# R-PackDroid: API Package-Based Characterization and Detection of Mobile Ransomware

Davide Maiorca
Department of Electrical and Electronic Engineering
University of Cagliari
Cagliari, Italy
davide.maiorca@diee.unica.it

Francesco Mercaldo
Institute for Informatics and Telematics
National Research Council
Pisa, Italy
francesco.mercaldo@iit.cnr.it

Giorgio Giacinto
Department of Electrical and Electronic Engineering
University of Cagliari
Cagliari, Italy
giacinto@diee.unica.it

Corrado Aaron Visaggio
Department of Engineering
University of Sannio
Benevento, Italy
visaggio@unisannio.it

Fabio Martinelli
Institute for Informatics and Telematics
National Research Council
Pisa, Italy
fabio.martinelli@iit.cnr.it

## ABSTRACT

Ransomware has become a serious and concrete threat for mobile platforms and in particular for Android. In this paper, we propose R-PackDroid, a machine learning system for the detection of Android ransomware. Differently to previous works, we leverage information extracted from system API packages, which allow to characterize applications without specific knowledge of user-defined content such as the application language or strings. Results attained on very recent data show that it is possible to detect Android ransomware and to distinguish it from generic malware with very high accuracy. Moreover, we used R-PackDroid to flag applications that were detected as ransomware with very low confidence by the VirusTotal service. In this way, we were able to correctly distinguish true ransomware from false positives, thus providing valuable help for the analysis of these malicious applications.

## CCS Concepts

**•Security and privacy → Malware and its mitigation; Mobile and wireless security; Software security engineering;**

## Keywords

Malware; Android; Ransomware; Machine Learning

## 1. INTRODUCTION

Ransomware is any type of malware that demands a sum of money from the infected user while promising to "release" a hijacked resource in exchange. Ransomware targets both private users and public organizations, thus often causing significant economical losses. For instance, The Atlantic recently reported on a sequence of attacks that impacted small police departments in Massachusetts, Tennessee and New Hampshire, where hackers extorted \$500 to \$750 for the departments to regain access to their critical, encrypted data[1].

The number of attacks based on ransomware has exponentially grown. More than 4 million ransomware samples were identified in the second quarter of 2015, indicating a worrisome trend in comparison to 2014, where fewer than 1.5 million samples were analyzed. According to Security Magazine[2], this number is expected to further increase in 2016.

Ransomware has also started targeting mobile platforms during the past years, with a particular focus on the Android platform. As shown by Kaspersky[3], the proportions of malware-affected users targeted by ransomware almost doubled from 2014-2015 to 2015-2016.

Ransomware-based attacks employ various strategies to extort victims. In the most common case, once a victim has downloaded and installed a fake or Trojanized app, the malware locks the screen and displays a bogus alert claiming the user has accessed forbidden materials. Meanwhile, the malware gathers the victims' contacts list and encrypts their data in the background. Users will then be prompted to pay a ransom, threatened by the loss of the encrypted data and the submission of the user's browsing history to all their contacts.

Signature-based approaches (e.g., anti-malware) are often inadequate at detecting ransomware. There are two main

*SAC 2017, April 03-07, 2017, Marrakech, Morocco*
© 2017 ACM. ISBN 978-1-4503-4486-9/17/04. . . \$15.00
DOI: http://dx.doi.org/10.1145/3019612.3019793

[1]http://blog.trendmicro.com/ransomware-one-of-the-biggest-threats-in-2016/
[2]http://www.securitymagazine.com/keywords/5254-ransomware
[3]https://securelist.com/analysis/publications/75183/ksn-report-mobile-ransomware-in-2014-2016/

reasons for that: **1)** The attacks need to be already in the engine database to be recognized. This typically needs 48 hours to occur, but it can span to three months [7]. **2)** Complex techniques such as obfuscation, dynamic code loading or simple string renaming are very effective at thwarting anti-malware detection [5, 6]. Likewise, it has been shown that Google Bouncer (the service used by Google to establish the maliciousness of the applications in its market) can also be evaded by malware writers [8].

