Towards Large-Scale Hunting for Android Negative-Day Malware

Lun-Pin Yuan
Penn State University
lunpin@psu.edu

Wenjun Hu
Palo Alto Networks Inc.
whu@paloaltonetworks.com

Ting Yu
Qatar Computing Research Institute
tyu@hbku.edu.qa

Peng Liu
Penn State University
pliu@ist.psu.edu

Sencun Zhu
Penn State University
szhu@cse.psu.edu

Abstract

Android malware writers often utilize online malware scanners to check how well their malware can evade detection, and indeed we can find malware scan reports that were generated before the major outbreaks of such malware. If we could identify in-development malware before malware deployment, we would have developed effective defense mechanisms to prevent malware from causing devastating consequences. To this end, we propose Lshand to discover undiscovered malware before day zero, which we refer to as *negative-day* malware. The challenge includes scalability and the fact that malware writers would apply detection evasion techniques and submission anonymization techniques. Our approach is based on the observation that malware development is a continuous process and thus malware variants inevitably will share certain characteristics throughout its development process. Accordingly, Lshand clusters scan reports based on selective features and then performs further analysis on those seemingly benign apps that share similarity with malware variants. We implemented and evaluated Lshand with submissions to VirusTotal. Our results show that Lshand is capable of hunting down undiscovered malware in a large scale, and our manual analysis and a third-party scanner have confirmed our negative-day malware findings to be malware or grayware.

1 Introduction

Imagine, if you were an Android malware writer developing a zero-day malware, what would you do to check how well your malware can evade malware detection? One very convenient and convincing means is to anonymously submit the malware to online malware scanners and check the scan reports. In fact, it has been reported that some in-development malware were found on VirusTotal before their major outbreaks. For example, the very first known LeakerLocker sample could date back to November 2016 when it was submitted to VirusTotal [38], but not until July 2017 did security experts find it widespread. In addition, not just one malware sample but a trace of evolving malware samples did malware writers leave on online scanners during the ***continuous submit-and-revise process***. However, no signature was available on ***day zero*** (the time that such a threat is known to public) because the development case was not identified even though the malware was given malware labels at its early stage. Belated signatures could cause devastating consequences [33], and unfortunately it usually takes more than six months to generate well-crafted signatures [6]. If the ongoing development of malware had been identified and studied earlier, its breakout would not be as devastating as it had become. Therefore, our goal is to find Android malware before day zero, which we refer to as ***Android Negative-Day Malware***.

We propose ***Lshand*** (abbreviated from **L**arge **S**cale **H**unting for **A**ndroid **N**egative-**D**ays) to discover potential negative-day Android malware development cases (Neg-Day cases). Lshand faces the following challenges. First, with regard to binary, there is no obvious indication of malware development or kin relations among Neg-Day malware samples when detection-evasive techniques (e.g., dynamic-loading and obfuscation techniques such as [9, 14, 16, 19, 30, 41, 43, 44]) were applied to malware samples. Second, with regard to submitters, anonymization techniques may prevent us from inferring or linking the identities of submitters. We have observed many cases where a careful malware writer may submit malware samples from different anonymous identities (e.g., submitting samples from different sockpuppet accounts or with no account via free proxies or tor-network) and sign different malware samples with different keys. Third, with regard to scalability, online scanners constantly receive an enormous number of submissions (e.g., during January 2016, VirusTotal received 1.34M Android-related brand-new submissions that were never seen before). It is not practical to perform detailed analysis on each malware sample every time we observe a new one.

Lshand is designed carefully to overcome the above challenges. The key observation is that malware development is a continuous process, and malware variants will inevitably share certain characteristics throughout the development process.

Accordingly, our ***first principle*** is to process malware submissions based on similarity. However, as nowadays malware writers can easily leverage existing automated obfuscation tools, our ***second principle*** is to select features that are not likely to be obfuscated (e.g., set of permissions, contacted hosts, numbers of components). Finally, to be a practical solution, our third principle is that Lshand must be efficient, and is capapble of processing a large number of submissions within a reasonable time. Lshand, in a nutshell, consists of a *Data digestor*, a *Report clusterer*, an *AMDT* (Android Malware Development Trace) *extractor*, and a *Neg-Day alerter* (Fig. 1). To be brief, Lshand clusters malware reports (structural text format), and only if necessary it classifies malware samples (in binary) based on similarity and maliciousness.

We implemented and evaluated Lshand upon a snapshot of submission-stream towards VirusTotal during January 2016, which contains 1.3 million Android-related first-seen submissions from 3,852 different submitter identities (SIDs). Lshand, ran by a single thread on a desktop, took only an hour before giving us 10 Neg-Day cases. These 10 Neg-Day cases include 48 malware samples that were given no malware labels by any of the 62 scanner engines on VirusTotal by January 2016. In the end, according to submission rescans, our manual analysis, and scan results from Palo Alto Networks WildFire [29], we have confirmed that 100% of these 48 Neg-Day malware are actually malware. We also deployed Lshand over a more recent dataset dumped during May 2018, and Lshand hunted down 15 Neg-Day cases that include malware samples with zero malware labels. We have manually confirmed that 80% of these 15 Neg-Day cases are actually malware or grayware.

With regard to finding Android malware at their development stage, AMDHunter [17] is a strongly related work. However, unlike AMDHunter, which relies on the strong assumption that variants of the same malware development are submitted from the same SID, Lshand focuses on finding Neg-Day cases that were knowingly submitted from different accounts (e.g., sockpuppet accounts or no account), from different IPs (e.g., via proxies or via tor-network), or from decorated SIDs (e.g., knowingly submit benign apps in hope of confusing AMDHunter). Section 7 includes more details about the differences between Lshand, AMDHunter, and other related work. We make the following contributions.

- To the best of our knowledge, Lshand is the first malware hunting system that is capable of hunting down Android Neg-Day malware from multiple anonymous submitter identities through the analysis of submission records available on online scanners.

- We designed and implemented Lshand to overcome the following three challenges: lack of malware relations with regard to binaries, lack of development evidence with regard to identities, and scalability.

- We evaluated Lshand with two datasets. Lshand hunted down 10 Neg-Day cases from the submission records of VirusTotal during Jan. 2016 and 15 Neg-Day cases from records during May 2018. Our results show that Lshand is efficient and accurate.

2 Background and Motivating Example

Android malware detection has been heavily studied in recent years (e.g., [2, 3, 7, 10, 15, 20, 21, 23, 24, 27, 36, 37]). Although previous work has established significant detection capability, there are limitations when facing new challenges. Once a novel detection method is published, malware writers can knowingly evade the detection logic by leveraging, for example, code-loading (e.g., [30, 43]) and sandbox detection (e.g., [26]). Furthermore, malware writers can use malware-development tools (e.g., [19]) to embed detection-evasive modules. Wei et al. [41] and Suarez-Tangil et al. [35] detailed the trend of malware evolution and evasive techniques. Other related work regarding malware development includes [9, 14, 16, 40, 44].

We consider LeakerLocker [31] as a motivating example. Only after it sleeps for a while and checks non-sandbox artifacts, it loads dynamic code. Hence, its first VirusTotal report [38] only had one malware label from one scan engine. Fig. 2 illustrates an example of what information about LeakerLocker one can obtain from a VirusTotal scan report. The information in Fig. 2 is open to public except the submitter identity, which is only available to VirusTotal premium users.

While researchers use online malware scanners as partial ground truth (e.g., [34, 41]), malware writers also use them as convenient and convincing oracles to test malware variants and evasive techniques (e.g., [19]). Via public-API or web interface, a malware writer can submit up to four malware samples per minute to VirusTotal (exceeding this rate will cause empty JSON or reCAPTCHA depends on channel). It has been reported that a set of in-development malware were found on VirusTotal; for example, the very first LeakerLocker sample dated back to November 2016 [38], but its outbreak did not happen until August 2017.

