, L. Cavallaro, CopperDroid: Automatic Reconstruction of Android Malware Behaviors, in: NDSS, 2015, pp. 1–15.

[17] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, E. Vázquez, Anomaly-based Network Intrusion Detection: Techniques, Systems and Challenges, Computers & Security 28 (2009) 18–28.

[18] J. Camacho, A. Pérez-Villegas, R. Rodríguez-Gómez, E. J.-M. nas, Multivariate Exploratory Data Analysis (MEDA) Toolbox for Matlab, Chemometrics and Intelligent Laboratory Systems 143 (2015) 49–57.

[19] S. Dash, G. Suárez-Tangil, S. Khan, K. Tam, M. Ahmadi, J. Kinder, L. Cavallaro, DroidScribe: Classifying Android Malware Based on Runtime Behavior, in: Mobile Security Technologies (MoST 2016), 2016, pp. 252–261.

[20] W. Enck, M. Ongtang, P. McDaniel, Understanding Android Security, IEEE Security & Privacy (1) (2009) 50–57.

[21] T. Garfinkel, M. Rosenblum, et al., A Virtual Machine Introspection Based Architecture for Intrusion Detection, in: NDSS, Vol. 3, 2003, pp. 191–206.

[22] A. Lakhina, M. Crovella, C. Diot, Diagnosing Network-wide Traffic Anomalies, ACM SIGCOMM Computer Communication Review 34 (4) (2004) 219–230.

[23] T. Kourti, J. F. MacGregor, Multivariate SPC methods for process and product monitoring, Journal of Quality Technology 28 (4) (1996) 409–428.

[24] J. Camacho, Observation-based Missing Data Methods for Exploratory Data Analysis to Unveil the Connection Between Observations and Variables in Latent Subspace Models, Journal of Chemometrics 25 (11) (2011) 592–600.

[25] Z. Avraham, J. Drake, N. Bassen, Experts Found a Unicorn in the Heart of Android, Available Online. https://blog.zimperium.com/experts-found-a-unicorn-in-the-heart-of-android/ (2015).

[26] B. Scholkopf, A. J. Smola, R. C. Williamson, P. L. Bartlett, New Support Vector Algorithms, Neural computation 12 (5) (2000) 1207–1245. doi:10.1162/089976600300015565.

[27] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, R. C. Williamson, Estimating the Support of a High-Dimensional Distribution, Neural Computation 13 (7) (2001) 1443–1471. doi:10.1162/089976601750264965. URL http://www.mitpressjournals.org/doi/abs/10.1162/089976601750264965

[28] C.-C. Chang, C.-J. Lin, LIBSVM: A Library for Support Vector Machines, ACM Transactions on Intelligent Systems and Technology 2 (2011) 27:1–27:27, software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[29] M. H. Bhuyan, D. K. Bhattacharyya, J. K. Kalita, Network Anomaly Detection: Methods, Systems and Tools, IEEE Communications Surveys & Tutorials 16 (1) (2014) 303–336. doi:10.1109/SURV.2013.052213.00046. URL http://ieeexplore.ieee.org/lpdocs/epic03/

wrapper.htm?arnumber=6524462

[30] L. M. Manevitz, One-Class SVMs for Document Classification 2 (2001) 139–154.

[31] J. A. Quinn, M. Sugiyama, A Least-Squares Approach to Anomaly Detection in Static and Sequential Data, Pattern Recognition Letters 40 (1) (2014) 36–40. doi:10.1016/j.patrec.2013.12.016.
URL http://dx.doi.org/10.1016/j.patrec.2013.12.016

[32] K. Heller, K. Svore, A. D. Keromytis, S. Stolfo, One Class Support Vector Machines for Detecting Anomalous Windows Registry Accesses, Workshop on Data Mining for Computer Security (DMSEC), Melbourne, FL, November 19, 2003.
URL http://sneakers.cs.columbia.edu/ids/publications/ocsvm.pdf

[33] H. Chuang, S. Wang, Machine Learning based Hybrid Behavior Models for Android Malware Analysis, in: IEEE International Conference on Software Quality, Reliability and Security, 2015, pp. 201–216.

[34] G. Suárez-Tangil, J. Tapiador, P. Peris, A. Ribagorda, Evolution, Detection and Analysis of Malware for Smart Devices, IEEE Communications Surveys & Tutorials 16 (2) (2014) 961–987.

[35] S. Checkoway, L. Davi, A. Dmitrienko, A. Sadeghi, H. Shacham, M. Winandy, Return-Oriented Programming without Returns, in: A. Keromytis, V. Shmatikov (Eds.), Proceedings of CCS 2010, 2010, pp. 559–72.

[36] H. Kim, J. Smith, K. Shin, Detecting Energy-greedy Anomalies and Mobile Malware Variants, in: Proceedings of the 6th international conference on Mobile systems, applications, and services, 2008, pp. 239–252.

[37] S. Rosen, Z. Qian, Z. Mao, AppProfiler: A Flexible Method of Exposing Privacy-related Behavior in Android Applications to End Users, in: Proceedings of the third ACM conference on Data and application security and privacy, 2013, pp. 221–232.

[38] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, A. Sadeghi, XManDroid: A New Android Evolution to Mitigate Privilege Escalation Attacks, Tech. rep., Technische Universitat Darmstadt (2011).