Machine learning techniques have been recently used to detect malware in mobile applications. Such approaches resort both to static and dynamic analyis (see, for instance, [3,11]), and they have been applied to ransomware as well [1]. In particular, [1] showed that it is possible to distinguish ransomware from generic malware by resorting to text recognition, static taint-tracking analysis and dynamic analysis. Albeit effective, this approach strongly relies on string analysis and natural language recognition, which is subjected to be evaded by means of obfuscation or by changes to the application language. Moreover, such an approach relies on a precise knowledge of the ransomware encryption mechanisms.

On the basis of the previous works, in this paper we address the following research questions:

- **R.Q. 1: Is it possible to characterize Android ransomware and malware by using an *a priori determined* set of characteristics?**
- **R.Q. 2: Is it possible to use the set of characteristics in R.Q. 1 to reliably discriminate between ransomware, malware and trusted applications?**
- **R.Q. 3: Is it possible to use the set in R.Q. 1 to define previously unseen ransomware?**

We address these questions by introducing a novel method that resorts to a supervised analysis of Android applications. In particular, we characterize the apps by using a list of *system API packages*. Such packages are referenced when a system API call is invoked in the executable code. The key idea is that the usage of such packages can effectively help to understand the various *types* of malicious actions (e.g., crypto-type actions).

In this way, our approach can be used not only to effectively distinguish ransomware from generic malware, but also to recognize new ransomware. This technique avoids analyzing user-defined packages, thus guaranteeing more robustness against obfuscation when compared to string-based approaches. This is because system API calls cannot be easily renamed, as their active implementation is not contained in the Android executable [6].

We evaluated the effectiveness of our approach by using, along with generic malware and trusted files, three datasets of ransomware, which contain data released in 2015 and 2016 (including very recent ransomware). Results show three key findings:

- Our method allows to automatically separate ransomware from generic malware and trusted files with very high accuracy.
- Our method allows to recognize novel ransomware samples by only resorting to data available months before their release.
- Our method can be used to confirm uncertain reports from the VirusTotal service, where only few Antivirus flag an application as ransomware.

To foster research on the topic, we are releasing the hashes of the novel ransomware we used for our paper. We also plan to release `R-PackDroid` as an online service, so that it could be used in combination with VirusTotal or other malware analysis services.

**Paper structure.** The paper proceeds as follows: Section 2 provides the basic concepts of Android apps and ransomware; Section 3 describes the related work in the field; Section 4 describes the employed methodology; Section 5 illustrates the experimental results; Section 6 discusses the limitations of our work, which is finally concluded by Section 7.

## 2. BACKGROUND

### 2.1 Android Apps Basics

Android applications are compressed `.apk` files that contain the following elements: *(i)* The `AndroidManifest.xml` file, which provides the basic application components, the name of the main package, and the permissions that are required for the execution. *(ii)* The `classes.dex` file, which is the true executable of the application, and which contains all the implemented classes and methods (in the form of Dalvik bytecode) that are executed by the app. This file can be disassembled to a simplified format called `smali`. *(iii)* A number of `.xml` files that characterize the application layout. *(iv)* External resources that include, among others, images, native libraries, and other `.dex` files.

In this paper, we focus on analyzing the contents of the `classes.dex` file, as we are interested in retrieving the system API packages.

### 2.2 Android Ransomware

The main functionality of Android ransomware is not so different to that of their x86 counterparts: these malicious apps attempt to encrypt the user's personal data, lock the device and show the ransom request to the victim, typically when the device is turned on. *SimpleLocker*, for example, attempts to encrypt all the user's files contained in his SD card with an AES encryption mechanism.

However, ransomware does not always attempt to encrypt the user's data. This is because encrypting system partitions would require root privileges, which are difficult to obtain by exploiting the system kernel (due to protections such as `DEP` and `ASLR`). Some of them (for instance *Locker*) only lock the system GUI, without performing any changes to the user's files, thus often showing law-enforcement related screens while requesting a ransom (*ScarePackage, Svpeng*). Others randomly open the browser during normal user activities (*Koler*).

The aforementioned ransomware types are part of the dataset that we used for our experiments in Section 5.

