# Detecting Android Root Exploits by Learning from Root Providers

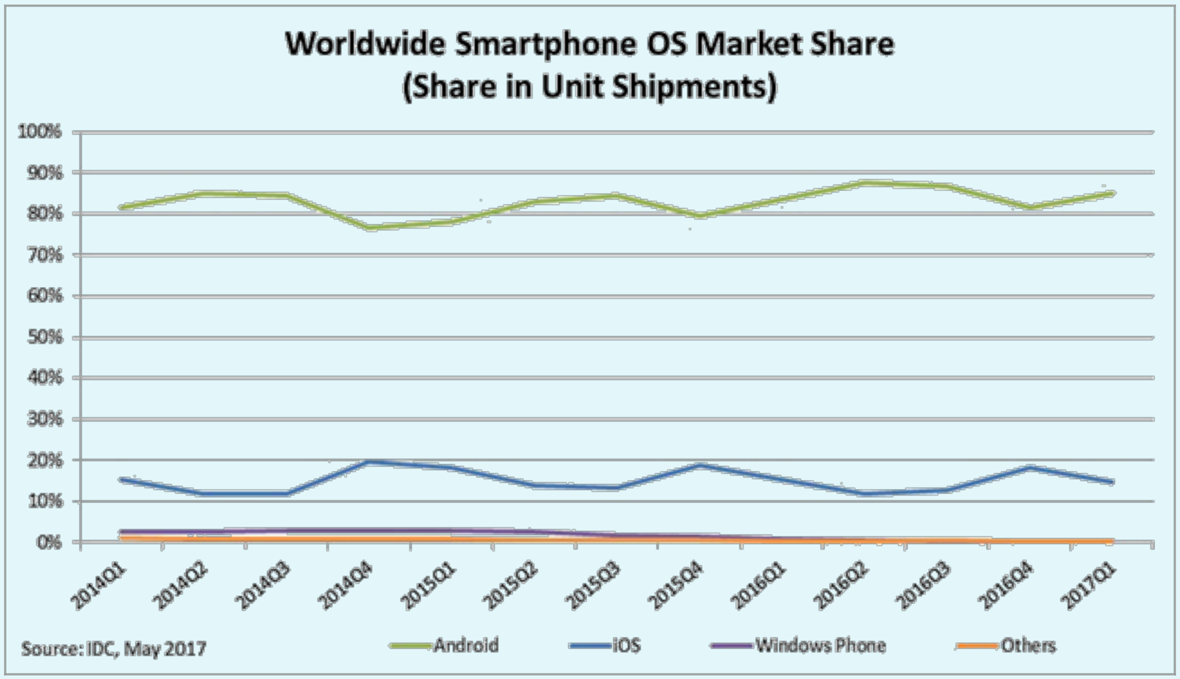
Ioannis Gasparis, **Zhiyun Qian**, Chengyu Song, Srinkanth V. Krishnamurthy

UC RIVERSIDE
UNIVERSITY OF CALIFORNIA

# We all love our smartphones

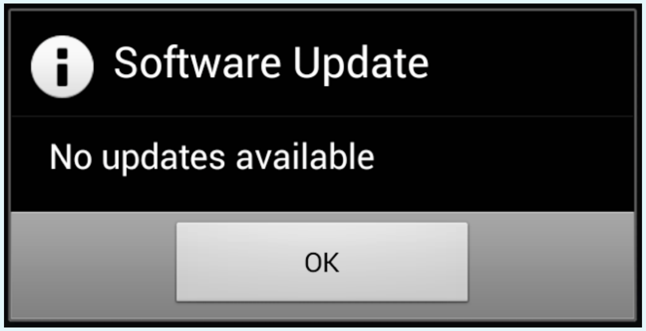

# We all love our smartphones

- I love my Android phones especially!

