PreCrime to the Rescue: Defeating Mobile Malware One-step Ahead

Cheng Tan[2 1], *Haibo Li*[1], *Yubin Xia*[1 3], *Binyu Zang*[1], *Cheng-Kang Chu*[4], *Tieyan Li*[4]

[1]Institute of Parallel and Distributed Systems, Shanghai Jiao Tong University, China
[2]Software School, Fudan University, China
[3]State Key Laboratory of Computer Architecture, ICT, Chinese Academy of Sciences
[4]Huawei Technologies Pte Ltd, Singapore

Abstract

Prior mobile malware defensive means is usually retroactive, which may either lead to high false negatives or can hardly recover systems states from malware activities. PreCrime is a proactive malware detection scheme that detects and stops malware activities from happening. PreCrime creates mirrors of a mobile device in a resource-rich and trusted cloud, which speculatively executes multiple likely user operations concurrently to detect potential tampering and information leakage. Our preliminary evaluation shows that PreCrime introduces small performance overhead on smartphones and feasible delay during speculative execution on the cloud.

1 Introduction

As mobile devices become increasingly pervasive in our daily lives, a mass of user privacy has been shifted to the mobile devices. This, while bringing convenience of uses, also stimulates a massive growth of mobile malware. According to one survey from McAfee [5], the number of mobile malware has increased by 197% in 2013.

To defend against mobile malware, researchers have proposed a number of approaches to detect and prevent malware, by leveraging either offline analysis or online analysis, or both [1, 8, 10, 14–16]. Among these systems, it is known that the offline analysis methods cannot well detect sophisticated malware, which use subtle conditions [4] or remote controlling [6] to delay attacks to bypass the detection. Recently, Jekyll [17] uses self return-oriented programming (ROP) to change its own control flow and form attack logic, which successfully passes Apple's offline censorship.

Online analysis systems can perform better, since they are running with the device and can get more runtime information. In order to reduce performance overhead and power consumption, some online systems, e.g., Paranoid [15] and Secloud [19], leverage a trusted and powerful cloud to detect malware by maintaining and monitoring a mirror of the mobile device on the cloud. In this way, the overhead of malware detection is offloaded to the cloud.

However, current cloud-based online malware detection systems can only *detect* malware but cannot *prevent* them. This is because the mirror on the cloud usually runs *behind* the mobile device. Thus, if a malware on a phone steals user's privacy and sends it out, such damage cannot be undone even if the cloud detects such malicious behavior later. Moreover, for severe attack, such as rootkit, the lag of the checking on cloud provides an attacking window for malware, who may bypass or even break the whole security offloading system.

In this paper, we propose a proactive defensive approach, called PreCrime. PreCrime also maintains a mirror on the cloud and leverages cloud computing resources for malware detection. Unlike previous systems, the cloud mirror of PreCrime runs *one-step ahead* of the mobile device, so that as long as any malicious behavior is detected on the mirror, a warning is sent back to the device to prevent any damage.

In order to ensure that the mirror runs ahead of the phone, PreCrime delays system events on the phone and speculates user input events on the mirror. For system events such as getting a new SMS, PreCrime first sends them to the mirror to process. These events will be delivered on the phone as long as they won't trigger any malicious behavior on the mirror. On the other hand, the trusted cloud will speculate all possible user input events

APSys '14, June 25-26, 2014, Beijing, China

http://dx.doi.org/10.1145/2637166.2637224

according to current state, and fork the mirror to multiple slaves, each of which will process one input event. If any input has triggered malicious behavior, PreCrime will send a warning to the mobile device to froze the application from executing. In order to control the number of possible inputs, PreCrime only speculatives *one step* ahead.

We have implemented a prototype of PreCrime to show its effectiveness and efficiency. Our preliminary evaluation result shows that, in 99.5% of all cases, the delay of next step speculative execution on PreCrime is within 500ms, which acquires forking corresponding number of slave-mirrors to the possible user inputs. And 99% of the synchronise latency is less than 70ms which is affordable in normal scenario. Per day network traffic overhead is about 9M for five hours of usage.

The main contributions of this paper are:

- The first cloud-based proactive defensive framework for mobile security that runs one-step ahead of the mobile device and thus can prevent damage from occurring.
- The implementation of a scalable detection system on cloud cluster that can make full use of computing resources of the cloud to achieve low latency of malware detection.

The remainders of the paper are organized as follows. Section 2 introduces the background of Android events, malware trigger points and some previous works on security issues. Section 3 describes the overall design of PreCrime. And we give a rudimental evaluation at section 4. At last, we conclude and present our future work at section 5.