## 3. RELATED WORK

Ransomware analysis for the Android platform is a rather novel topic, and only a few works have been released on the topic. The most prominent was the one made by Andronio et al., who introduced HelDroid [1]. This tool includes a text classifier based on NLP features, a lightweight `smali` emulation technique to detect locking strategies, and the application of taint tracking for detecting file-encrypting flows. Albeit effective, this system is computationally demanding. Moreover, it strongly depends on a text classifier: the authors trained it on generic threatening phrases, similar to those that typically appear in ransomware or scareware. This strategy can be easily thwarted by means of, e.g., string encryption [6]. Moreover, it strongly depends on the presence of a language dictionary for that specific ransomware campaign. Some of the analyzed ransomware could not be detected due to missing languages (e.g., Russian).

Other works were proposed for the generic detection of Android malware. Arp et al. proposed Drebin, a machine learning system that uses static analysis to discriminate between generic malware and trusted files. They extracted a lot of features from both the Manifest file and the Android executable, including IP addresses, suspicious API calls, permissions, etc. [3]. Tam et al. introduced a system to perform dynamic analysis and detection of Android malware by analyzing the system calls performed by the application [11].

Aresu et al. clustered Android malware by using the network HTTP traffic generated by those applications [2]. In this way, it is possible to use the clusters to generate signatures that allow to discriminate between malware and legitimate applications.

In [4], the authors experimentally evaluated two techniques for detecting Android malware: the first one is based on Hidden Markov Model (HMM), and the second one exploits Structural Entropy. They demonstrated that these methods attain high precision at detecting Android malware.

## 4. METHODOLOGY

In this Section, we provide details about the methodology we employ to analyze and classify our data.

Our goal is developing a system that, by receiving an Android application as input, is able to label it as one of the following classes: **a) Ransomware**, i.e., applications that perform crypto-based operations on the user data and that typically lock the system GUI; **b) Malware**, i.e., applications that perform malicious operations but that do *not* execute crypto-based operations on the user data and that do not lock the user GUI; **c) Trusted**, i.e., benign applications that do not perform malicious operations.

The main idea to fulfill this goal is using system API packages as *key information* to discriminate between the aforementioned classes. More specifically, this is done in three phases:

1. **Preprocessing**. We statically analyze the Android application executable to determine the *packages* the various APIs belong to.

2. **Feature Extraction**. We count the occurrences of each system API package in the Android app. This leads to a vector of numbers that will be used for the next phase.

3. **Classification**. The data extracted from the feature extractor are sent to a statistical classifier, which is a mathematical function whose parameters have been first tuned by a phase called *training*. On the basis of the received data, the classifier outputs the application label.

It is worth noting that, differently to previous works that employed supervised approaches (with the exception of [1]), our system employs more than two classes, which makes the classification task more complicated in comparison to the typical malicious/benign detection systems.

In the following, we provide a deeper insight into the aforementioned phases. We first describe the type of features that have been adopted in our analysis, and then we provide an insight into the employed classification methodology.

### 4.1 Preprocessing and Feature Extraction

The key idea of our approach is to *statically* analyze the Dalvik bytecode contained in the **classes.dex** file, by searching for the presence of API packages contained in the official Android SDK. Such choice is motivated by two reasons: **1)** Focusing on packages reduces the number of available features, thus limiting the computational complexity of the employed machine learning algorithms; **2)** Focusing on the Android SDK provides a fixed set of features that does not make the feature list dependent on the training data.

Understanding which packages are used in specific applications is also useful to have a better view of the *type* of actions that an application might execute. Since we are especially interested in ransomware, it is reasonable to expect that crypto-related API packages are going to be found in such applications.

In practice, we look for all the `invoke`-type instructions contained in the `classes.dex` code. Then, we check if the APIs contained in the instruction belong to system packages and, if so, we count them. For instance, in the instruction `invoke-virtual {v4, v5, v6, v7}, Ljavax/crypto/Cipher;.init:(ILjava/security/Key;Ljava/security/spec/AlgorithmParameterSpec;)V` we see that `javax/crypto`, `java/security`, and `java/security/spec` are in the System API package list, and they are therefore considered.

We select the API packages belonging to the latest Android release, Android N (API 24 - Android Nougat). This choice is related to the fact that recent API versions have been built as an extension of the previous ones. Thus, as in this paper we also analyze recently released ransomware, employing the latest API guarantees to avoid missing API from recent samples. This leads to 243 API packages.