[39] M. Grace, Y. Zhou, Z. Wang, X. Jiang, Systematic Detection of Capability Leaks in Stock Android Smartphones, in: Proceedings of the 19th Annual Symposium on Network and Distributed System Security, 2012.

[40] K. Elish, D. Yao, Ryder, G. Barbara, X. Jiang, A Static Assurance Analysis of Android Applications, Tech. rep., Virginia Polytechnic Institute and State University (2013).

[41] L. Lu, Z. Li, Z. Wu, W. Lee, G. Jiang, CHEX: Statically Vetting Android Apps for Component Hijacking Vulnerabilities, in: Proceedings of the 2012 ACM conference on Computer and communications security, 2012, pp. 229–240.

[42] M. Grace, Y. Zhou, Q. Zhang, S. Zou, X. Jiang, RiskRanker: Scalable and Accurate Zero-day Android Malware Detection, in: Proceedings of the 10th international conference on Mobile systems, applications, and services, 2012, pp. 281–294.

[43] H. Hao, Z. Li, Y. He, J. Ma, Characterization of Android Applications with Root Exploit by Using Static Feature Analysis, in: Algorithms and Architectures for Parallel Processing, Springer, 2015, pp. 153–165.

[44] L. Wu, Vulnerability Detection and Mitigation in Commodity Android Devices, Ph.D. thesis, North Carolina State University (2015).

[45] T. Ho, D. Dean, X. Gu, W. Enck, PREC: Practical Root Xxploit Containment for Android Devices, in: Proceedings of the 4th ACM conference on Data and application security and privacy, 2014, pp. 187–198.

[46] J. Camacho, R. Magán-Carrión, P. García-Teodoro, J. Treinen, Networkmetrics: Multivariate Big Data Analysis in the Context of the Internet, In press in Journal of Chemometrics (Wiley) (2016) 1–45.

[47] K. Kim, M. Choi, Android Malware Detection Using Multivariate Time-Series Technique, in: Network Operations and Management Symposium (APNOMS), 2015 17th Asia-Pacific, 2015, pp. 198–202.

APPENDIX

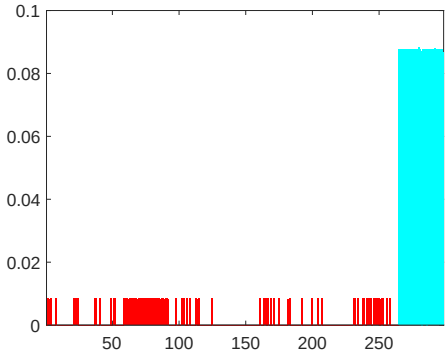
A. EXPLANATION OF $F508$ AND $F202$

As mentioned before in Section 3.2, the most relevant features highlighted by oMEDA are listed in Table 4. Out of all features, in this paper we have highlighted four of them: two system calls and two file system access (FS), which are as shown in Figure 4. In this subsection, we show further details on $f508$ and $f202$, corresponding to a syscall and a FS, respectively.

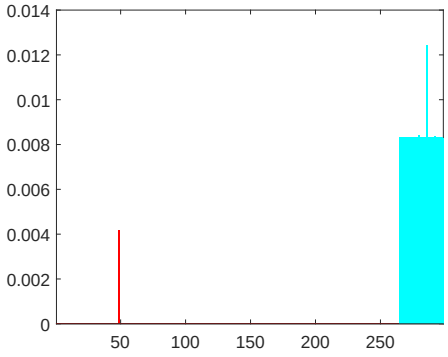
Figure 10 shows the values of the complete set of files (calibration and test) for these two features. Similar to what we observed with the other top features, we can see that $f508$ and $f202$ can clearly tell apart benign behaviors from the malicious ones.

B. ONLINE ANALYSIS

This section reports additional details of the online analysis presented in Section 3.2.2. In particular, we illustrate the online MSNM system applied over a goodware file for the D-st test as illustrated in Figure 11. As shown with the Q-st test, statistics for this test remain below the control limits.

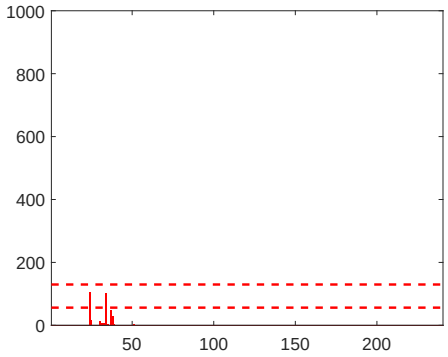


(a) f508

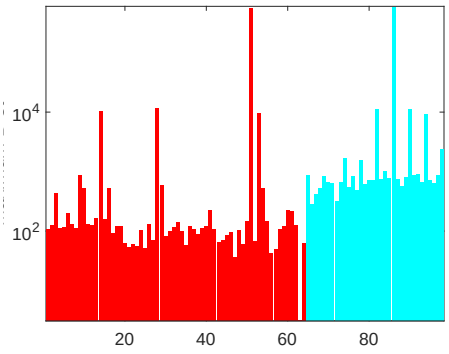


(b) f202

Figure 10: Scores in the remaining two variables highlighted in Figure 4 for the complete set of files: goodware in dark red color, malware in light blue color.



(a) D-st



(b) Max D-st

Figure 11: The online monitoring D-st charts for a goodware file in the MSNM system is shown in the figure on the left; and the maximum D-st value of the MSNM statistics in test files is shown in the figure on the right. Both figures are displayed in logarithmic scale.