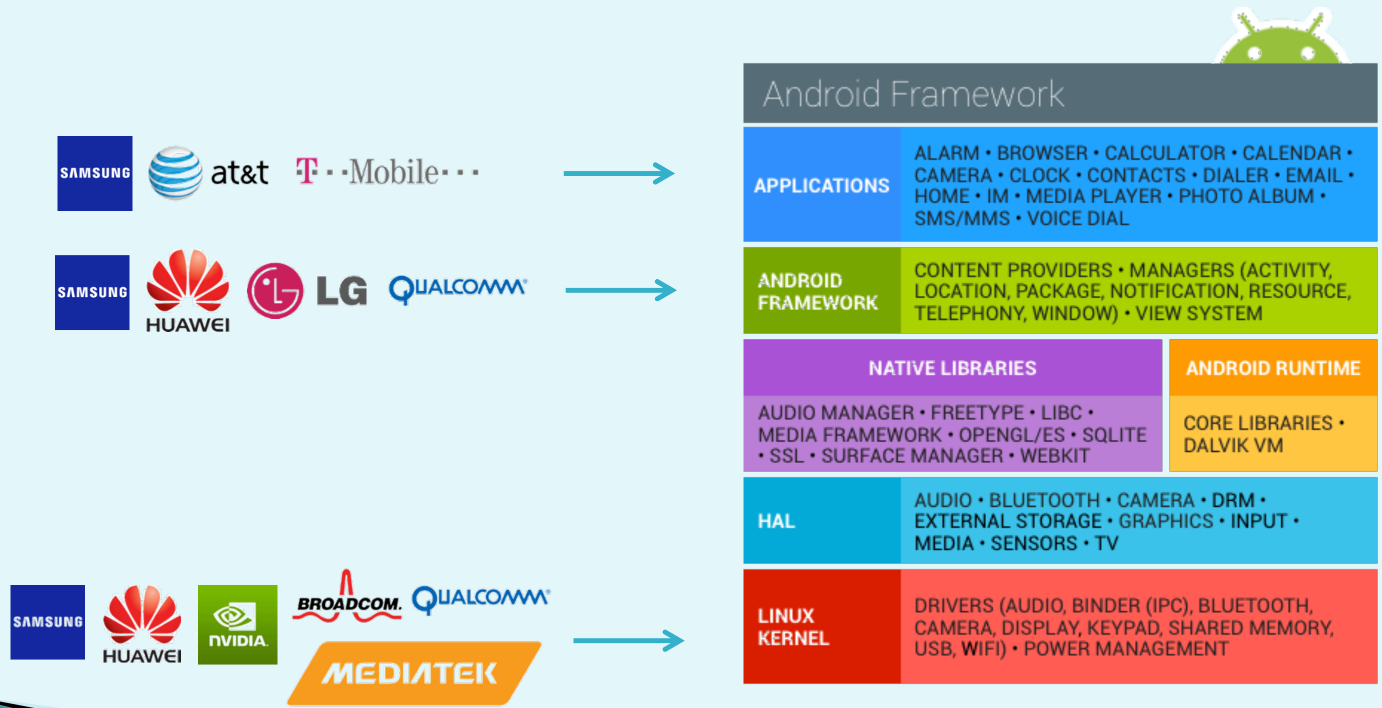


# Why Android security is challenging?

- Fragmentation / diversity / customization
- Slow update

# Everybody puts their code into Android



# Kernel / driver vulnerabilities

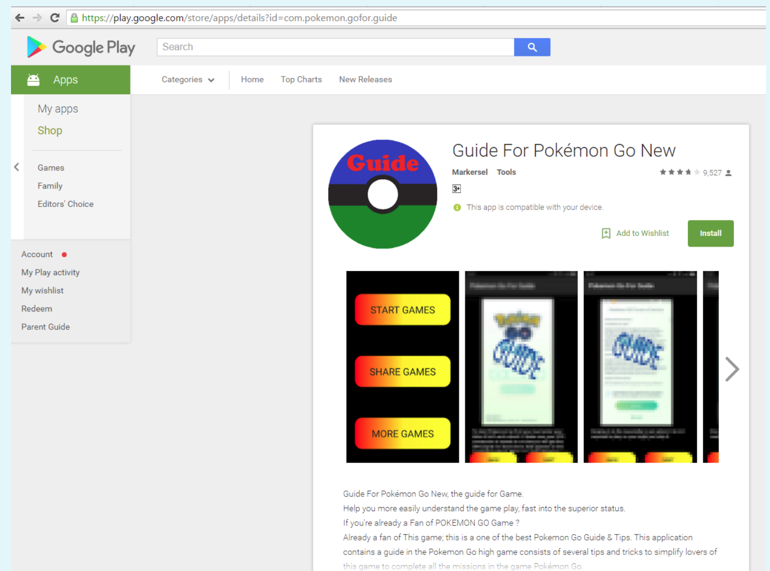
Android Security Bulletins

| Bulletin | Languages | Published date | Security patch level |
|---|---|---|---|
| August 2017 | Coming soon | August 7, 2017 | 2017-08-01<br>2017-08-05 |
| July 2017 | English / 日本語 / 한국어 / ру́сский / 中文 (中国) / 中文 (台灣) | July 5, 2017 | 2017-07-01<br>2017-07-05 |
| June 2017 | English / 日本語 / 한국어 / ру́сский / 中文 (中国) / 中文 (台灣) | June 5, 2017 | 2017-06-01<br>2017-06-05 |
| May 2017 | English / 日本語 / 한국어 / ру́сский / 中文 (中国) / 中文 (台灣) | May 1, 2017 | 2017-05-01<br>2017-05-05 |
| April 2017 | English / 日本語 / 한국어 / ру́сский / 中文 (中国) / 中文 (台灣) | April 3, 2017 | 2017-04-01<br>2017-04-05 |
