Evading Android Runtime Analysis via Sandbox Detection

Timothy Vidas, Nicolas Christin
Carnegie Mellon University

Presented by Hitakshi Annayya

Contents

Motivation

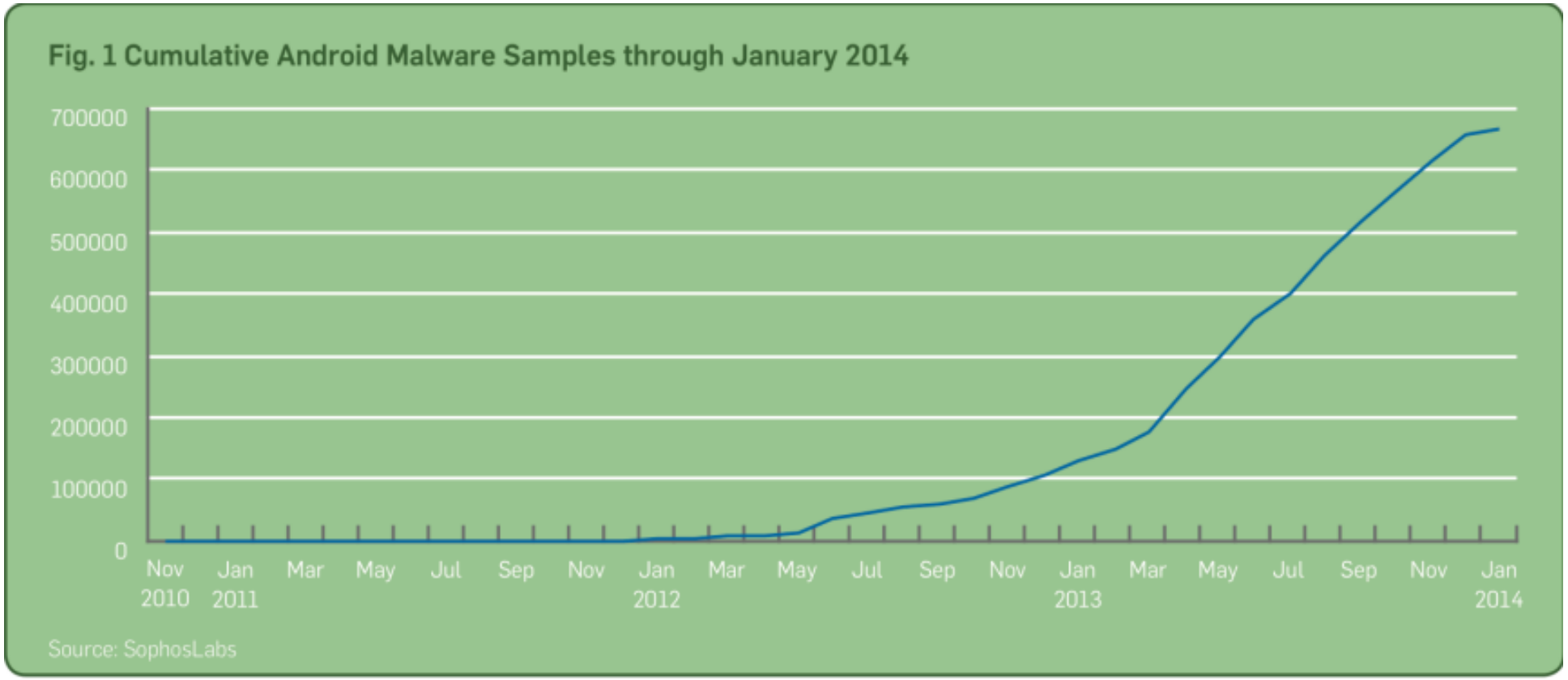
The mobile app market is truly a global phenomena. In 2012 alone, there were 45 billion apps downloaded.

The increased computing power and network connectivity is attracting the attention of attackers, looking to peddle malware on innocent mobile bystanders.