2 Background

2.1 Events on Android

Android is the most popular open source OS running on mobile platforms. Using Java as its programming language, Android builds an entire new middleware to hide low level details. Applications utilize a new set of event APIs provided by Android framework which is highly event driven. Such event APIs are the interfaces that are defined by the system and implemented by applications. For example, when the battery become low, a BATTERY_LOW event is broadcasted, application should register a *broadcast receiver* and override the method onReceive() to handler that event. Corresponding to each event, there are one or a series of event handlers which an application can register.

In general, Android applications can be regarded as a collection of event handlers. Each handler will be and can only be triggered by a certain kind of event. Thus, every piece of code in application, either normal or malicious, needs to be triggered by some events.

Events on Android can be classified into two groups: *System events*, which are generated by system, including low battery message, boot complete message, SMS and so on, and *user events*, which represent user's interaction, such as touching events. One important difference between the two is that the delivering of system events can be delayed, but delaying user events will hurt user's experience.

2.2 Trigger Points of Malware

Previous researches [16, 18] show that many malware are triggered by various types of events, such as phone calls, SMS receiving, boot complete message and so on. Current anti-malware systems usually need to generate events, e.g., using fuzzy testing to generate events randomly, to trigger potential attacks for malware detection.

However, malware are evolving and their trigger conditions are becoming nitpicking. For instance, some of the malware are content-based [7] that receive the malicious commands from SMS or network packages; Some hide the malicious payload on benign code which will be reorganized during attack [17]. Hence, such sophisticated malware can bypass most of current security checks easily, such as Google Bouncer and app verification service.

2.3 Related Work

Static analysis [2, 9, 13] is known to have false positive and false negative since they do not have runtime information. For example, malware can use reflection mechanism in Java to invoke sensitive functions or utilize encryption to conceal their payloads. Dry-run (e.g. symbolic execution) has the state explosion problem, which cannot be used on the complicated applications.

Using cloud to enhance mobile security is a choice to mitigate the performance collapse on the phone. Several previous works leverages cloud servers' mass resources to solve security issues: some of them are online [11, 15, 19] and some are offline [16].

Paranoid [15] is a system that clones a full replica of the phone in the cloud. With more resources in cloud server, Paranoid can provide more aggressive and powerful anti-virus check and intrusion detections. However, Paranoid cannot prevent damages caused by malware, such as privacy leak, from happening.

AppsPlayground [16] is a framework that automates the analysis process by analyzing GUI and injecting arbitrary events to trigger malicious behaviors. However,

the code coverage of the AppsPlayground is only 33% which is far from complete.

Compared with the previous works, PreCrime tries to combine the concrete states of the online checking and the isolated safety of offline checking.

3 Design and Implementation

A high-level architecture of PreCrime is illustrated in figure 1. On the cloud, PreCrime maintains a mirror of the smartphone by synchronizing their events and states. Based on current mirror state, speculative module will predict and verify the possible user input in the cloned slave-mirror. Various malware detection mechanisms listed in table 1 are adopted on cloud, for detecting abnormal behavior one step ahead.

One challenge of PreCrime is to ensure that the mirror on cloud must run ahead of the mobile phone. Therefore, malware on mobile device can be stopped before it causes any damage. PreCrime has developed the following three technologies to address this issue: first, on the mobile side, delay the delivering of *system events*; second, on the cloud side, speculate possible *user events* one-step ahead; and third, clone multiple slave-mirrors to check malware behavior concurrently.

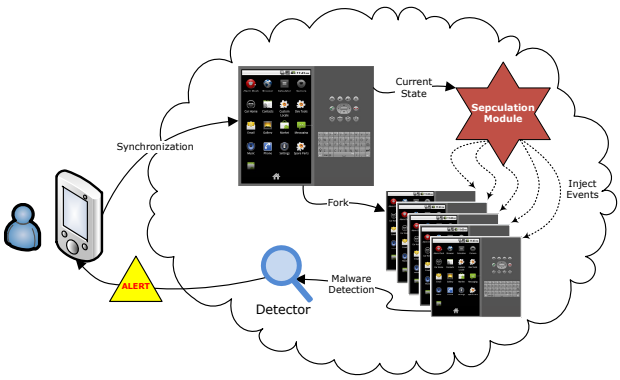

Figure 1: PreCrime Architecture

3.1 Mirror Synchronization

In order to sync the states between the cloud mirror and the mobile device, previous systems [15, 19] record the full execution trace of mobile device and replay the trace on the mirror. To minimize performance overhead and power consumption, Paranoid [15] adopts a very loose synchronization strategy, which is not suitable for PreCrime since we need to run head of the mobile device.