| March 2017 | English / 日本語 / 한국어 / ру́сский / 中文 (中国) / 中文 (台灣) | March 6, 2017 | 2017-03-01<br>2017-03-05 |

100+ CVEs ->
Elevation of privilege:
1. only AOSP,
2. excluding closed-source components

# Android root exploits

- From vulnerabilities to exploits
  - Towelroot, Pingpong root, DirtyCow, perf use-after-free (Samsung S7)
- Malware with root exploit capabilities
  - GODLESS, HummingBad, PokemonGo guide app, Dvmap, SpyDealer, …
  - Obfuscated, anti-debugging/virtualization
  - Dynamically load exploits
  - Survive months before taken down
    - e.g., Pokemon Go Guide removed after 500,000+ downloads



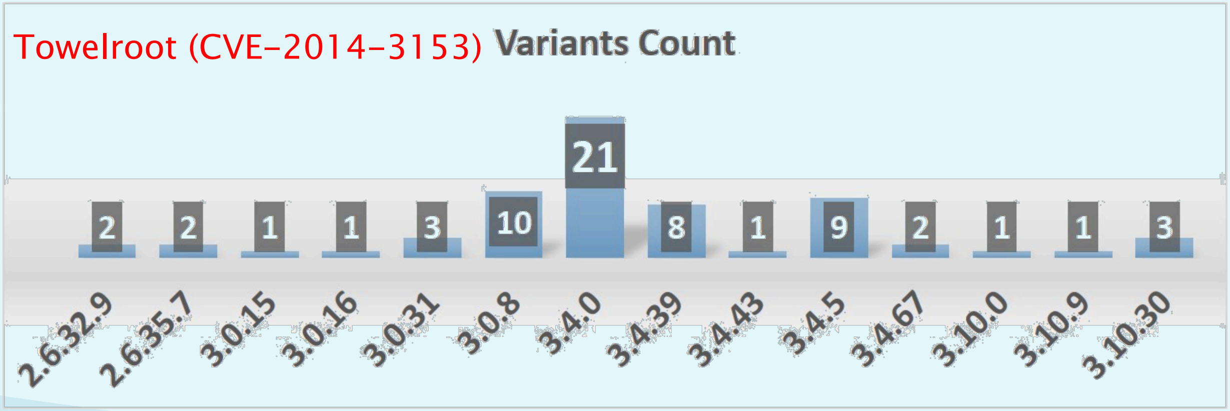
# Goal

- Develop a cloud-based app screening system (similar to bouncer)
  - Addressing challenges mentioned earlier