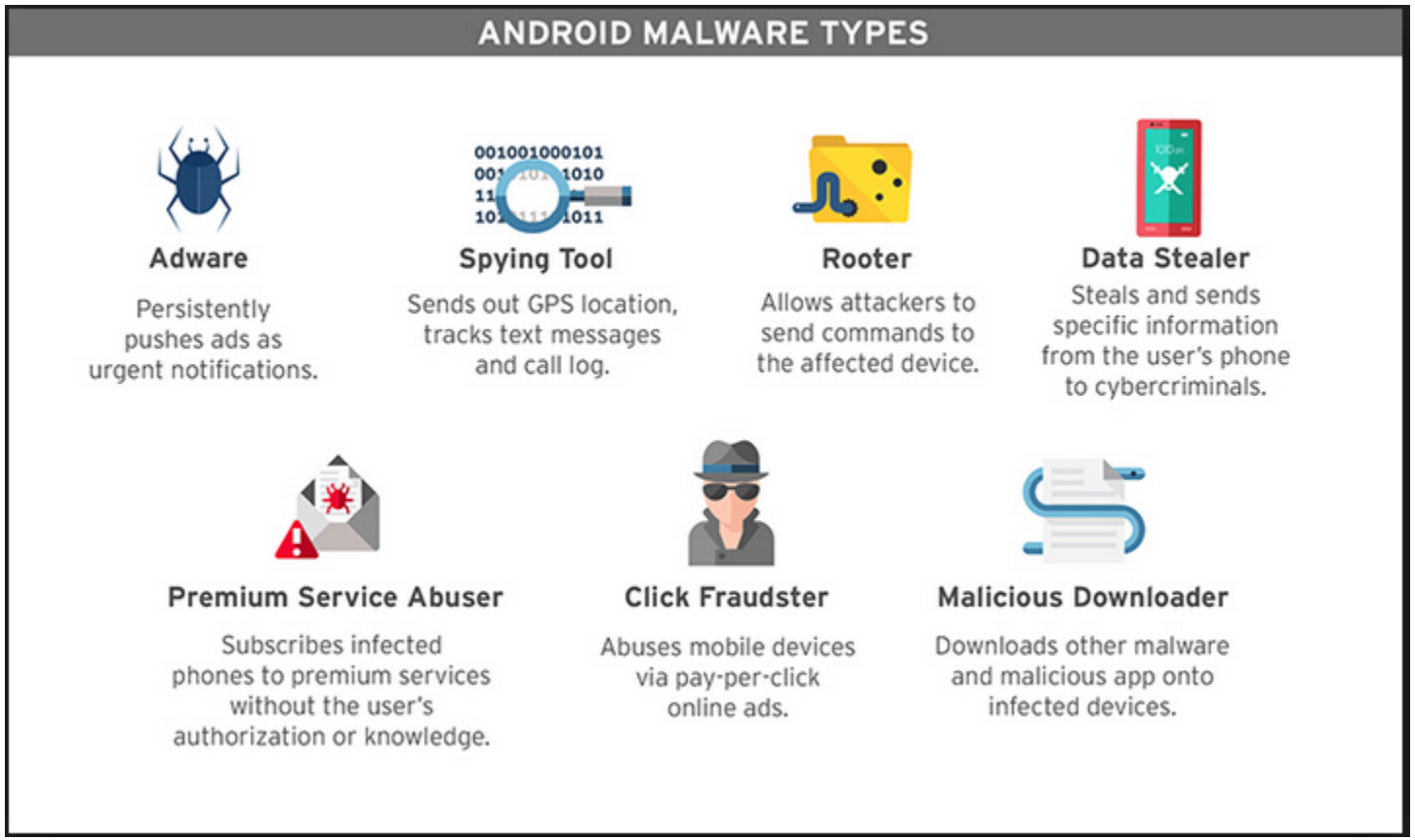
The mobile application ecosystem is lacking in strong analysis tools and techniques.

Open Question???

Recent years witness colossal growth of Android malware



https://www.sophos.com/en-us/medialibrary/PDFs/other/sophos-mobile-security-threat-report.pdf?la=en



https://www.google.com/search?q=cumulative+android+malware+samples&espv=2&biw=1366&bih=623&source=lnms&tbm=isch&sa=X&ved=0CAcQ_AUoAWoVChMIqL2C7KuXyAIVSQySCh0M-wL5#imgrc=U1HeMrNw0avuuM%3A

Most dangerous Android malware attacks:

- **Fake Banking Apps**: This lured the customers into entering their online account login details.
- **Android.Geinimi**: This corrupted many legitimate Android games on Chinese download sites.
- **DroidDream**: It infected devices, breached the android security sandbox and stole data.
- **AndroidOS fake player**: It seems to be a media player and silently sends SMS to premium SMS numbers.