Our prototype implements the event-level synchronization mechanism described in Secloud [19]. PreCrime synchronizes every event on the mobile device (e.g., SMS arriving, app starting, user input) to the mirror. However, it is still possible that the states between mobile and cloud may diverge. In this case, we can furthur adopt memory-level synchronization, which is on-demand and happens infrequently. Accurate system state synchronization is another challenge which is orthogonal to our PreCrime's design.

3.2 Delaying System Events

Many malware use system events as the trigger points of their malicious payload. The most frequently used events [18] are BOOT_COMPLETED and SMS_RECEIVED. As long as the system has booted or a short message has been received, some attack is triggered. Furthermore, instead of being triggered by arbitrary events, some malware receives commands, e.g., through SMS, from attacker and execute them on the phone. In this way, naively injecting random SMS cannot trigger the malicious behaviors.

In PreCrime, the delivering of system events on the mobile side is first sent to the mirror on the cloud and delayed on the phone. Hence, the malicious code will be triggered firstly on the cloud. As long as such attack is detected, the cloud will send alert to the phone to stop the event delivering to prevent any damage on the phone. For example, considering a malware is waiting for an SMS containing attacker's command, once an SMS has arrived, the phone will first send the SMS to the cloud mirror and postpone the notification of SMS arriving. After the mirror delivering the SMS, the lurking malware will extract command from the SMS and execute it. Such behavior will be caught by deployed malware detection mechanisms, such as TaintDroid, since executing command from SMS is considered to be malicious. Then the cloud will send alert back to the phone to stop the delivery of the SMS.

By delaying the system events on the phone, the cloud mirror is able to run ahead of the mobile phone. Meanwhile, since most of the system events are asynchronous, a user will hardly notice such delay.

3.3 Speculating User Events

Another exploration takes place when the view of the phone has been changed. PreCrime will speculate possible *user events* in this occasion. Based on the strategy that only explore one step ahead of the current state, PreCrime do not meet state explosion problem. Since the number of handlers registered on one particular view is relatively small, the future states are limited. In our evaluation, there are no more than 63 handler components in 99.5% of the total views.

In terms of speculative the user's behavior, master-mirror will parse the components on the screen while GUI have been modified. All the registered handlers and handler components on current state will be reported to the *speculative module*, which is a component running on the cloud to predict the user events. Speculative module keeps a database of previous user event trace to eliminate the redundant exploration. Based on the current states and history traces, speculative module will predict the user events for further investigation. Meanwhile, master-mirror forks itself and copy-on-write its images to several slave-mirrors. These slave-mirrors will receive the predicated behaviors from speculative module and execute them locally.

3.4 Concurrent slave-mirror Clone

Many previous works believe that mass of high overhead checking mechanisms can be equipped easily using cloud. However, the cloud is actually *not* unlimited in performance. Our preliminary evaluation shows that the KVM emulator, which is the fastest emulator for Android currently available, on an Intel i7 processor can only slightly outperform a Galaxy-Nexus phone. A standard Java benchmark, CaffeinBenchmark3.0, has been tested, and the KVM emulator can only outperform the mobile device in about 18% in overall scores.

Nevertheless, considering the number of cores, cloud servers still have much more powerful computing resources than mobile phones. For example, from previous research [15] dozens of emulators can be running concurrently on one physical server. So as to utilize the resource of nowadays multi-core physical machine, PreCrime tries to explore phone's dangerous states simultaneously, instead of detecting malicious behaviors sequentially in one emulator. Therefore, concurrent slave-mirror clone is the key technique to generate these emulators.

In our preliminary implementation, PreCrime forks slave-mirrors by copy-on-write the master-mirror's memory and images and reinitializing the IPC tunnels and sockets. In addition, two algorithms are implemented for cloning multiple slave-mirrors simultaneously. One straw-man implementation is called Loop Cloning Algorithm which invokes master-mirror clone multiple times within a loop. Such algorithm will serialized all the cloning operations and has poor scalability. Another algorithm, named Hierarchy Cloning Algorithm, utilizes all the available slave-mirror with master-mirror to clone new slave-mirrors in parallel. This algorithm can fully utilize multi-core processor to achieve good performance

3.5 Malicious Behavior Detection

PreCrime tries to combine different security systems focusing on different levels as shown in table 1. Among these systems, static schemes can be used with little limitation, since the detection is done offline and there are plenty of resources on cloud. Static analysis will be only employed when the application has been installed. After the application installation, PreCrime on the phone will prevent its running until the static checking on cloud is over.