- Detect exploits against known vulnerabilities
  - Zero-day out-of-scope
  - Android malware exploit known vulnerabilities

# Android root exploits

- Dual problem
  - A challenging task to write
  - A challenging task to detect
- Availability of vulnerabilities
- Exploit adaptation



# How to catch variety of root exploits?

Naïve solution:

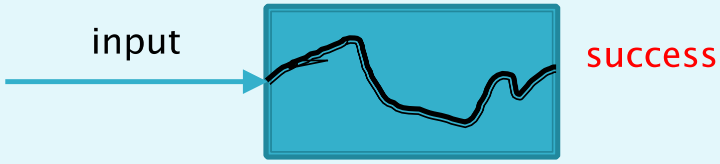
Our solution:

# Requirements of the analysis environment

- Need to emulate the correct environment/preconditions
  - Device environment (device, model, version)
  - Program preconditions

```
int fd = open("/dev/exynos-mem", O_RDWR);
if (fd == -1) {
    printf("[!] Error opening /dev/exynos-mem\n");
    exit(1);
}
```

  - Input generation problem (system call return values)



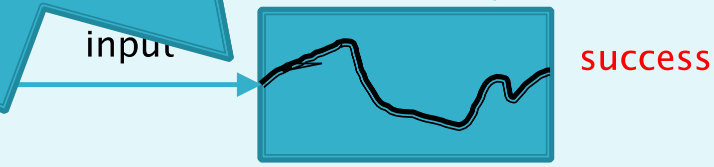
# Requirements of the analysis environment

- Need to emulate the correct environment/preconditions
  - Device environment (device, model, versi
  - Program preconditions

```
int fd  ("/d  R);
if (fd
  em\n");
}
```

  - Inpu

Where can we collect data?



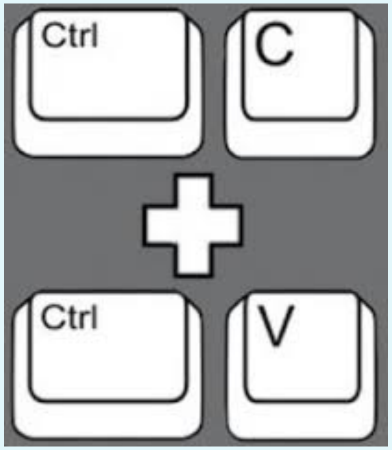
# Key observations

- One-click root apps [CCS’ 15]
  - Some claim to support 100,000+ models
  - One app contains: 167 exploit binaries, 59 exploit families

[CCS 15] H. Zhang, D. She, Z. Qian,
Android Root and its Providers: A Double-Edged Sword

# Key observations

- One-click root apps [CCS' 15]
- Root exploits mostly reused




[CCS 15] H. Zhang, D. She, Z. Qian,
Android Root and its Providers: A Double-Edged Sword