Introduction

- When a new piece of malware is discovered, it must be analyzed in order to understand its capabilities and the threat it represents.
- Techniques for detecting Android runtime analysis systems often rely on virtualization or emulation, to process mobile malware.
- Dynamic analysis, consists of executing the malware in a controlled environment to observe effects to the host system and the network.

The primary contribution of this paper is to demonstrate that dynamic analysis platforms for mobile malware authors may still employ virtualization or emulation detection to alter behavior and ultimately evade analysis or identification

Android Emulator

- Can run virtual mobile devices on a computer
- mimics all of the hardware and software features of a typical mobile device

Android Emulator

The Android SDK includes a mobile device emulator — a virtual mobile device that runs on your computer. The emulator lets you develop and test Android applications without using a physical device.

Techniques used to detect a runtime analysis in Android

- Differences in behavior
- Performance
- Hardware and software components and
- Those resulting from analysis system design choices

Emulator Detection

Differences in behavior

API method	Value	meaning
Build.ABI	armeabi	is likely emulator
Build.ABI2	unknown	is likely emulator
Build.BOARD	unknown	is emulator
Build.BRAND	generic	is emulator
Build.DEVICE	generic	is emulator
Build.FINGERPRINT	generic††	is emulator
Build.HARDWARE	goldfish	is emulator
Build.HOST	android-test††	is likely emulator
Build.ID	FRF91	is emulator
Build.MANUFACTURER	unknown	is emulator
Build.MODEL	sdk	is emulator
Build.PRODUCT	sdk	is emulator
Build.RADIO	unknown	is emulator
Build.SERIAL	null	is emulator
Build.TAGS	test-keys	is emulator
Build.USER	android-build	is emulator
TelephonyManager.getDeviceId()	All 0's	is emulator
TelephonyManager.getLine1 Number()	155552155xx†	is emulator
TelephonyManager.getNetworkCountryIso()	us	possibly emulator
TelephonyManager.getNetworkType()	3	possibly emulator (EDGE)
TelephonyManager.getNetworkOperator().substring(0,3)	310	is emulator or a USA device (MCC)‡
TelephonyManager.getNetworkOperator().substring(3)	260	is emulator or a T-Mobile USA device (MNC)
TelephonyManager.getPhoneType()	1	possibly emulator (GSM)
TelephonyManager.getSimCountryIso()	us	possibly emulator
TelephonyManager.getSimSerial Number()	89014103211118510720	is emulator OR a 2.2-based device
TelephonyManager.getSubscriberId()	310260000000000‡‡	is emulator
TelephonyManager.getVoiceMailNumber()	15552175049	is emulator