### 4.2 Classification

We resort to a supervised approach for our classification task. In this approach, the system is trained with a number of samples for which we exactly know if they are benign, generic malware or ransomware. Such technique has been used in previous work with very good results [3]. It is worth noting that we also considered unsupervised approaches such as clustering, but the results were not satisfactory with the types of features we chose.

To perform the classification task, we employ random forest classifiers, which are very good for handling multiclass problems. Other choices, such as nearest neighbor classification, are possible. However, in general, random forest classifiers exhibit a better detection rate for these types of problems.

## 5. EXPERIMENTAL EVALUATION

In this Section, we report the experimental results of the `R-PackDroid` evaluation. Each application was disassembled with ApkTool[4], the most popular disassembler for Android. The smali output has then been parsed to extract the packages occurrences. To perform the machine learning analyses, we leveraged Scikit-Learn, a popular machine learning libary written in Python.

This Section will proceed as follows: we start by describing the dataset employed in our experiments. Then, we describe three experiments. In the first one, we evaluated the general performances of R-PackDroid. In the second one, we tested our system against novel data that are released after the data used to train the system. In the third one, we used R-PackDroid to analyze VirusTotal reports where applications were flagged as ransomware with very low confidence.

## 5.1 Dataset

In the following, we describe the dataset we used for our experiments, which includes data from the classes described in Section 4.

### *5.1.1 Ransomware*

The real world mobile ransomware samples examined in our experiments have been gathered from three datasets:

The first dataset, that we call HelDroid (Public) is a collection of freely available 672 samples[5] belonging to HelDroid dataset. The samples are generically labelled as ransomware [1] (i.e. the samples are not labelled with the family they belong to) and appeared from December 2014 to June 2015.

The second dataset is composed of 1350 new ransomware samples that have been distributed from the end of 2015 to November 2016. These ransomware samples are not publicly available and have been retrieved from the VirusTotal database. We call this dataset VirusTotal (Private).

The third dataset is composed by 23 samples retrieved from the VirusTotal service. All the samples have been flagged as ransomware by only *three* of the VirusTotal Antivirus. This small dataset will be very important for a practical evaluation that will be explained in detail in Section 5.4.

The hashes belonging to the second and third dataset can be found on the R-PackDroid website[6].

### *5.1.2 Malware and Trusted*

We consider a dataset composed of 5560 Android malware samples taken from the Drebin dataset, one of the most recent, publicly available datasets of malicious Android applications[7] (which also contains the samples from the Genome dataset [12]).

In order to download trusted applications, we crawled the Google Play market using an open-source crawler[8]. We obtained 4098 applications that belong to all the different categories available on the market. More specifically, they are among the most downloaded ones in each category and they are free. We chose the most popular apps in order to increase the probability of downloading malware-free apps.

## 5.2 Experiment 1: General Performances

In this experiment, we use the $F_1$ score as a metric to evaluate the performances of our system [9]. As we are considering a multiclass problem, the goal is evaluating the *average* $F_1$ score for each class, provided by the following:

$$F_{1_{avg}} = \frac{1}{|C|} \sum\nolimits_{c_i \in C} F_{1_{c_i}} \quad (1)$$

Where $C = \{c_1, c_2, ...c_n\}$ is the ensemble of classes. In our case, $|C| = 3$.

We used as dataset the HelDroid (Public) for the Android ransomware, the Drebin dataset for malware and 4098 trusted applications[9]. It is worth noting that, contrarily to other works that employed a lot of trusted applications, we chose a trusted set size that is similar to the one of malware and ransomware. This is because employing a lot of benign samples in the analysis could bias the general performances and provide misleading results [10].

The dataset was randomly divided into training and test sets. The size of the two sets are the same (50% training, 50% test). We repeated this process five times and we calculated the average of the various $F_{1_{avg}}$ that we obtained during these 5 runs. The random forest classifier was trained with a 10-fold cross validation, which led to a number of trees between 100 and 200, depending on the run. Table 1 shows the various $F_{1_{avg}}$ scores obtained for each split, along with their mean and standard deviation.