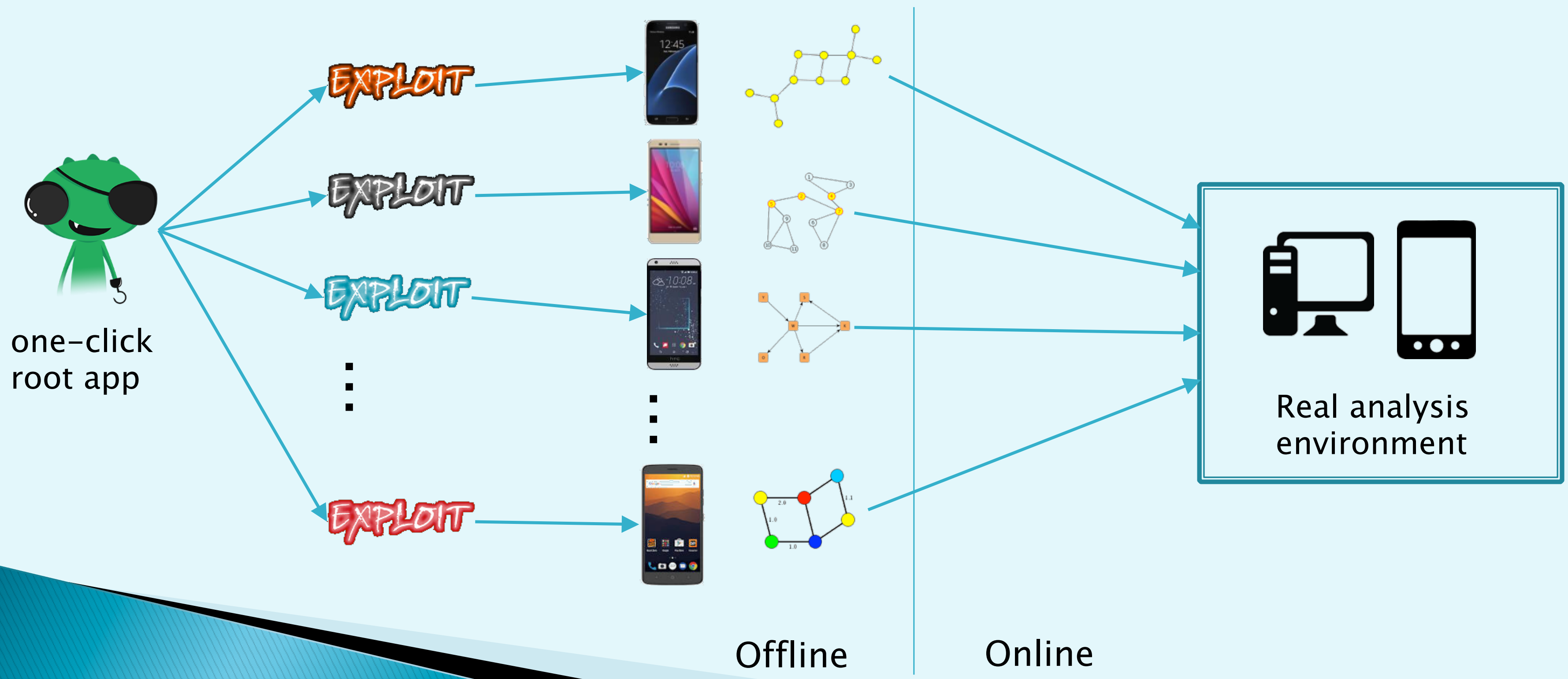
# Key observations

- One-click root apps [CCS' 15]
- Root exploits mostly reused

- Why not learn from them to understand exploit requirements?
  - <Exploit,  Device/Model/Version,  Program Preconditions>

[CCS 15] H. Zhang, D. She, Z. Qian,
Android Root and its Providers: A Double-Edged Sword