http://users.ece.cmu.edu/~tvidas/papers/ASIACCS14.pdf

Emulator Detection

Differences in performance

CPU Performance

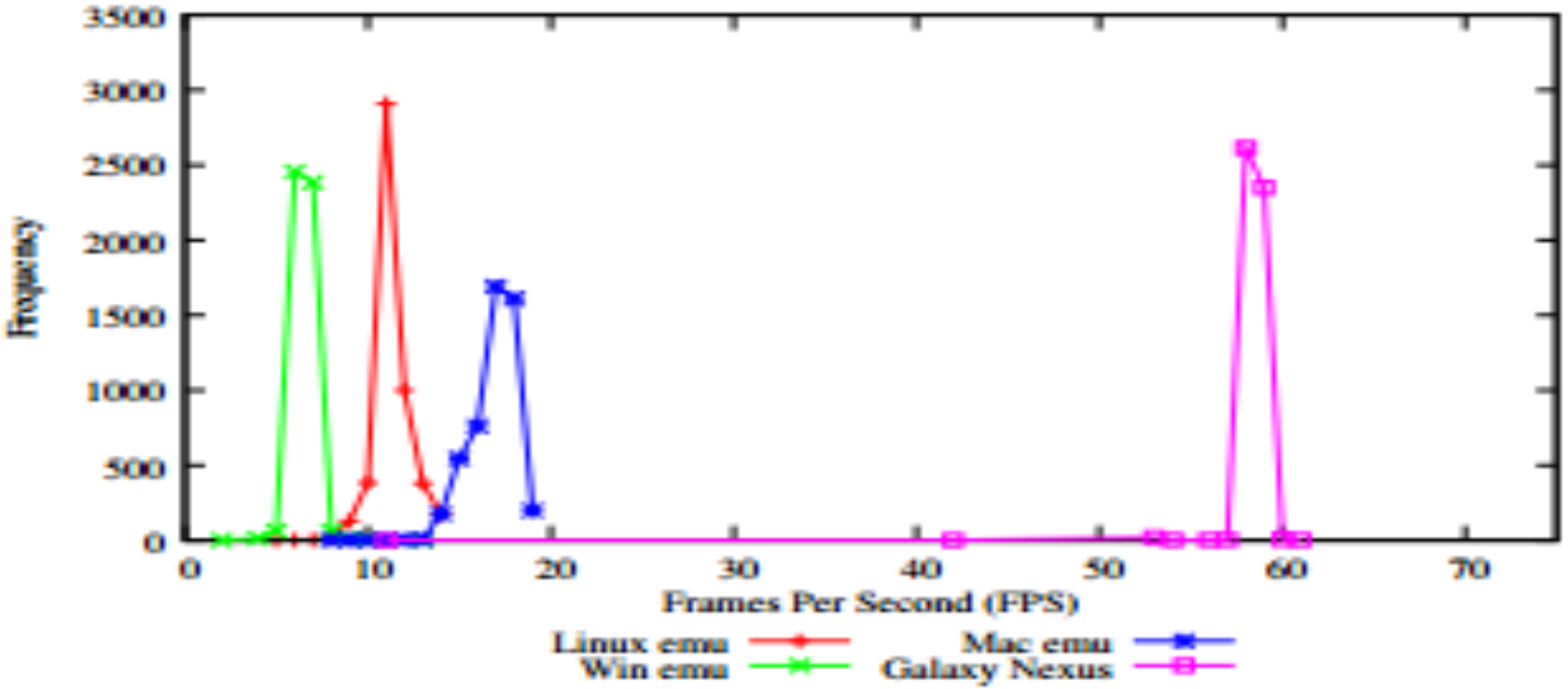
Created a Java Native Interface (JNI) application for Android using the NDK

Device	Average Round Duration (Seconds)	Standard Deviation
PC (Linux)	0.153	0.012
Galaxy Nexus (4.2.2)	16.798	0.419
Samsung Charge (2.3.6)	22.647	0.398
Motorola Droid (2.2)	24.420	0.413
Emulator 2.2	62.184	7.549
Emulator 4.2.2	68.872	0.804

http://users.ece.cmu.edu/~tvidas/papers/ASIACCS14.pdf

Pi calculation round duration on tested devices using AGM technique (16 rounds). The tested devices are noticably slower at performing the calculations than related devices running similar software.

Graphical performance



http://users.ece.cmu.edu/~tvidas/papers/ASIACCS14.pdf

Android 4.2.2 FPS Measurements: Emulators clearly show a low rate, and more of a bell curve than the Galaxy Nexus which shows almost entirely 59-60 FPS.

Emulator Detection

Differences in components

```
1 int level = batteryStatus.getIntExtra(BatteryManager.EXTRA_LEVEL, -1);
2 int scale = batteryStatus.getIntExtra(BatteryManager.EXTRA_SCALE, -1);
3 float batteryPct = level / (float)scale;
4
5 boolean isCharging = status == BatteryManager.BATTERY_STATUS_CHARGING ||
6                      status == BatteryManager.BATTERY_STATUS_FULL;
```

http://users.ece.cmu.edu/~tvidas/papers/ASIACCS14.pdf

Battery level emulator detection example

If batteryPct is exactly 50% or the level is exactly 0 and the scale is exactly 100, the device in question is likely an emulator. The level could be monitored over time to ensure it varies, and the charging status could be used to determine if the battery should be constant

Emulator Detection

Differences due to system design

Android-specific design decisions

If an attacker can determine that a device is not actually in use, the attacker may conclude that there is no valuable information to steal or that the device is part of an analysis system.