However, dynamic schemes cannot be deployed wildly. PreCrime must constrain the performance overhead of dynamic detection systems to a relatively low level. In PreCrime's malicious behavior detection design, we choose several powerful and low overhead systems as the dynamic detection tools. Each of them also should fulfill the requirement of running on different software layers and being without any conflicts when running.

3.6 Latency Analysis

PreCrime uses two different ways to handle user events and system events. For user events, they cannot be delayed for a long time. Our idea is to extend handler's time with a imperceptible time (less than 100ms) and use user's thinking and hesitating time to hide the latency of events forwarding. As illustrated in figure 2, PreCrime synchronizes the event to master-mirror before it is delivered locally. This will guarantee that the event handler on master-mirror is executed ahead of the smartphone. Supposing this user input starts a new activity, PreCrime is going to explore all the possibilities based on current view. Several slave-mirrors will be forked and each of them explores one possible path. If there is any malicious behaviors detected, an alert will send to user to stop that attack from happening.

System event's handling is straightforward. PreCrime just delays the propagation of the events on the phone and using a slave-mirror on cloud to verify whether it is safe to deliver this event. If a malicious behavior is detected, smartphone will not broadcast this event until the malware is stopped.

4 Preliminary Evaluation

In this section, we present the preliminary evaluation result of PreCrime. All the evaluation was conducted on a Samsung Galaxy Nexus smartphone, which has a 1.2 GHz TI OMAP4460 CPU, a 1 GB memory, 16 GB internal storage, a 1750 mAh Battery and 1280x720 display. The cloud server is a PC with 2.8 GHz Intel i7-930

Table 1: Cloud Mirror Protection Mechanism

Security Mechanism	Problems	Integrated Systems
Static Scheme	Virus	AV tools [2, 3]
	Spyware	RiskRanker [13], ComDroid [9]
Dynamic Scheme	Privacy Leak	TaintDroid [10]
	Privilege Escalation	XMandDroid [8]
	RootKit	RGBDroid [14]
	Code Injection/ Buffer Overflow	Syscall-based Detector [12]

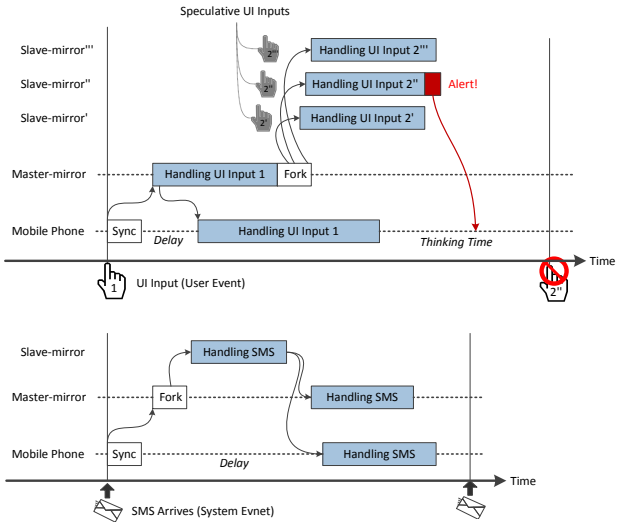

Figure 2: Procedure of Two Events Handling

quad-core CPU, 16 GB memory, 3 TB disk and 100/1000 Mbps NIC.

4.1 Number of Next Possible States

The number of next possible states for a specific view determines how many slave-mirror would be forked. We have evaluated top 150 apps from GooglePlay free list. By using fuzzy test tool (Monkey) and manually clicks, we have explored 654 distinct views from these apps. In our test, the average number of handler components in one window is 6, and the average number of event handlers for each view is 3. The detailed distribution is illustrated in CDF figure 3.

For 99.5% of total activities of the tested apps, there are no more than 44 handlers and 63 handler components. Thus, the worst case of 99.5% is that PreCrime would clone 63 slave-mirrors. According to our evaluation at section 4.3, cloning 63 slave-mirrors will cost about 500ms, which is acceptable when a new activity is started.