Table 1: General Performances - $F_{1_{avg}}$ score for each training/test split and the mean of the scores, along with the standard deviation (reported in brackets).

| **Split** | $F_{1_{avg}}$ |
|---|---|
| **1** | .97939 |
| **2** | .97822 |
| **3** | .97723 |
| **4** | .97664 |
| **5** | .97938 |
| **Mean (std)** | .97817 (.00111) |

From this Table, it is possible to see that R-PackDroid exhibits very good performances with high stability, and it is also able to effectively discriminate between generic malware and ransomware. This shows that, at least with respect to the system general performances, the system API can be reliably used to discriminate between applications.

## 5.3 Experiment 2: Performances on New Data

In this experiment, we examined the capabilities of our system to identify new ransomware. The rationale in this case is that, by using training samples released until a certain year, the system is able to correctly identify malware that are released *after* that year.

This experiment is made of two parts:

1. As training set, we used a variant of the dataset described in the previous experiment (Section 5.2). In particular, the set is composed by the entire Drebin dataset for generic malware, by 4098 trusted apps and by the 232 ransomware samples contained in the training dataset released by [1]. We tested our system on a

[4] https://ibotpeaches.github.io/Apktool/
[5] http://ransom.mobi/
[6] http://pralab.diee.unica.it/en/RPackDroid
[7] https://www.sec.cs.tu-bs.de/~danarp/drebin/
[8] https://github.com/liato/android-market-API-py

[9] Some files could not be analyzed by ApkTool: 1 from the HelDroid (Public) and 27 from Drebin.

dataset composed by 440 ransomware samples, which were used as a test set to assess HelDroid performances in [1]. This is done to verify if `R-PackDroid` performances are consistent with the ones of HelDroid.

2. We used as training set the entire dataset described in the first experiment (Section 5.2). The malicious samples contained in the training set were released before the end of 2015. In addition, we expanded the training set with 392 files of the `VirusTotal (Private)` dataset that were (see Section 5.1.1) released until January 2016. As a test set, we used the remaining 958 samples of the `VirusTotal (Private)` dataset that have been released between February and November 2016. To establish the release date, we referred to the date of first submission of the samples to the VirusTotal service.

We set R-PackDroid parameters by means of a 10-fold cross validation. As results can vary due to the folds that are randomly generated, we repeated this process 5 times. Tables 2 and 3 show the `R-PackDroid` results on the test sets, respectively, for part 1 and 2 of the experiment. In particular, for each run, we express the results in terms of number of samples that have been identified as ransomware, trusted and malware. We remind that the test sets for both tables are *uniquely* composed by ransomware.

Table 2: `R-PackDroid` predicted labels for the HelDroid ransomware test set (440 samples - HelDroid correctly predicted 375 samples). Standard deviation is reported in brackets.

| **Run** | **Predicted Class** | | |
|---|---|---|---|
| | **Ransom** | **Malware** | **Trusted** |
| **1** | 408 | 7 | 25 |
| **2** | 406 | 9 | 25 |
| **3** | 407 | 6 | 27 |
| **4** | 406 | 9 | 25 |
| **5** | 407 | 8 | 25 |
| **Mean (std)** | 407 (1) | 8 (1) | 25 (1) |

Table 3: `R-PackDroid` predicted labels for a test set of 958 novel ransomware samples released until November 2016. Standard deviation is reported in brackets.

| **Run** | **Predicted Class** | | |
|---|---|---|---|
| | **Ransom** | **Malware** | **Trusted** |
| **1** | 663 | 211 | 84 |
| **2** | 657 | 217 | 84 |
| **3** | 664 | 217 | 77 |
| **4** | 638 | 235 | 85 |
| **5** | 658 | 222 | 78 |
| **Mean (std)** | 656 (9) | 220 (8) | 81 (3) |

The results of Table 2 are consistent with the ones obtained by Andronio et al. in [1]. In particular, the authors claimed that that there were 14 files that were not correctly labelled by VirusTotal, but that were considered as ransomware in the test set. This is confirmed by our results. It is important to observe that we also consider as ransomware applications that redirect to an execution of a ransomware (that were considered as false negatives in [1]). We also observe that HelDroid could not detect 11 files due to their language properties. Our system is able to detect ransomware regardless of the application language. Overall, `R-PackDroid` was able to precisely detect almost all the tested ransomware.