Usage indicators such as the presence and length of text messaging and call logs

Evaluation

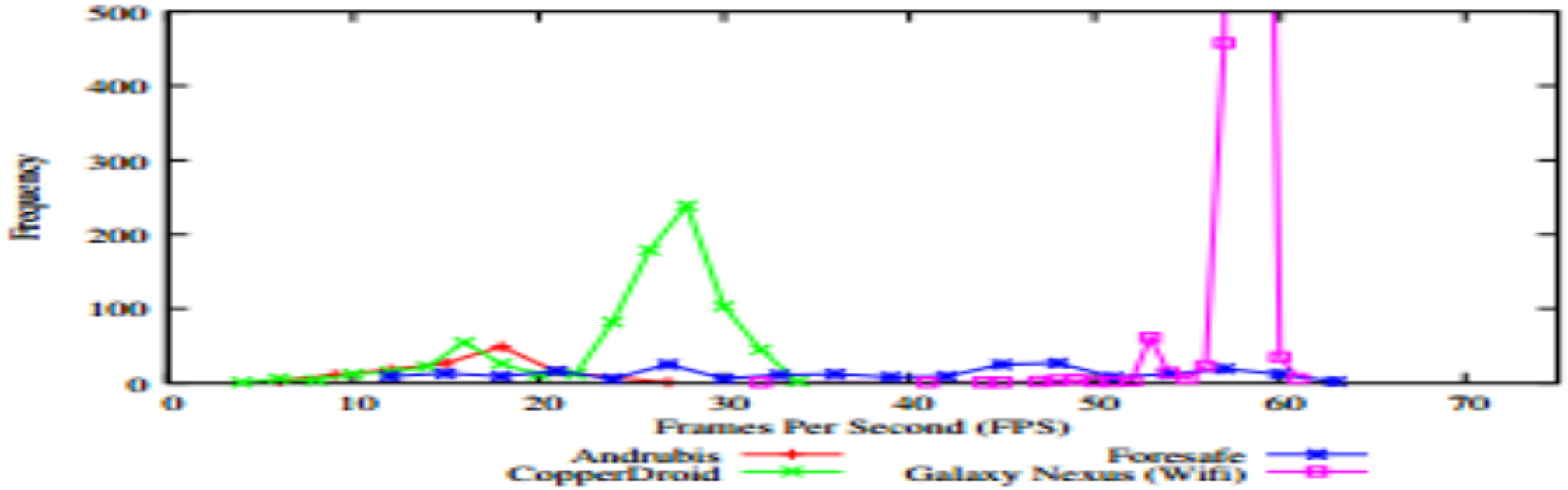
Candidate Sandboxes :

- Andrubis
- SandDroid
- Foresafe
- Copperdroid
- AMAT
- Mobile-sandbox and
- Bouncer

Behavior evaluation

- The SDK and TelephonyManager detection methods prove successful against all measured sandboxes.
- The Build parameters, such as HOST, ID, and manufacturer require a more complex heuristic in order to be useful.
- Detecting the emulated networking environment was also very successful

Performance evaluation



http://users.ece.cmu.edu/~tvidas/papers/ASIACCS14.pdf

FPS measurements for sandboxes: For comparison, a physical Galaxy Nexus was re-measured using the same application.

The physical device shows strong coupling at 59 FPS and all of the sandboxes demonstrate loose coupling and wide distribution, indicating that they all make use of virtualization.

CONCLUSION

- Virtualization is not broadly available on consumer mobile platforms.
- Mobile-oriented detection techniques will have more longevity than corresponding techniques on the PC.
- Presented a number of emulator and dynamic analysis detection methods for Android devices
- Designers of dynamic analysis systems must universally mitigate all detections.

References

[12] U. Bayer, P. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda. Scalable, behavior-based malware clustering. In NDSS, 2009.

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

[32] T. H. Ptacek and T. N. Newsham. Insertion, evasion, and denial of service: Eluding network intrusion detection. Technical report, DTIC Document, 1998.


THANK
YOU!

Evading Android Runtime Analysis via Sandbox Detection.

Timothy Vidas and Nicolas Christin. In AsiaCCS'14

Paper Discussion

- Zhenyu Ning,
- CSC 6991 – Advanced Computer System Security

- Evading Android Runtime Analysis via Sandbox Detection

- This paper mainly talks about approaches to detect the virtualization or emulation system from a simple android application with few or no permissions. To achieve this point, some differences between the physical device and the virtualization or emulation system on system behavior, runtime performance, physical components and system design are listed in the paper together with approaches to detect these differences. After that, these approaches are experimentally proved by the evaluation on some sandboxes such as Andrubis, SandDroid and Foresafe.

- Upon the paper, we can easily conclude that a malware using these approaches can detect the existence of a virtualization or emulation system and then to perform a different, benign behavior to avoid from being detected. And as it concluded in the paper, this problem is still an open problem which may need more and more efforts and research.

- But as we learned in previous classes, a hardware based isolated execution environment, such as SMM in x86, TrustZone in arm and so on, can generally resolve this problem and provide a nearly complete transparent environment to perform malware analysis.