In practice, Android malware writers often repackage a popular app or a handy utility app in order to lure innocent users to install or distribute the malware. In LeakerLocker's example, some variants impersonate Call Recorder, and others impersonate WallPapers HD Blur. Consequently, each group demonstrates a set of shared characteristics (e.g., Call Recorder variants share 16 permissions, 13 activities, and 3 services, and Wall Papers HD Blur variants share 18 permissions, 8 activities, and 3 services). Based on this observation, Lshand hunts malware development traces by examining shared characteristics.

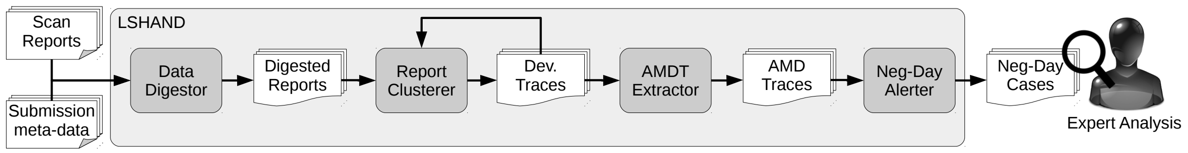

Figure 1: LSHAND Workflow Overview

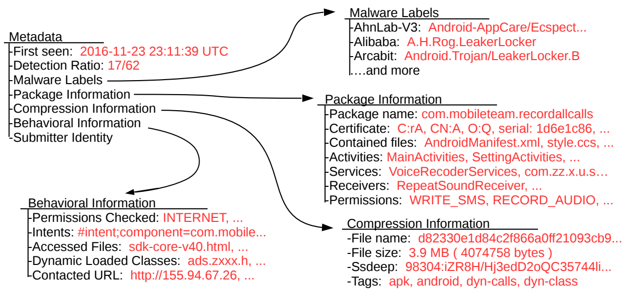
Metadata
-First seen: 2016-11-23 23:11:39 UTC
-Detection Ratio: 17/62
-Malware Labels
-Package Information
-Compression Information
-Behavioral Information
-Submitter Identity
Malware Labels
-AhnLab-V3: Android-AppCare/Ecspect...
-Alibaba: A.H.Rog.LeakerLocker
-Arcabit: Android.Trojan/LeakerLocker.B
....and more
Package Information
-Package name: com.mobileteam.recordallcalls
-Certificate: C:rA, CN:A, O:Q, serial: 1d6e1c86, ...
-Contained files: AndroidManifest.xml, style.ccs, ...
-Activities: MainActivities, SettingActivities, ...
-Services: VoiceRecoderServices, com.zz.x.u.s...
-Receivers: RepeatSoundReceiver, ...
-Permissions: WRITE_SMS, RECORD_AUDIO, ...
Behavioral Information
-Permissions Checked: INTERNET, ...
-Intents: #intent;component=com.mobile...
-Accessed Files: sdk-core-v40.html, ...
-Dynamic Loaded Classes: ads.zxxx.h, ...
-Contacted URL: http://155.94.67.26, ...
Compression Information
-File name: d82330e1d84c2f866a0ff21093cb9...
-File size: 3.9 MB (4074758 bytes)
-Ssdeep: 98304:iZR8H/Hj3edD2oQC35744li...
-Tags: apk, android, dyn-calls, dyn-class

Figure 2: A Digested Report for a LeakerLocker variant

3 Problem Statement

We are specifically interested in answering the following research question: how can we find potential Android Neg-Day malware in a large scale from online malware scanners? The challenges include the followings. First, with regard to malware binaries, there is no obvious indication of malware development or kin relations among or between Neg-Day malware samples. Second, with regard to submitters, there may not be evidence such as submitter identity that can help us with inferring or linking malware writers. Third, it is not realistic to perform detail analysis on every newly received malware samples submitted to an online scanner.

3.1 Attack Model

Our attack model includes a careful malware writer who does not want to leave obvious traces but does need online malware scanners for up-to-date analysis of a newly developed malware. We assume that a careful writer has two objectives ***detection evasion*** and ***submission anonymization***, and we assume that the malware writer will keep continuing the development process in a ***submit-and-revise*** fashion.

Regarding detection evasion, we assume that a malware writer would apply techniques such as package name obfuscation, class name obfuscation, method and variable name obfuscation, inter-component interaction injection, dataflow-analysis evasive code injection, native code and bytecode injection, dummy and benign methods injection, and dropper payload loading. However, trying to be less suspicious just like a regular app, evasion techniques such as *incrementally* adding *large number* of dummy Activities, dummy Services, and irrelevant files on the malware are not considered (in other words, numbers of activities, services, and files stay similar).

Regarding submission anonymization, we assume that a malware writer would submit malware samples anonymously from different free accounts (or no account) via different free proxies (or tor-network) throughout the continuous development process. In addition, a malware writer can submit many irrelevant apps in hope of decorating his profile as if the submitter was benign. However, trying to keep a low profile, we assume that a malware writer will not submit more than on-average 100 newly developed samples per-day, per-account, and per-proxy. We will justify in Section 5 that this assumption is realistic by submission statistics.

3.2 Definitions

Since Neg-Day hunting is a relatively new research topic, we would like to define terms to avoid confusion.

Digested Report: The outcome of preprocessing a scan report from online malware scanner is a digested report, and it includes only necessary fields, including submission timestamps, submitter identity, malware labels, package information, compression information, and behavioral information (malware sample binary is not included). Fig. 2 shows an digested report example.

Submitter Identity (SID): An SID is a unique identifier that specifies a submitting individual or a submitting organization; however, an individual or organization may correspond to one or more SIDs. SID on VirusTotal is a hash value calculated based on account profile, or based on IP-port profile if no account is presented. In most online malware scanners, SID information is available only to premium users.

Malware Label: In a malware scan report, each scan engine may list multiple informative labels, including malware family (e.g., leakerlocker, svpeng, slocker), malicious functionality (e.g., ransom, trojan, dropper), suspiciousness (e.g., dangerous, high confidence malware, potentially unsafe), non-malicious characteristics (e.g., adware, riskware, potentially unwanted), and some other information (e.g., downloader, obfuscated, xor-crypt).

Development Trace (DT): A DT implies a development of an app. A DT is represented by a chronological sequence of digested reports that share certain features. A DT may include digested reports that are sourced from different SIDs. We say two samples are "***kins***" to each other if they were put into the same DT. Unlike variants of a malware family (e.g., LeakerLocker), if there are two different fake apps (e.g., Call Recorder and WallPaper HD Blur), we consider them as two different DTs.

Android Malware DT (AMDT): An AMDT implies a development of an Android malware. Essentially, an AMDT is a DT; however, unlike DTs, an AMDT includes at least one ***malware indicator***, a digested report that has a sufficient number of malware labels. Based on needs, one can define different sufficiency metrics; in this paper, we use detection ratio r (i.e., the ratio of engines that reported malware labels).

Negative-Day AMDT case (Neg-Day case): A Neg-Day case implies a potential development case of a Neg-Day malware. Essentially, a Neg-Day case is an AMDT; however, unlike AMDTs, a Neg-Day case includes at least one recent submission that has no malware labels (i.e., $r = 0$). Since this seemingly benign sample has evaded detection from all scanning engines, terrible harm could be done once it is distributed. Hence, detecting such a Neg-Day case with benign samples is our goal.