# System Overview – RootExplorer



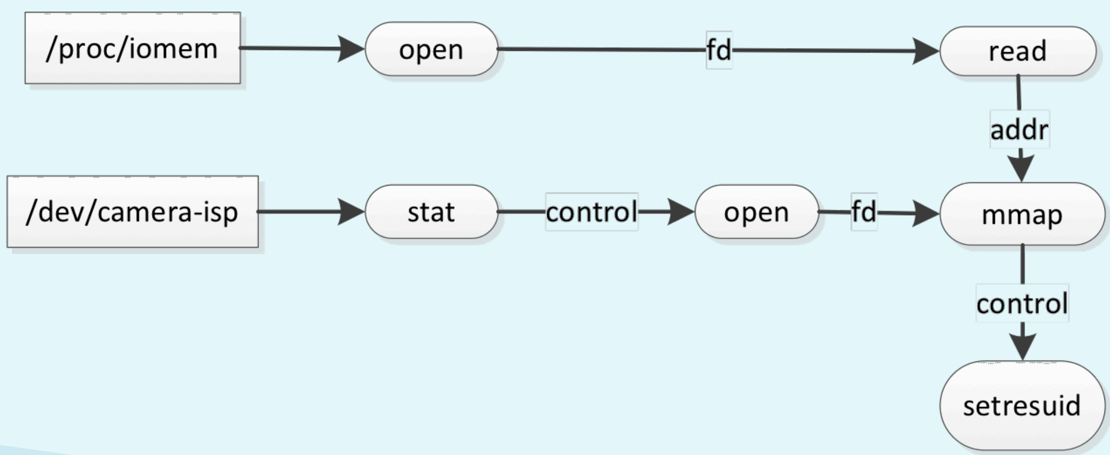
# Behavior Graph

- System call based behavior signature [1]
  - Root exploits interact with the OS through syscalls in unique ways
  - Data & control dependencies (robust to simple reordering)

[1] KOLBITSCH, C., COMPARETTI, P. M., KRUEGEL, C., KIRDA, E., ZHOU, X.Y., AND WANG, X. Effective and efficient malware detection at the end host. USENIX Security 2009

# Behavior Graph

- System call based behavior signature [1]
  - Root exploits interact with the OS through syscalls in unique ways
  - Data & control dependencies (robust to simple reordering)
  - Statically extracted (for a given path)




[1] KOLBITSCH, C., COMPARETTI, P. M., KRUEGEL, C., KIRDA, E., ZHOU, X.Y., AND WANG, X. Effective and efficient malware detection at the end host. USENIX Security 2009

# Extracting expected syscall returns

- Example:  what 'line' should be returned from readline()?

```
int fdIo = open("/proc/iomem");
// locate the kernel code offset in physical memory
while ((line = readline(fdIo)) > 0} {
    if((buf = strstr(line, "Kernel code")) != NULL) {
        addr = getAddress(buf);
        break;
    }
}
int getAddress(buf) {
    return atoi[buf-20];
}
```

readline() of file("/proc/iomem") cannot return NULL, **line** != NULL

strstr() of the line returned cannot return NULL → **line** has to contain "Kernel code"

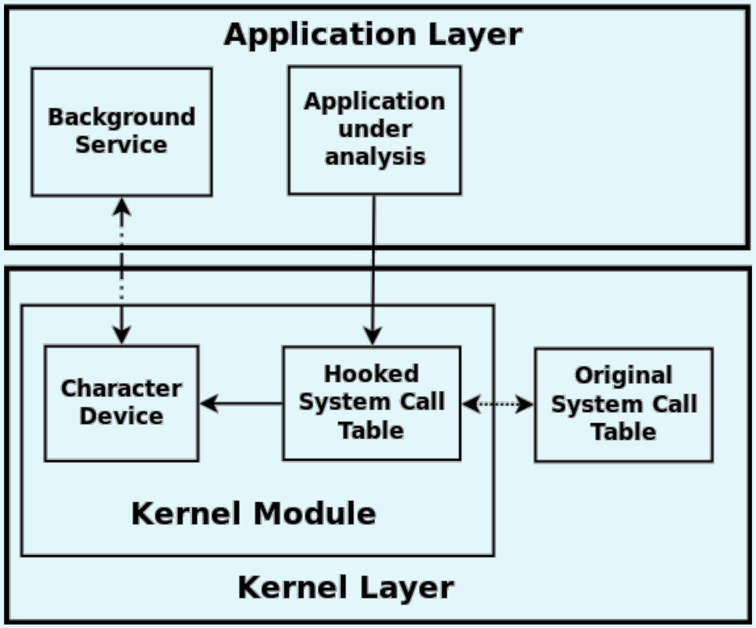
The buf has to have at least 20 preceding characters → **line** has to have 20 characters before "Kernel code"