The results of Table 3 show that `R-PackDroid` was able to effectively recognize more than 80% of the ransomware samples released in 2016, and to distinguish them from generic malware. It is interesting to observe that only a moderate percentage of files has been recognized as trusted. We interpret this result as an indication that, although a percentage of files might be recognized as generic malware, `R-PackDroid` rarely misclassifies malware as trusted files.

## 5.4 Experiment 3: Practical Use Case

In this experiment, we performed a *practical* analysis of the `R-PackDroid` capabilities. In the VirusTotal service, there are samples that are flagged as ransomware only by a very small number of Antivirus. The idea here was using `R-PackDroid` as a tool to confirm whether these samples were truly ransomware/malicious or false positives. As a decisive ground truth, we *manually* installed each sample that we tested with `R-PackDroid` to concretely verify if ransom-type operations (such as screen-lock) were actually performed.

We retrieved from the VirusTotal service all the samples that were detected as ransomware (i.e., with a ransom reference in the malware definition) by *exactly three Antivirus*. We found 23 samples holding this characteristic (the samples were retrieved on September 16th, 2016). We deliberately chose this number as it is very difficult to establish whether these files are truly ransomware or just false positives. As 22 files out of 23 were released in the second half of 2016, `R-PackDroid` was trained with the same training set used in the second part of the experiment described in Section 5.3.

To concretely verify the dangerousness of the samples, we installed each of them on an Android Emulator, which featured a Nexus 6 running Android Marshmallow (6.0). We used the Intel Atom (x86) instruction set to speed up emulation and the application execution. From our manual analysis of the samples, we found that 13 were actually ransomware, and 10 were (supposedly) benign.

`R-PackDroid` reported that 10 of the samples were ransomware, 3 of them were generic malware and the remaining 10 were trusted. Out of the 10 predicted ransomware samples, all were correctly detected. Most of these files contained ransom requests in Russian for approximately 15$, usually masked as porn viewers. Other ransomware samples directly showed a police lockscreen without resorting to fake applications to ask for more permissions.

Table 4 reports the MD5 hashes of the detected ransomware samples. At the date of submission of this paper (end of September 2016), only 3 of these samples were recognized by more than 3 Antivirus as ransomware. Two months later, the number of Antivirus that correctly detected the samples significantly increased for almost all the files. By using `R-PackDroid`, we were able to *precisely* anticipate the detection reports of most Antivirus in the wild.

3 files were flagged as malware by `R-PackDroid` (**a362cc6-5dd6adb662c2b6be8425c7550**, **adf25f78e74908aec72-d6685fad6fceb**, and **f29efc49f46320cf8f4233111506e8-cb**), and they are all real ransomware.

Finally, the reamaining ten samples were all flagged as

Table 4: List of the hashes of the ransomware correctly detected by `R-PackDroid`. In bold, we report the ones for which the VirusTotal detection rate was still 3 at the time of the paper submission.

| Ransomware Hashes |
|---|
| c28856f221f81a6dd217d7081fe3373f |
| 2eef47b61383591b8512eb6fdccb14f4 |
| f83ed56f56776ec083e0fd4a6794e148 |
| **1a62aebaf963d800dd44791699baac55** |
| **6e258502c69552b4f93dc0ec7ccdca45** |
| **d702a54aca8456c88a2e1bfffb303e1b** |
| **de4ebb490e150f4e1209ae3edd6d5709** |
| **2b2ea5ab2ed7221d3b4c233f7f3beaf4** |
| **5d25b6589e8d9eb13ea59bc92be1b0bf** |
| **e2525f4377573cb12eb9174f4a7d3ff6** |

trusted by `R-PackDroid`. Our manual analysis showed that they were Chinese shops and apps for watching online tv. Although all the applications did not exhibit behaviors that could be referred to malware, more in-depth static and dynamic analysis of the code should be performed to be 100% sure that those samples are legitimate (some samples might also employ anti-emulation techniques). Nevertheless, we can safely claim that these applications do not perform ransomware-like activities. Hence, we are inclined to say that the examined applications are real false positives. Note that `R-PackDroid` *never flagged as trusted any file that exhibit anomalies of any type.*

The complete list of the examined hashes can be found on the link provided in Section 5.1.1.