4 LSHAND Design

4.1 Design Overview

Lshand's design is based on our key observation that malware writers tend to test their in-development malware through online malware scanners. We assume that a malware development is a continuous process; hence, malware writers inevitably will leave development traces online, and malware variation will demonstrate certain shared or similar characteristics. However, finding AMDT is challenging because malware writers could always apply evasion and anonymization techniques. Hence, our design follows the following principles: first, we process malware submission based on similarity; second, we select features that are not likely to be obfuscated. For scalability reason, our third principle is that we analyze scan reports (specifically, digested reports), and only if necessary we analyze malware binaries.

Lshand, in a nutshell, consists of a *Data digestor*, a *Report clusterer*, an *AMDT extractor*, and *Neg-Day alerter* (Fig. 1). Lshand's workflow is as follows. Lshand's Data digestor looks for Android-related first-seen submissions, and then produces digested reports. For each digested report, Lshand's Report clusterer deduces its DTs. From DTs, Lshand's AMDT extractor extracts AMDTs. From AMDTs, Lshand's Neg-Day alerter verifies potential Neg-Day cases and raises alarms. In our design, report clusterer is the module that identifies *kin* relation among malware samples, and Neg-Day alerter is the module that identifies the development evidence.

4.2 Data Digestor

Data digestor finds Android-related first-seen submissions and generates corresponding digested reports. By first-seen submissions, we mean those submissions whose samples were never processed by the selected online malware scanner. We only focus on first-seen submissions, because re-submissions cannot imply malware development.

Data digestor learns whether or not a submission is Android-related by checking malware labels (if there is any "android" or android-related label), package information (if there is any Android application components), and compression information (if there is any "apk", "dex", or "android" tag). Data digestor learns whether or not a submission is first-seen by checking "last-seen" or "last-scan" timestamps in the submission metadata (both should be slightly greater or equal to the submission timestamps), or by checking the existence of previous scan reports (reports with the same file hash should not exist). Upon getting an Android-related first-seen submission, Data digestor parses the output information and produces a digested report (as shown in Fig. 2).

4.3 Report Clusterer

For each digested report, Report clusterer finds the corresponding DTs by clustering digested reports along with existing DTs. We say two samples are *kins* if two samples belong to the same DT. This approach is based on our observation that kin malware variants often bear some similarity or characteristics even though some contents could be obfuscated. Here we detail our design choices, including feature extraction and clustering scheme.

4.3.1 Feature Extraction

We cannot directly use the informative knowledge in the digested reports, because malware writers may have applied obfuscation. Rather, Lshand extracts features that are less likely to be obfuscated or manipulated. We split our features into the following categories: submission timestamps, package information, compression information, and behavioral information. Note that Lshand does not take malware labels as part of the features, because Neg-Day submissions may receive no labels whatsoever.

Submission Timestamps: We consider malware development a continuous submit-and-revise process, hence the chronological gap between two kin submissions will not be large. If two similar submissions are faraway apart in time, then they are less likely to be developed by the same malware writer, as it is more likely to be a new development case of different malware writers who are trying to gain benefits from a previously known malware. Submission timestamps are issued by online malware scanner, so malware writers cannot manipulate them (note that file and certificate timestamps can be manipulated to such as 1980-00-00 and 2107-15-19).

Package Information: Lshand examines structural information and permission information of a package. For structural information, although class names and file names can be obfuscated, we can still count the numbers of Activities, Services, and other files by types (we consider 26 different file types, including mp3, png, and dex). Hence, Lshand has $1+1+26=28$ numerical features. Lshand will not be affected by name obfuscation because names are not used as features. As for permissions, besides predefined permissions in Android framework, developers can define app-specific or device-specific permissions. Upon 399.9K digested reports, we split 22K known permissions into 56 categories (e.g., network, storage, camera, c2dm). For each category, Lshand counts the number of related permissions listed in the digested reports. In summary, Lshand has $28+56=84$ package-related features in total.

Compression Information: Lshand examines file tags and uncompressed file size of a submitted sample. Most online malware scanners provide tags for file types (e.g., "apk", "jar", "android") and tags for characteristics (e.g., "dyn-class", "via-tor", "check-gps"). Lshand considers 21 tags as features, where each feature is represented by either true or false whether or not the tag appears in the digested report. As for uncompressed file size, Lshand treats it as one feature (hence $21+1$ features in total). Uncompressed file size is essential because simply using package-related features and tags is not enough to accurately deduce DTs. Our experiments showed that, without uncompressed file size, apps with similar structural information and similar file tags will be wrongly put together. With uncompressed file size, in contrast, we have had better clustering results. Since uncompressed size does not have impact on malware scanners' judgement, it is often not malware writers' concern to obfuscate. Therefore, uncompressed size often stays similar from one malware kin to another malware during a development.

Behavioral Information: Contacted URLs, accessed files, sent SMS, and inter-process communication could be available in the digested reports if the selected online malware scanner(s) provides sandbox analysis. While most strings can be obfuscated in the package, certain strings cannot be obfuscated at runtime. For example, contacted hosts (domain and IPs) and inter-app intents (actions and components) cannot be obfuscated (but parameters and contents can still be obfuscated). Hence, Lshand extracts contacted hostnames and IP from HTTP(s) communication and intented actions and components from intent communication. There are many ways to represent strings in the feature space. Although it is an option to apply complex string comparison methods which are accurate but slow, Lshand simply counts alphanumeric (i.e., a-z and 0-9) n-grams, where $n=1$ in our current implementation; for example, *"google.com"* has numeric feature "3" for the character *"o"*. Lshand has 36 features for contacted hosts and for inter-app intents; therefore $36+36=72$ behavioral features in total. Note that it is also optional to use $(n>1)$-grams for a more fine-grained feature set.

4.3.2 Clustering Scheme

In order to produce accurate results, we carefully studied clustering models, weighted features, and distance thresholds. However, note that since our ultimate goal is to hunt down Neg-Day cases, we tune Report clusterer with only AMDTs rather than DTs in general; whether or not a benign DT is accurate is not a major concern.

Clustering Model: We consider malware development a continuous process, so AMDTs will demonstrate progressive evolution of a malware. As such, Lshand leverages incremental density-based clustering model. Under this clustering model, clusters are formed by merging observations (feature-based representation of digested reports), and the linkage distance between two clusters is defined by the euclidean distance between the two nearest observations (single-linkage). If the linkage distance between two clusters is smaller than a given quality threshold τ, then the two clusters will be merged into one. We do not consider the other linkage approaches (e.g., complete linkage and average linkage) because they are not suitable for progressive evolution (where a malware variant demonstrates similarity with a few previous variants rather than with the entire development), and we do not consider k-clustering model because we do not know how many DTs are out there. There are a few density-based clustering models that are applicable to our research problem, but for scalability reason we implemented an incremental density-based clustering model, whose time complexity is $O(n\times(m+n))$, where n is the number of new observations, and m is the number of old observations (hence $O\left(n^2\right)$ from scratch or $O(m)$ for an incremental step). Discussion on optimizing clustering algorithm for malware analysis can be found in [4, 8, 32].