Paper Discussion

- Sai Tej Kancharla
- CSC 6991- Advanced Computer Security
- Evading Android Runtime Analysis via Sandbox Detection

- The paper "Evading Android Runtime Analysis via Sandbox Detection" by Timothy Vidas and Nicolas Christin discusses about detecting Emulation or Virtulalization from Android applications which are commonly accessible to all with few or no permissions. The detection based techniques are divided into 4 classes. They are Behavior, Performance, Hardware and Software Components and lastly System Design. All the approaches are applied on various widely available sandboxes like Andrubis, SandDroid, Foresafe, Copperdroid, AMAT, and Bouncer
- The paper talks about detecting emulation software relatively easily exploiting some fundamental flaws in the systems. The paper discusses about how we can detect Emulation using Android API. The paper also compares about the difference in CPU performance between Emulators and similar physical devices using crude system of measure duration of lengthy computation. It also discusses about the difference in Graphical performance which is measured by calculating Frames Per Second(FPS). It also discusses on how the emulation software cannot encompass all the sensors working on the physical device leading to its detection easily.
- This paper proves that malware exploiting these techniques can easily avoid detection and that could prove quite harmful. Most of the problems could be solved by changing some fundamental flaws in the systems excluding the tough to solve problems like hardware related ones. But this problem needs to be addressed quickly ranging on how fast mobile platform is advancing in the global market.

Paper Discussion

- Hitakshi Annayya
- Evading Android Runtime Analysis via Sandbox Detection
- This paper “Evading Android Runtime Analysis via Sandbox Detection” by Timothy Tidas and Nicolas Christin presents different dynamic analysis platforms for mobile malware that purely rely on emulation or virtualization face fundamental limitations that may make evasion possible. Since there is a rapid increase in mobile malware, the authors did demonstration and evaluated the techniques by dynamic analysis, consists of executing the malware in a controlled environment to observe effects to the host system and the network.
- Techniques are classified into four broad classes showing the ability to detect systems based on differences in behavior, performance, hardware and software components, and those resulting from analysis system design choices. Emulator detection by difference in behavior is detected through the mobile API methods and their return values and emulated networking. Emulator detection by performance is detected by CPU performance and Graphical performance. Emulator detection by difference in components is by Hardware and software components. For example detecting battery level emulator. Lastly, emulator detection in differences due to system design is through PC system design decision and Andriod-specific design decision.
- Evaluation of these techniques are done against the real analysis system such as Andrubis, SandDroid, Foresafe, Copperdroid, AMAT, mobile-sandbox, and Bouncer. Evaluation report is generated for all the Android API methods including for networking but not for HOST methods. The Sensors counts were different from experimented results and Sandbox exhibit very few sensors.
- Finally, as a conclusion strong dynamic analysis tool for mobile malware still an open question in research fields.
- Advantages:
  - Detections are rooted in observed differences in hardware, software and device usage
  - Accelerometer values would yield definitive clues that the malware is running in a sandboxed environment
- Limitations:
  - Mobile-oriented detection techniques will have more longevity than corresponding techniques on the PC because of Virtualization
  - Sensor counts needs to be addressed by techniques to provide all the values

Paper Discussion

- Lucas Copi
- CSC 6991
- Transparent Malware Analysis II

- The paper Evading Android Runtime Analysis via Sandbox Detection focuses on different methods utilized by malware on the Android platform to detect virtualization and thwart dynamic analysis systems. All of the techniques assume the permission level of standard applications downloaded from the Google Play store on a standard Android system.

- The paper breaks down emulation detection into three main sections: differences in performance, differences in components, and differences due to system design. Due to overhead from emulation, malware can utilize performance differences as a detection method for emulation. On both emulators tested the emulated CPU performance was much lower than CPU performance of standard Android systems. Additionally, graphical performance measures also showed a distinct decline due to emulation. Moreover, malware can use several built in Android API's to detect the difference between a physical machine and an emulator. Traits such as CPU serial numbers, memory types, network configuration, and system level software required to monitor hardware (i.e. battery monitor) all display different values and behavior during emulation than they typically would on a physical device. Finally, stock software components that are traditionally found on Android devices (i.e. Google Play Store) are rarely present on an emulated environment.

- By checking many of the above features malware can detect the presence of an emulated environment and modify behavior to display dormant rather than malicious behavior. This reduces the effectiveness of dynamic analysis systems. Although some emulators have begun to remedy these issues they are still largely immature and present minimal barriers for malware.

Term Projects Discussion