## 6. DISCUSSION AND LIMITATIONS

The results obtained with `R-PackDroid` show that it can be used as a lightweight, effective alternative to systems that require a lot of information in order to perform their detection (such as `HelDroid`). However, as `R-PackDroid` entirely resorts to static analysis, it is subject to some intrinsic limitations. For instance, it is not possible to analyze applications that feature code that is dynamically loaded at run time or class that are fully encrypted [6]. Such approach might therefore be integrated with other dynamic analysis techniques (such as the one proposed by [11]). The application analysis also depends on the parsing capabilities of the `ApkTool` framework. Hence, we discarded the applications that were not analyzed due to `ApkTool` crashes (see Section 5.2). In this paper, we did not analyze possible adversarial attacks to the machine learning algorithm, as the main idea of the paper was understanding if API packages could really help recognizing new ransomware samples. We plan to do this in future work, along with testing `R-PackDroid` against obfuscated applications to better assess its robustness.

## 7. CONCLUSIONS

In this paper, we proposed `R-PackDroid`, a supervised machine learning system for the detection of Android ransomware. Differently to previously proposed systems, our approach is lightweight and does not depend on a priori knowledge of the ransomware encryption capabilities. We showed that `R-PackDroid` is able to effectively discriminate between ransomware, generic malware and trusted files. In particular, `R-PackDroid` can detect novel ransomware by only using previously released training samples. Finally, we showed how `R-PackDroid` can be used to confirm reports from the VirusTotal service, thus detecting novel ransomware or false positives.

## Acknowledgements

This work has been partially supported by H2020 EU-funded project *European Network for Cyber Security* (NeCS), GA #675320 and EIT-Digital Project MCloudDaaS.

## 8. REFERENCES

[1] N. Andronio, S. Zanero, and F. Maggi. Heldroid: Dissecting and detecting mobile ransomware. In *RAID*, pages 382–404. Springer, 2015.

[2] M. Aresu, D. Ariu, M. Ahmadi, D. Maiorca, and G. Giacinto. Clustering android malware families by http traffic. In *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*, pages 128–135, Oct 2015.

[3] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck. Drebin: Effective and explainable detection of android malware in your pocket. In *NDSS*. The Internet Society, 2014.

[4] G. Canfora, F. Mercaldo, and C. A. Visaggio. An hmm and structural entropy based detector for android malware: An empirical study. *Computers & Security*, 61:1–18, 2016.

[5] J. Hoffmann, T. Rytilahti, D. Maiorca, M. Winandy, G. Giacinto, and T. Holz. Evaluating analysis tools for android apps: Status quo and robustness against obfuscation. In *CODASPY '16*, pages 139–141, New York, NY, USA, 2016. ACM.

[6] D. Maiorca, D. Ariu, I. Corona, M. Aresu, and G. Giacinto. Stealth attacks: An extended insight into the obfuscation effects on android malware. *Computers & Security*, 51:16–31, 2015.

[7] J. Oberheide, E. Cooke, and F. Jahanian. Cloudav: N-version antivirus in the network cloud. In *USENIX Security Symposium*, pages 91–106, 2008.

[8] J. Oberheide and C. Miller. Dissecting the android bouncer. *SummerCon2012, New York*, 2012.

[9] C. J. V. Rijsbergen. *Information Retrieval.* Butterworth-Heinemann, Newton, MA, USA, 2nd edition, 1979.

[10] S. Roy, J. DeLoach, Y. Li, N. Herndon, D. Caragea, X. Ou, V. P. Ranganath, H. Li, and N. Guevara. Experimental study with real-world data for android app security analysis using machine learning. In *ACSAC 2015*, pages 81–90. ACM, 2015.

[11] K. Tam, S. J. Khan, A. Fattori, and L. Cavallaro. Copperdroid: Automatic reconstruction of android malware behaviors. In *NDSS*. The Internet Society, 2015.

[12] Y. Zhou and X. Jiang. Dissecting android malware: Characterization and evolution. In *IEEE Symposium on Security and Privacy*, pages 95–109. IEEE, 2012.