Weighted Features: We assign different weights to features because different features could vary in different scale throughout its development. For example, features in package information (e.g., number of Activities and permissions) shall be very similar if not the same, whereas features in submission timestamps and features in compression information (e.g., uncompressed size and tags) can be different every time. The resulting DTs will be problematic if Lshand treats these features equally. Hence, Lshand puts less weight on features in timestamps and in compression information, but more weight on features in package information (Table 1). Another factor of assigning weights to features is how thorough the analysis can be provided by the selected online scanners (for example, we put light weights on behavioral features because VirusTotal does not always apply behavioral analysis to Android submissions). To determine weights, we collected 50 confirmed AMDTs and conducted an empirical study. We repeatedly tuned weights and ran experiments until we reached a state

Table 1: Weighted Features

Category	Feature	#	Weight
Timestamps	submission timestamps	1	medium
Package	# of Activities	1	heavy
Package	# of Services	1	heavy
Package	# of file by types	26	medium
Package	# of Permissions by categories	56	heavy
Compression	file tags and labels	21	medium
Compression	uncompressed size	1	medium
Behavioral	contacted URLs	36	light
Behavioral	inter-app communication	36	light

where AMDTs were formed only by true kin observations and as many observations in each AMDT as possible.

Clustering Threshold: Quality threshold τ is an important parameter in density-based clustering models (some may use the term density distance ε). If τ is too large, a cluster would wrongly include outsider observations that otherwise should not be included, and if τ is too small, a cluster would not include kin observation that otherwise should be included (or equivalently, split one AMDT into smaller AMDTs). In this research, we consider having outsider observations much more destructive than having one AMDT split into smaller AMDTs. The reason is that our ultimate goal is to provide malware analysts with Neg-Day cases for comprehensive analysis so that they can craft signatures that can detect undiscovered malware; however, contrary to our goal, outsider observations could cost malware analysts more time to conduct inaccurate analysis. To determine τ, we conducted an empirical study upon 50 confirmed AMDTs. We first set a large τ and then tuned τ down until we reached a state where we have no outsider observations.

4.4 AMDT Extractor

Lshand extracts AMDTs from DTs. However, AMDT extractor is not just a noise filter. Besides extracting AMDTs, it also updates digested reports in seemingly benign DTs, and it also removes digested reports that are no longer useful in AMDTs.

Extract AMDTs from DTs: An AMDT essentially is a DT with at least one *malware indicator*, a digested report that has sufficient number of malware labels. Depending on label availability from the selected online malware scanner, one may opt to use different metrics for determining malware indicators. In our implementation, Lshand simply checks detection ratio r (e.g., 30 out of 62 engines) rather than analyze label semantics (e.g., differentiate “malware” and “riskware”). Checking detection ratio r has an advantage that it does not require prior knowledge of existing labels, hence it is generic to engine-specific labels and flexible to newly-defined labels. It is a common practice to consider a submission suspicious when there are many engines reported it with malicious or suspicious labels regardless of label semantics (note that the LeakerLocker example [38] was given only suspicious labels and no malicious labels).

Update digested reports in benign DTs: Previously seemingly benign digested reports could become malware indicators once the scan engines are patched, and once a digested report becomes a malware indicator, a DT becomes an AMDT. Hence, for each seemingly benign DT, Lshand needs to submit rescan requests for those seemingly benign samples that do not have recent digested reports. Note that updating malware labels will not merge or split DTs, because Lshand does not consider malware labels as clustering features.

Remove digested reports from AMDTs: The growth in the number of digested reports is enormous, and it will not be scalable if Lshand keeps all the digested reports and process all of them repeatedly. However, Lshand cannot simply discard *ancient* digested reports. By ancient, we mean that the submission timestamp of a digested report is so far away from current timestamp that the distance between two weighted timestamp features has exceeded the quality threshold τ (in other words, ancient digested report will no longer affect clustering results). We cannot simply discard ancient digested reports because ancient digested reports could become malware indicators someday, and discarding malware indicators may cause AMDTs to become DTs. Rather, Lshand discards two kinds of digested reports: first, ancient digested reports that are prior to any malware indicator in the same AMDT (hence no AMDT will become DT); second, digested reports in an ancient DT that will not be clustered with any new digested reports.

4.5 Neg-Day Alerter

Neg-Day alerter verifies Neg-Day cases by examine *maliciousness* and *similarity*. On one hand, Lshand skips those AMDTs that are already known to be malicious, because we are particularly interested in Neg-Day cases that nobody knows they are malicious. On the other hand, Lshand looks for package-level similarity as evidence of development (even though some correlation can be derived from digested reports, certificate and binary and GUI correlation are not yet assured). In the end, if a sample from an AMDT passes both maliciousness test and similarity test (that is, not malicious but similar), Lshand will raise a Neg-Day alert to malware analysts with a Neg-Day case. Note that Neg-Day alerter only checks binary only when necessary.

Maliciousness Test: Since we are interested in Neg-Day samples that nobody knows they are malicious, only when a sample is not obviously malicious will Lshand proceed to the next step. To examine the maliciousness of a new sample in an AMDT, Lshand checks reports, signatures, and binaries: first, Lshand examines the digested report and will proceed if there are very few malware labels (e.g., $r = 0$); second, Lshand examines its certificate and will proceed if the certificate is not publicly known to be benign (white-list certificates);

third, Lshand checks its binaries by using third-party binary-level Android malware classifiers (e.g., MaMaDroid [24] and Drebin [3]), and will proceed if the results are benign (specifically, Lshand trains classifiers with publicly known malware and publicly known benign apps). If a sample passes all the above tests, Lshand will then test whether the sample is similar to its AMDT.

Similarity Test: Different from Report clusterer which compares feature-space similarity, here we compare signature and binary similarity among malware samples within an AMDT. If a new sample does not show similarity at this stage (hence it is not a new variant but an outsider who coincidentally has similar features), then Lshand needs to label it as an outsider and remove it from the AMDT. We say a new sample has passed the similarity test if *any* of the following comparisons show similarity: in certificate comparison Lshand checks whether or not the information in the signing certificate (e.g., public key and serial number) is the same as that in the other samples in the AMDT, and in binary comparison Lshand checks whether or not the control call graph or GUI-callback methods has similar characteristics to the other samples in the AMDT by using repackaging classifiers (e.g., ViewDroid [45]) and malware classifiers (e.g., MaMaDroid [24] and Drebin [3]). Specifically, for each AMDT Lshand trains the classifier with *malware indicators of the AMDT* and publicly known benign apps, and Lshand considers a sample to be similar to its AMDT if the classifier reports the sample *"malicious"*, which is the case that the sample is closer to the malware indicators than to the other irrelevant apps in the feature space (different from the maliciousness test where we compare binary features with known malware).

5 Accuracy Evaluation

5.1 Dataset and DT Statistics

Our evaluation was conducted upon submissions to VirusTotal. Since neither do we nor VirusTotal have a solid ground truth for recent submissions, we test Lshand upon old submission-stream captured during January 2016 (detection ratio is denoted as r_u), but then we evaluate Lshand with the re-scan results that were produced during May 2018 (detection ratio is denoted as r'_u). Even after two years, we consider these re-secan results as *partial* ground truth rather than complete ground truth because it is possible that some malicious samples are still not discovered. During January 2016, VirusTotal received 1,345K Android-related first-seen submissions from 3,852 different submitter identities (SIDs). The HTML-based reports totally used 154 GB storage space, whereas the digested reports took only 8.3 GB.