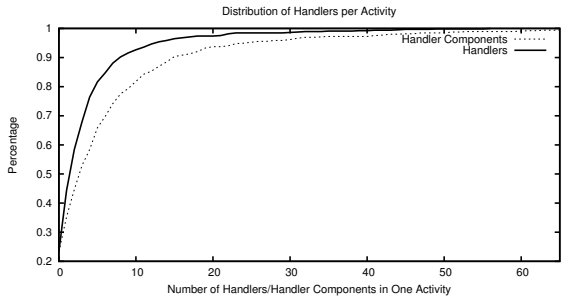

Figure 3: Distribution of Handlers/Handler Components in one Activity

4.2 PreCrime Latency

We evaluated the latency of synchronization on PreCrime in our laboratory environment to prove the feasible of our design. The latency contains two parts: network round-trip time from mobile to cloud and the time of event collection and injection to mirror. Hence, event sync time is highly relevant to the network status.

In our test, we have deployed PreCrime on the smartphone, Galaxy Nexus, and a PC as cloud. They are connected through the wifi network. Figure 4 is a CDF graph of network RTT and event sync latency. In our lab, 90% of the event sync latencies are less than 21.4ms, and 95%, 99%, 99.5% are less than 25.2ms, 74.9ms, 112ms respectively.

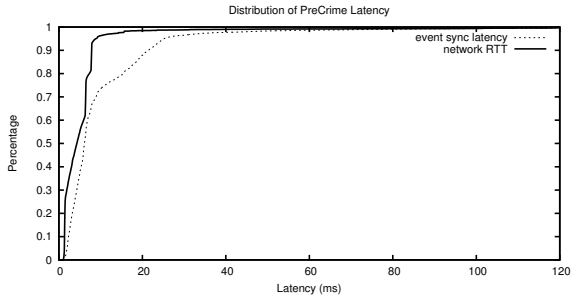

Figure 4: Network RTT and Event Sync Latency

4.3 Forking Concurrent Slave-mirrors

The performance of concurrent Slave-mirrors forking is critical for the speculative execution. Thus, we have evaluated the time for cloning 1, 3, 7, 15, 31, 63 and 127 slave-mirrors from one master-mirror in both Hierarchy Cloning Algorithm and Loop Cloning Algorithm. The detailed data is illustrated in figure 5.

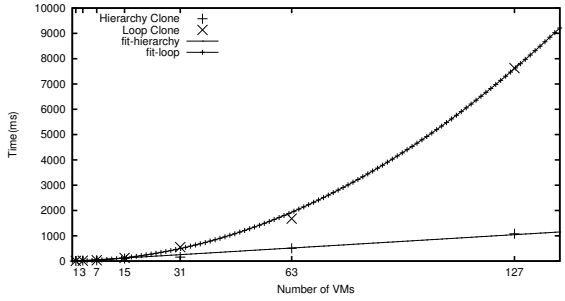

Figure 5: Two Fork Algorithm Performance

As we can see, the scalability of Hierarchy Cloning Algorithm is linear, and the scalability of Loop Cloning Algorithm is polynomial. By using Hierarchy Cloning Algorithm, the average time for cloning one slave-mirror is about 8.3ms. In detail, the time of cloning 1, 3, 7, 15, 31, 63 and 127 slave-mirrors are 6.3, 25, 36.5, 100.6, 158.3, 504.3, 1080 ms respectively.

Strangely, there is an random stall, which steadily costs 4s, during copy-on-write the images on btrfs. Such pauses might come from the implementation of COW operations in btrfs. And the problem might be solved by building PreCrime on two IO-mirrored btrfs disks, which is our future work.

4.4 Network Traffic Overhead

In order to test the network traffic overhead of PreCrime, we designed two scenarios about daily usage of smartphone: adding ten contacts with email address, phone number, home address and playing a popular mobile game “Cut the Rope” for ten minutes.

Because of the preliminary implementation, only events have been sent without other non-determinisms. And all the data are plaintext without any encoding. Table 2 shows the evaluation result.

Table 2: Network Traffic on two scenarios

Application	Evaluation Time (s)	Traffic Size (KB)	Network Traffic (M/H)
Contacts	615	253	1.45
Cut the Rope	600	400	2.34

The result shows that PreCrime leads to feasible network traffic overhead for synchronization. If the user use its smartphone five hours a day, including three hours of normal usage and two hours of game, about 9M network traffic will be cost per day.

5 Conclusion and future work

In this paper, we propose a cloud-based proactive defensive framework named PreCrime. PreCrime constructs a mirror of the smartphone on cloud, which is called master-mirror. Based on the current states of master-mirror, PreCrime explores the possible states simultaneously in several slave-mirrors by concurrent slave-mirror clone to utilize the multi-core capability on today’s hardware. All the speculative behaviors running on slave-mirrors will be inspected by multiple detection tools.