Subset Selection: We do not need to consider every single submission, as we can exclude premium users that are not anonymous. Table 2 and 3 show the statistics of SID-to-peak-rate and SID-to-monthly-rate. If we consider terminal-based

Table 2: SIDs and Peak Rate

Peak Rate	$\leq$ 4/min	$\leq$ 60/min	$\leq$ 120/min	Unlimited
# of SIDs	3,364	3,750	3,788	3,852
# of subs	5,318	14,309	19,837	1,345,696

Table 3: Clusterng Statistics

Monthly Rate (ρ)	$\leq$ 3100	$\leq$ 12400	$\leq$ 24800	Unlimited
# of SIDs	3,837	3,841	3,843	3,852
# of subs	58,765	92,603	125,267	1,345,256
# of observ.	21,800	49,205	62,773	371,187
# of clusters	14,266	32,986	38,236	85,139
Time Used	01:01:36	4:59:58	07:51:07	291:28:12

Note that the difference between the number of submissions and the number of observations comes from the fact that our Lshand evaluation does not consider submissions that have missing information and submissions that have a small number of application components (i.e., < 5 activities and services counted together).

non-premium submissions (i.e., peak rate $\leq$ 4/min), then we only need to worry about 5,318 submissions from 3,364 SIDs. Nevertheless, to test the scalability of our approach, we relax this constraint to submission peak rate under 120 submissions per minute (19,837 submissions from 3,788 SIDs), or alternatively 3,100 submissions per month (58,765 submissions from 3,837 SIDs) assuming malware writers will keep their SIDs low profile. In our dataset, the latter subset is a superset of the former subset, and hence we consider the submission-subset of SIDs that have no greater than 3,100 submission per month (denoted as $\rho \leq 3100$) a sufficiently large hunting ground. This statistics justifies our assumption in Section 3 that a malware writer will not submit more than on-average 100 new malware samples per-day, per-account, and per-proxy.

DT Statistics: Table 3 also shows the time usage from scratch (there was no pre-built cluster, and clusters were built up incrementally on a single thread). For the recommended dataset-subset of $\rho \leq 3{,}100$, Lshand spent an hour in clustering DTs on a single thread. Although we state that the submission-subset of $\rho \leq 3{,}100$ is sufficient, our approach is certainly computational feasible to higher rates (a more scalable alternative approach is discussed in Section 6). Table 4 shows the resulting DT statistics. We can see that 1,005 DTs out of 14,226 DTs were submitted from multiple SIDs.

Table 4: Development Trace Statistics

Intervals	# of DTs to [a,b) of SIDs	# of DTs to [a,b) of reports
1	13,261	12,650
[2, 10)	983	1,499
$[10, 10^2)$	19	108
$[10^2, 10^3)$	3	9
MAX	194	562

5.2 AMDT Accuracy Analysis

Lshand extracted AMDTs from the recommended submission-subset of $\rho \le 3{,}100$. Each AMDT has at least five digested reports and at least one malware-indicating digeseted report u that has detection ratio $r'_u \ge 30/62$ by May 2018. The reason why we did not evaluate low detection-ratio DTs is twofold: first, we are focusing on checking AMDT rather than clusters in general; second, we have better partial ground truth when r'_u is higher.

Baseline Selection: We compare Lshand AMDTs with AMDTs from Ssdeep-based clustering method SSDC [39], because SSDC is similar to our work in the way that it requires only the metadata to produce AMDTs. Ssdeep has been leveraged in several work that aims to cluster malware (e.g., SSDC and Graziano et. al [13]). These work requires only the Ssdeep values available in VirusTotal scan reports: two samples that have homologies will have common byte sequences in their Ssdeep values, and two Ssdeep values can imply how homologous the two corresponding samples are.

Oracle Setup: We built an oracle for comparing AMDTs of different approaches in binary-level. The oracle consists of Android malware classifers MaMaDroid [24] and Drebin reimplementation [28], and it considers a sample to be ***affiliated*** to its AMDT if and only if (1) the sample is similar to the other samples in the same AMDT, and (2) the sample is verified malicious by recent partial ground truth we gathered by May 2018 (unlike Neg-Day alerter which looks for benign samples using partial ground truth by Jan. 2016). For each AMDT, the oracle gives us a correlation score S_i calculated with Eq. 1. The training details is stated as follows. We carefully verified an ***outsider malware set*** and an ***outsider benign set***, where malware set includes 362 malware ($r'_u \ge 30/62$) that each has few kins ($|kins| \le 2$), and benign set includes 631 apps ($r'_u = 0/62$) that each has few kins ($|kins| \le 2$). For similarity test upon an AMDT, we split AMDT samples into two halves based on the detection ratio r'_u, and we trained the classifiers with the high-ratio half and *outsider benign set* to verify the low-ratio half. For maliciousness test upon an AMDT, we trained the classifiers with *outsider malware set* and *outsider benign set* to verify the samples. Note that, we tried to include some other tools, but each of them failed to dissect a large portion of our dataset, either because its design was not for the latter Android app framework (e.g., ViewDroid [45]), or because it took too much time and eventually timed out (e.g., Asteroid [11]).

$$\text{Correlation score } S_i = \frac{\text{\# of affiliated kin samples}}{\text{\# of samples}} \tag{1}$$

$$\text{cell}_{xy} = \begin{bmatrix} \dfrac{\text{\# of samples in the correlation level } y}{\text{\# of all samples in column } x} \\ \\ \dfrac{\text{\# of AMDTs in the correlation level } y}{\text{\# of all AMDTs in column } x} \end{bmatrix} \tag{2}$$

Table 5: Accuracy of different AMDT sets

$\rho = 3{,}100$	SSDC AMDT	Lshand AMDT	Lshand Neg-Day
# of samples	1,532	1,698	253
# of AMDTs	68	56	10
Perfect	85.51%	99.16%	100.00%
(100%)	79.41%	96.43%	100.00%
Excellent	3.59%	0.00%	0.00%
(90-100%)	2.94%	0.00%	0.00%
Good	2.81%	0.00%	0.00%
(80-90%)	4.41%	0.00%	0.00%
Fair	0.00%	0.00%	0.00%
(70-80%)	0.00%	0.00%	0.00%
Problematic	0.00%	0.48%	0.00%
(60-70%)	0.00%	1.79%	0.00%
Bad	8.09%	0.36%*	0.00%
(0-60%)	13.24%	1.79%*	0.00%
hours per revision	168.84	207.81	16.26

* False negatives were mistakenly introduced by the oracle; we have verified this cluster a perfect cluster. Hour-per-revision is calculated based on the highest file-timestamp in an apk.

Table 6: SID Statistics of Neg-Day Cases

Neg-Days	Jan 2016	May 2018
# of samples	253	256
# of DTs	10	15
# of DTs with 1 SID	1	3
# of DTs with [2, 10) SIDs	8	5
# of DTs with [10, ∞) SIDs	1	7
Maximum # of SIDs	25	41

AMDT Results: Table 5 shows our accuracy evaluation. For presentation purpose, we split the correlation scores into six levels (i.e., perfect, excellent, good, fair, problematic, and bad), and each table cell is calculated with Eq. 2 (i.e., upper scores represent the percentage among all samples, whereas lower scores represent the percentage among all AMDTs). For example, an AMDT with five affiliated kins and three unaffiliated samples has correlation score $5/8 = 62.5\%$, and hence this AMDT is *Problematic*. This one AMDT (out of 56) with eight samples (out of 1,698) contributes to the *Problematic* cell 0.48%-1.79% in Table 5. We can see that Lshand outperforms SSDC in terms of providing accurate AMDTs: not only did Lshand provide more accurate AMDTs (99.16% vs 85.51%), but also it did provide more samples in total (1,698 vs 1,532). In addition, Lshand produced less *Bad* AMDTs. Note that our AMDT results were given by AMDT extractor before attested by Neg-Day alerter.

5.3 Hunting Negative-Day Malware

Our prey is Neg-Day cases, which are AMDTs with seemingly benign submissions. We evaluate Lshand with an old submission-stream snapshots, but we also deploy Lshand for a more recent submission-stream snapshot.

5.3.1 Submission-stream of January 2016

Lshand hunted down 10 Neg-Day cases from the submission-subset where $\rho \leq 3{,}100$ subs/month. Each Neg-Day case includes at least one seemingly benign digested report u that has detection ratio $r_u = 0/62$, at least one malware-indicating digested report v that has $r_v \geq 3/62$, and at least five samples in total. Table 6 shows that nine (out of 10) Neg-Day cases were submitted from multiple SIDs. Among these 10 Neg-Day cases, Lshand identified 49 potential Neg-Day Android malware ($r_u = 0/62$ by Jan 2016). In addition, 96 (out of 253) samples have the tag *"xorcrypt"* (sample is xor-encrypted), and 55 samples have obfuscated class name.

Table 5 shows our Neg-Day accuracy evaluation. A score of 100% is attested by the same oracle that was trained by the selective outsider datasets by May 2018. To be more convincing, we also manually verified these Neg-Day cases. We submitted rescan requests to VirusTotal, studied the rescan reports, and verified potential AMDT evidences (e.g., malware labels, Ssdeep values, certificates, permissions, components, and names if not obfuscated). Our manual verification aligns with our oracle's evaluation: every Neg-Day cases are indeed homologous and malicious. All 48 potential Neg-Day malware ($r_u = 0/62$ by Jan. 2016) have become malicious ($r'_u > 0/62$ by May 2018). The average detection ratio for these 253 submissions was $\bar{r}_u = 4.31/62$ by Jan. 2016 and has become $\bar{r}'_u = 32.06/62$ by May 2018. Malware labels for these 10 Neg-Day cases include Dnotua, Dowgin, Ewind, Huer, Jiagu, Rootnik, SmsPay, SmsReg, and Triada. Based on this evaluation, we conclude that Lshand is capable of hunting down Neg-Day Android malware. Yet, note that since we do not have ground truth for false negatives, we think that there still could be some undiscovered Neg-Day cases in DTs.

5.3.2 Submission-stream of May 2018

We also deployed Lshand for more recent data which was captured during May 2018. We fed Lshand's report clusterer the same features, weights, and thresholds; however, we fed Lshand's AMDT extractor a different set of parameters in order to narrow down possible Neg-Day cases due to the limited computational and personnel resource we have.

During May 2018, Lshand hunted down 15 Neg-Day cases with totally 256 samples (114 samples have $r'_u = 0/62$ by May 2018). Table 6 shows that 12 out of 15 Neg-Day cases were submitted from multiple SIDs. We asked the same oracle for automatic judgment and got 96.77% similarity score but 31.25% maliciousness score. Without recent data in the training set, the evaluation oracle failed to judge maliciousness due to concept drift [1, 18]. Hence, we manually analyzed the latest sample in each Neg-Day cases that were reported benign ($r'_u = 0/62$).

In our manual analysis, we submitted 15 samples to Palo Alto Networks WildFire [29] for behavior scanning, and we checked samples with JEB decompiler (it is possible that we missed the malicious part in some cases). By the time we wrote this paper, we have confirmed that 12 out of 15 Neg-Day cases (80%) are malware, but VirusTotal and 62 engines failed to identify these variants. Their malware labels include SLocker, Triada, Trojan, riskware, downloader, and potentially unwanted. One may argue some listed labels are not truely malicious, but please recall that the maliciousness of a Neg-Day malware is often underestimated (e.g., LeakerLocker had only one suspicious label [38]), and it is not Lshand's job but malware analysts' job to tell its true maliciousness.

5.3.3 False-Positive Case Study

Lshand relies on the assumptions that malware development is a continuous submit-and-revise process. However, it is not always true. In the results of May 2018, we got two (out of 15 Neg-Day cases) inevitable false positive cases that violate our assumptions, and both cases are apps (54 apps in total) created by Appbyme [5], an online DIY app-creation platform for non-programmer users with a variety of templates and resources. We consider them false positive cases because their latest kins are not actually malicious. These two cases show that Lshand will be inaccurate in judging apps from such a platform.

The reason that Lshand reported Appbyme is threefold: first, the after-creation apps were structurally similar (they share the same set of 81 activities, 9 services, 28 permissions, and they all contains more than 1700 files) because of similar selections on templates and resources even though these apps were from different non-programmer users; second, one of their malware-indicating kin is malicious (labels include Adware, Malware-HighConfidence, PUA, and TencentProtect, an anti-cheat system monitor); third, the latest kin seems not malicious. For confirmation, we submitted these malware indicators to Palo Alto Networks WildFire, and the results are that the malware indicators are indeed malicious, but the latest benign apps are indeed benign. In Lshand's point of view, the latest benign Appbyme apps do look like variants of the malicious Appbyme apps. After all, these apps are developed by the same proxy developer using the same set of templates and resources. Since the development was not a submit-and-revise process, it becomes Lshand's limitation.

6 Limitations and Discussions

6.1 What are the limitations?

Our goal is to hunt down potential Neg-Day cases based on a few assumptions (Section 3), and the limitations are the following cases that violate our assumptions. First, we assume malware development is a continuous submit-and-revise process, and hence Lshand cannot accurately judge an app development if the development case involves no submission for revision (thus no scan reports) or if the development is done on a proxy app-creation platform (see the above false-postive case study). Second, we assume malware writers never incrementally add a large number of dummy components or dummy files, and hence Lshand cannot accurately judge a malware once adding dummy components become automated (see the below discussion). In addition to the above limitations, there are two requirements: first, the scan reports from the selected online malware scanner must be comprehensive; second, feature weights and distance thresholds must be studied because information availability may differ across different selected online malware scanners.

6.2 Can we learn weight and threshold?

In our current implementation, we tune the weights and thresholds based on our confirmed AMDTs. Nevertheless, this process can be robust and automatic by applying machine learning techniques (e.g., [25, 42]). Specifically, we can learn weights and thresholds from *unobfuscated same-certificate AMDTs*, which can be derived by collecting malware samples that have high detection ratio ($r \geq 30$), overlapping components (> 5), and the same certificate. However, we did not incorporate the automation of weights and thresholds into our current implementation because we do not have an AMDT dataset that is large and convincing enough to learn from, as we only have two month-long submission snapshots. If we focus on *unobfuscated same-certificate AMDTs* that have at least five kin samples and were compiled within seven days from submission (based on the most recent file-timestamp), then we have 24 AMDTs from January 2016 based on the rescans submitted in 2018 (on average 21.99 hours per revision), and 11 AMDTs from May 2018 (7.87 hr/revision). During this research, we also have discovered 16 *unobfuscated multi-certificate AMDTs*, that each has at least five kin samples but no more two kin samples sharing the same certificate, compiled in January 2016 (34.32 hr/revision). We plan to incorporate the automation in the future when we acquire more data.

6.3 Is there a way to evade Lshand?

Lshand bases its hunting skills mainly on clustering techniques, and thus Lshand is vulnerable to manipulation upon features. We came up with two evasion approaches. First, a malware writer can knowingly incrementally adds a large number of dummy components into a malware, so that the kin malware samples could end up in different DTs because of the enlarged distance in feature space. Second, a malware writer can develop a malicious module with only necessary components and then embed this module into a fake app that is far away in feature space (hence the product malware could be in a different DT).

Although Lshand is not fully evasion-resilient, the outcome of the above evasion techniques can be limited because of the following reasons. First, malware writers cannot easily predict our Neg-Day results unless they deploy the same algorithms with the same variables and collect the same set of samples. Second, either above evasion techniques has drawbacks making them less compelling to malware writers: on one hand, incrementally adding a large number of dummy components will make the latter variants more suspicious to malware analysts and malware classifiers; on the other hand, the development of a small malicious module can still be detected and it is much easier to craft effective malware signatures from a small codebase.