Several benefits will be achieved by PreCrime: False positives will be avoided by using real states for checking; Performance overhead on the phone is mitigated by offloading heavy dynamic inspections to cloud; States explosion problem is assuaged by forecasting only one step ahead. Also, integrating new security mechanism to PreCrime is transparent to user, because the performance of smartphone will keep stable.

In our future work, we will implement the master-mirror detailed state synchronization using DSM or non-deterministic replay system. And the slave-mirror clone feature will be extended to the KVM based emulator from current qemu based emulator. In order to reduce the exploring space of PreCrime on runtime, some offline exploring can be achieved to examine some critical paths, just like what AppsPlayground [16] does. And another optimization about speculative execution might be to explore more aggressively with several steps ahead of current states in the spare time. This will lessen the number of exploration in the following checking.

Further more, slave-mirror do not have to be a full-fledged emulator. It can only run a Dalvik virtual machine instead. Since all we want to know is that whether there are any attacks in the future, display or touching inputs are not necessary.

6 Acknowledgments

We thank the anonymous reviewers for their insightful comments. This work is supported by the Opening Project of State Key Laboratory of Computer Architecture, a research grant from Huawei Technologies, Inc., and China National Natural Science Foundation (No. 61303011).

References

[1] ”android and security”. ”http://googlemobile.blogspot.jp/2012/02/android-and-security.html”.

[2] Clamav. http://www.clamav.net/lang/en/.

[3] F-secure corporation. f-secure mobile anti-virus. http://mobile.f-secure.com/.

[4] Magna carta holy grail. http://www.esecurityplanet.com/mobile-security/mcafee-warns-of-july-4-android-malware.html.

[5] "mcafee mobile security report". http://www.mcafee.com/hk/resources/reports/rp-mobile-security-consumer-trends.pdf.

[6] Security alert: Anserverbot, new sophisticated android bot found in alternative android markets. http://www.csc.ncsu.edu/faculty/jiang/AnserverBot/.

[7] Security alert: New nickibot spyware found in alternative android markets. http://www.csc.ncsu.edu/faculty/jiang/NickiBot/.

[8] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, and A.-R. Sadeghi. Xmandroid: A new android evolution to mitigate privilege escalation attacks.

[9] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner. Analyzing inter-application communication in android. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 239–252. ACM, 2011.

[10] W. Enck, P. Gilbert, B. Chun, L. Cox, J. Jung, P. McDaniel, and A. Sheth. TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In *Proc. of OSDI*, 2010.

[11] J. Flinn and Z. Mao. Can deterministic replay be an enabling tool for mobile computing? In *Proc. HotMobile*, 2011.

[12] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. A sense of self for unix processes. In *Security and Privacy, 1996. Proceedings., 1996 IEEE Symposium on*, pages 120–128. IEEE, 1996.

[13] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang. Riskranker: scalable and accurate zero-day android malware detection. In *Proc. MobiSys*, 2012.

[14] Y. Park, C. Lee, C. Lee, J. Lim, S. Han, M. Park, and S.-J. Cho. Rgbdroid: a novel response-based approach to android privilege escalation attacks. In *Proceedings of the 5th USENIX conference on Large-Scale Exploits and Emergent Threats*, LEET'12, pages 9–9, Berkeley, CA, USA, 2012. USENIX Association.

[15] G. Portokalidis, P. Homburg, K. Anagnostakis, and H. Bos. Paranoid android: versatile protection for smartphones. In *Proc. ACSAC*, 2010.

[16] V. Rastogi, Y. Chen, and W. Enck. Appsplayground: automatic security analysis of smartphone applications. In *Proceedings of the third ACM conference on Data and application security and privacy*, pages 209–220. ACM, 2013.

[17] T. Wang, K. Lu, L. Lu, S. Chung, and W. Lee. Jekyll on ios: when benign apps become evil. In *Presented as part of the 22nd USENIX Security Symposium*, pages 559–572. USENIX, 2013.

[18] Y. Zhou and X. Jiang. Dissecting android malware: Characterization and evolution. In *Proc. Oakland*, 2012.

[19] S. Zonouz, A. Houmansadr, R. Berthier, N. Borisov, and W. Sanders. Secloud: A cloud-based comprehensive and lightweight security solution for smartphones. *Computers & Security*, 37:215–227, 2013.