6.4 Can Lshand be more scalable?

To be more scalable, Lshand is equipped with a multi-phase clustering option, and the idea is based on the map-reduce concept. To be brief, digested reports are split based on selective information (e.g., SID or timestamps), then digested reports are clustered into development subtraces, and then development subtraces are clustered into DTs. In our implementation, features for subtraces are aggregated from the digested reports by taking the mean value of features from the latest few reports. Compared to the single-phase clustering approach that took 291 hours in clustering 371,187 observations (ρ =unlimited), our two-phase clustering implementation took 93 hours for the same set of observation (note that clusterers were executed sequentially without parallelization).

However, since features are aggregated, multi-phase clustering approach may incorrectly filter out more AMDT samples (false negatives) than single-phase clustering approach. Indeed, two-phase approach reported 56 AMDTs with 1,622 samples (76 less samples than its single-phase approach), and the portion of perfect clusters is 99.14%-96.43%. A simple workaround to reduce false negatives is to set different threshold τ_i at different phase i. For example, setting $\tau_2 > \tau_1$ could reduce false negatives but it could also incorrectly report benign samples (false positives). Indeed, two-phase two-threshold approach reports 58 AMDTs with 1,799 samples, and the portion of perfect clutser is 86.83%-89.66%.

7 Related Work

Many research on Android malware countermeasures (e.g., [9,14,16,19,26,30,35,40,41,43,44]) against Android malware detection (e.g., [2, 3, 7, 10, 15, 20, 21, 23, 24, 27, 36, 37]) has been studied, but only a few focus on detecting malware development (e.g., [17]) or detecting a variant of a malware family (e.g., [12, 13]).

AMDHunter [17] aims to find Android malware development; however, AMDHunter has some critical flaws. First, AMDHunter tracks malware development cases by profiling SIDs (assuming an SID is more malicious if its submissions demonstrate little diversity and little repetition), and hence a malware writer can easily evade AMDHunter by submitting samples from multiple SIDs or decorated SIDs (e.g., repeatedly submit a variety of apps). Second, AMDHunter highly depends on the trends of detection ratio (assuming malware development will demonstrate decreasing trends which is not true for LeakerLocker), and hence a malware writer can easily evade AMDHunter by submitting detectable variants of known malware. In contrast, Lshand leverages neither SID profiles nor trends of detection ratio, and hence not only is Lshand capable of hunting down Neg-Day cases from multiple SIDs (Table 6), but also did Lshand find more Neg-Day samples in the five common Neg-Day cases in the overlapped dataset (January 2016).

RevealDroid [12] and Graziano et. al [13] each solves a similar research problem. RevealDroid [12] aims to efficiently and accurately detect-and-identify malware family, but RevealDroid is not as scalable as Lshand, because it requires all malware samples to be dissected. The author stated that RevealDroid spent 3.5 days in analyzing 9,731 apps, and most of the execution time was spent on extracting features from samples. Graziano et. al [13] aims to mine malware intelligence from public dynamic analysis sandboxes; however, their work focuses on unobfuscated and unpacked Windows PEs, and thus some of their *"most effective"* features (e.g., filename edit distance) could be easily obfuscated.

Graziano et. al [13] and SSDC [39] also aim to detect-and-identify malware family based on Ssdeep rather than malware dissection. Ssdeep-based solutions require only the Ssdeep values: two samples that have homologies will have common byte sequences in their Ssdeep values, and two Ssdeep values can imply how homologous the two corresponding samples are. Although Ssdeep seems very convenient, there are a few reasons why we avoid using Ssdeep in Lshand: first, obfuscation and compression may mess up Ssdeep triggers, so two kin obfuscated APKs may have mismatched Ssdeep values; second, even when decompressed, repackaged fake apps will still demonstrate matched Ssdeep values with benign apps and cause false classification; third, fuzzyhash-based clustering for malware has already been studied and considered problematic [22]. The downside of Lshand compared to Ssdeep-based solutions is that Lshand focuses only on Android malware; nevertheless, the idea in Lshand can be extended to different malware types by importing different features and classifiers.

8 Conclusion

We propose Lshand that is capable of hunting down 10 Neg-Day cases (with 253 samples) from January 2016 and 12 Neg-Day cases (with 256 samples) from May 2018. Our results show that Lshand is efficient and accurate in hunting down Neg-Day malware samples that were given no malware label on VirusTotal.

Acknowledgments

The work of Zhu was partially supported through NSF CNS-1618684 and ARO W911NF-17-S-0002. Peng Liu was supported by ARO W911NF-13-1-0421 (MURI), NSF CNS-1814679, and ARO W911NF-15-1-0576.

References

[1] TESSERACT: Eliminating experimental bias in malware classification across space and time. In *28th USENIX Security Symposium (USENIX Security 19)*, Santa Clara, CA, 2019. USENIX Association.

[2] Shahid Alam, Zhengyang Qu, Ryan Riley, Yan Chen, and Vaibhav Rastogi. Droidnative: Automating and optimizing detection of android native code malware variants. *Computers and Security*, 65:230 – 246, 2017.

[3] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, and Konrad Rieck. Drebin: Effective and explainable detection of android malware in your pocket. In *NDSS*. The Internet Society, 2014.

[4] Ulrich Bayer, Paolo Milani Comparetti, Clemens Hlauschek, Christopher Krügel, and Engin Kirda. Scalable, behavior-based malware clustering. In *NDSS*, 2009.

[5] Beijing Infinite Effect Media Information Technology Co., Ltd. Appbyme, an online creation platform of diy mobile applications., 2012. http://appbyme.com/.

[6] David Braue. Security tools taking too long to detect new malware, analysis warns, 2015. https://www.cso.com.au/article/566738/security-tools-taking-too-long-detect-new-malware-analysis-warns/.

[7] Gerardo Canfora, Francesco Mercaldo, and Corrado Aaron Visaggio. An hmm and structural entropy based detector for android malware. *Comput. Secur.*, 61(C):1–18, August 2016.

[8] Sanjay Chakraborty and N. K. Nagwani. Analysis and study of incremental DBSCAN clustering algorithm. *CoRR*, abs/1406.4754, 2014.

[9] Melissa Chua and Vivek Balachandran. Effectiveness of android obfuscation on evading anti-malware. In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, CODASPY '18, pages 143–145, New York, NY, USA, 2018. ACM.

[10] Ali Feizollah, Nor Badrul Anuar, Rosli Salleh, Guillermo Suarez-Tangil, and Steven Furnell. Androdialysis: Analysis of android intent effectiveness in malware detection. *Computers and Security*, 65:121 – 134, 2017.

[11] Yu Feng, Osbert Bastani, Ruben Martins, Isil Dillig, and Saswat Anand. Automated synthesis of semantic malware signatures using maximum satisfiability. In *NDSS Symposium 2017*, San Diego, CA, 2017.

[12] Joshua Garcia, Mahmoud Hammad, and Sam Malek. Lightweight, obfuscation-resilient detection and family identification of android malware. In *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, pages 497–497, New York, NY, USA, 2018. ACM.

[13] Mariano Graziano, Davide Canali, Leyla Bilge, Andrea Lanzi, and Davide Balzarotti. Needles in a haystack: Mining information from public dynamic analysis sandboxes for malware intelligence. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 1057–1072, Washington, D.C., 2015. USENIX Association.

[14] Mahmoud Hammad, Joshua Garcia, and Sam Malek. A large-scale empirical study on the effects of code obfuscations on android apps and anti-malware products. In *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, pages 421–431, New York, NY, USA, 2018. ACM.

[15] W. Hu, J. Tao, X. Ma, W. Zhou, S. Zhao, and T. Han. Migdroid: Detecting app-repackaging android malware via method invocation graph. In *2014 23rd International Conference on Computer Communication and Networks (ICCCN)*, pages 1–7, Aug 2014.

[16] Wenjun Hu, Xiaobo Ma, and Xiapu Luo. Protecting android apps against reverse engineering. *Protecting Mobile Networks and Devices: Challenges and Solutions*, page 155, 2016.

[17] Heqing Huang, Cong Zheng, Junyuan Zeng, Wu Zhou, Sencun Zhu, Peng Liu, Suresh Chari, and Ce Zhang. Android malware development on public malware scanning platforms: A large-scale data-driven study. In *Proceedings of the 2016 IEEE International Conference on Big Data (Big Data)*, BIG DATA '16, Washington, DC, USA, 2016. IEEE Computer Society.

[18] Roberto Jordaney, Kumar Sharad, Santanu K. Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. Transcend: Detecting concept drift in malware classification models. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 625–642, Vancouver, BC, 2017. USENIX Association.

[19] Jinho Jung, Chanil Jeon, Max Wolotsky, Insu Yun, and Taesoo Kim. AVPASS: Leaking and Bypassing Antivirus Detection Model Automatically. In *Black Hat USA Briefings (Black Hat USA)*, Las Vegas, NV, July 2017.

[20] Dongwoo Kim, Jin Kwak, and Jaecheol Ryou. Dwroiddump: Executable code extraction from android applications for malware analysis. *International Journal of Distributed Sensor Networks*, 11(9):379682, 2015.

[21] Dhilung Kirat, Giovanni Vigna, and Christopher Kruegel. Barecloud: Bare-metal analysis-based evasive malware detection. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 287–301, San Diego, CA, 2014. USENIX Association.

[22] Yuping Li, Sathya Chandran Sundaramurthy, Alexandru G. Bardas, Xinming Ou, Doina Caragea, Xin Hu, and Jiyong Jang. Experimental study of fuzzy hashing in malware clustering analysis. In *8th Workshop on Cyber Security Experimentation and Test (CSET 15)*, Washington, D.C., 2015. USENIX Association.

[23] Arvind Mahindru and Paramvir Singh. Dynamic permissions based android malware detection using machine learning techniques. In *Proceedings of the 10th Innovations in Software Engineering Conference*, ISEC '17, pages 202–210, New York, NY, USA, 2017. ACM.

[24] Enrico Mariconti, Lucky Onwuzurike, Panagiotis Andriotis, Emiliano De Cristofaro, Gordon Ross, and Gianluca Stringhini. Mamadroid: Detecting android malware by building markov chains of behavioral models. In *NDSS Symposium 2017*, San Diego, CA, 2017.

[25] M. Marszaek and C. Schmid. Spatial weighting for bag-of-features. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 2118–2125, June 2006.

[26] N. Miramirkhani, M. P. Appini, N. Nikiforakis, and M. Polychronakis. Spotless sandboxes: Evading malware analysis systems using wear-and-tear artifacts. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 1009–1024, May 2017.

[27] A. Narayanan, M. Chandramohan, L. Chen, and Y. Liu. Context-aware, adaptive, and scalable android malware detection through online learning. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 1(3):157–175, June 2017.

[28] Annamalai Narayanan, Mahinthan Chandramohan, Lihui Chen, and Yang Liu. Context-aware, adaptive, and scalable android malware detection through online learning. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 1(3):157–175, 2017.

[29] Palo Alto Networks. Wildfire malware analysis. https://www.paloaltonetworks.com/products/secure-the-network/wildfire.

[30] Sebastian Poeplau, Yanick Fratantonio, Antonio Bianchi, Christopher Kruegel, and Giovanni Vigna. Execute this! analyzing unsafe and malicious dynamic code loading in android applications. In *NDSS Symposium 2014*, San Diego, CA, 2014.

[31] Ford Qin. Leakerlocker mobile ransomware threatens to expose user information, 2017. http://blog.trendmicro.com/trendlabs-security-intelligence/leakerlocker-mobile-ransomware-threatens-expose-user-information/.

[32] Konrad Rieck, Philipp Trinius, Carsten Willems, and Thorsten Holz. Automatic analysis of malware behavior using machine learning. *Journal of Computer Security*, 19:639–668, 2011.

[33] James Scott. Signature based malware detection is dead. *The Cyber Security Think Tank, Institute for Critical Infrastructure Technology*, 2017.

[34] Linhai Song, Heqing Huang, Wu Zhou, Wenfei Wu, and Yiying Zhang. Learning from big malwares. In *Proceedings of the 7th ACM SIGOPS Asia-Pacific Workshop on Systems*, APSys '16, pages 12:1–12:8, New York, NY, USA, 2016. ACM.

[35] Guillermo Suarez-Tangil and Gianluca Stringhini. Eight years of rider measurement in the android malware ecosystem: Evolution and lessons learned. *arXiv preprint arXiv:1801.08115*, 2018.

[36] L. Sun, Z. Li, Q. Yan, W. Srisa-an, and Y. Pan. Sigpid: significant permission identification for android malware detection. In *2016 11th International Conference on Malicious and Unwanted Software (MALWARE)*, pages 1–8, Oct 2016.

[37] M. Sun, X. Li, J. C. S. Lui, R. T. B. Ma, and Z. Liang. Monet: A user-oriented behavior-based malware variants detection system for android. *IEEE Transactions on Information Forensics and Security*, 12(5):1103–1112, May 2017.

[38] VirusTotal. A leakerlocker sample found on virustotal (first seen: 2016-11-23), 2016. https://www.virustotal.com/en/file/d82330e1d84c2f866a0ff21093cb9669aaef2b07bf430541ab6182f98f6fdf82/analysis/1479960699.

[39] Brian Wallace. Optimizing ssdeep for use at scale and ssdeep cluster. 2015. https://www.virusbulletin.com/virusbulletin/2015/11/optimizing-ssdeep-use-scale and https://github.com/bwall/ssdc.

[40] X. Wang, Y. Yang, and S. Zhu. Automated hybrid analysis of android malware through augmenting fuzzing with forced execution. *IEEE Transactions on Mobile Computing*, pages 1–1, 2019.

[41] Fengguo Wei, Yuping Li, Sankardas Roy, Xinming Ou, and Wu Zhou. *Deep Ground Truth Analysis of Current Android Malware*, pages 252–276. Springer International Publishing, Cham, 2017.

[42] Dietrich Wettschereck and David W. Aha. Weighting features. In Manuela Veloso and Agnar Aamodt, editors, *Case-Based Reasoning Research and Development*, pages 347–358, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

[43] Y. Xue, G. Meng, Y. Liu, T. H. Tan, H. Chen, J. Sun, and J. Zhang. Auditing anti-malware tools by evolving android malware and dynamic loading technique. *IEEE Transactions on Information Forensics and Security*, 12(7):1529–1544, July 2017.

[44] Wenbo Yang, Yuanyuan Zhang, Juanru Li, Junliang Shu, Bodong Li, Wenjun Hu, and Dawu Gu. Appspear: Bytecode decrypting and dex reassembling for packed android malware. In Herbert Bos, Fabian Monrose, and Gregory Blanc, editors, *Research in Attacks, Intrusions, and Defenses*, pages 359–381, Cham, 2015. Springer International Publishing.

[45] Fangfang Zhang, Heqing Huang, Sencun Zhu, Dinghao Wu, and Peng Liu. Viewdroid: Towards obfuscation-resilient mobile application repackaging detection. In *Proceedings of the 2014 ACM Conference on Security and Privacy in Wireless Mobile Networks*, WiSec '14. ACM, 2014.