Answer:   Line = "aaaaaaaaaaaaaaaaaaaaKernel code"

20 unconstrained characters

# Implementation

- Offline learning
  - Custom symbolic execution engine based on IDA pro
- Online learning
  - Loadable Kernel Module
    - System call hooking
  - Background Service
    - Decides what to do for syscall invocations
    - Monitor and match behavior graphs



# Experiment Setup

- Analysis environment: Android emulators
  - Loaded with real files, e.g., call logs, messages, contacts.
  - Real IMEI number, appropriate build.prop file

- App input generator
  - DroidBot: random user input and system events
  - Run every app for 10 minutes

# Evaluation Dataset

- Training set:
  - 167 exploit binaries (from 59 exploit families)
    - from a single one-click root app
- Testing set:
  - Positive samples
    - PoC from Internet, GODLESS malware, 7 one-click root apps
  - Grey samples
    - 1497 malware samples from an antivirus company
    - 2000 recently uploaded apps from unofficial Android markets
  - Negative samples
    - Top 1000 free apps from Google’s Play Store

# Detecting Exploit PoCs from the Internet

| Exploit | VirusTotal | RootExplorer |
|---|---|---|
| diag | 1/57 | ✓ |
| exynos | 4/57 | ✓ |
| pingpong | 1/57 | ✓ |
| towelroot | 3/57 | ✓ |

**Detection rate for debug compilation**

| Exploit | VirusTotal | RootExplorer |
|---|---|---|
| diag | 0/57 | ✓ |
| exynos | 1/57 | ✓ |
| pingpong | 0/57 | ✓ |
| towelroot | 1/57 | ✓ |

**Detection rate for obfuscated compilation**

- Downloaded and compiled 4 different PoC exploits from the Internet
- Compiled them with all debugging info
- Compiled them with obfuscation (using LLVM) and packed them with UPX

# Detecting GODLESS

- Run GODLESS against 5 different emulated devices and observed the following:

| | HTC J Butterfly | Fujitsu Arrows Z | Fujitsu Arrows X | Galaxy Note LTE | Samsung S3 |
|---|---|---|---|---|---|
| acdb | ✓ | ✗ | ✗ | ✗ | ✗ |
| hdcp | ✗ | ✓ | ✗ | ✗ | ✗ |
| msm_camera | ✓ | ✓ | ✓ | ✓ | ✓ |
| put_user | ✓ | ✓ | ✓ | ✓ | ✓ |
| fb_mem | ✓ | ✓ | ✓ | ✓ | ✓ |
| perf_swevent | ✗ | ✗ | ✓ | ✗ | ✗ |
| diag | ✗ | ✗ | ✗ | ✓ | ✗ |

# Detecting other one-click root apps

| One-Click Root Apps | Exploit |
|---|---|
| $O_1$ | /dev/camera-sysram |
| $O_2$ | /dev/graphics/fb5 |
| $O_3$ | /dev/exynos-mem |
| $O_4$ | /dev/camera-isp |
| $O_5$ | /dev/camera-isp |
| $O_6$ | /dev/camera-isp |
| $O_7$ | towelroot |

**Different one-click root apps choose to launch different exploits against a device**

# Detecting Malware in the Antivirus malware dataset and 3rd-party Android Markets

- 1497 malware samples, 2000 apps from 3rd party Android Market
  - Emulated Samsung S3 device
- Two apps flagged as containing root exploits
  - *Wifi Analyzer* from MoboMarket (pingpong root)
  - *Flashlight* app from the malware samples (camera-isp)
  - Confirmed by VirusTotal or Anti-virus company

# Conclusions

- *RootExplorer*, a system that learns from commercial-grade root exploits and detects similar root exploits in malware
- We show that this is an effective solution and has the potential to be used in practice

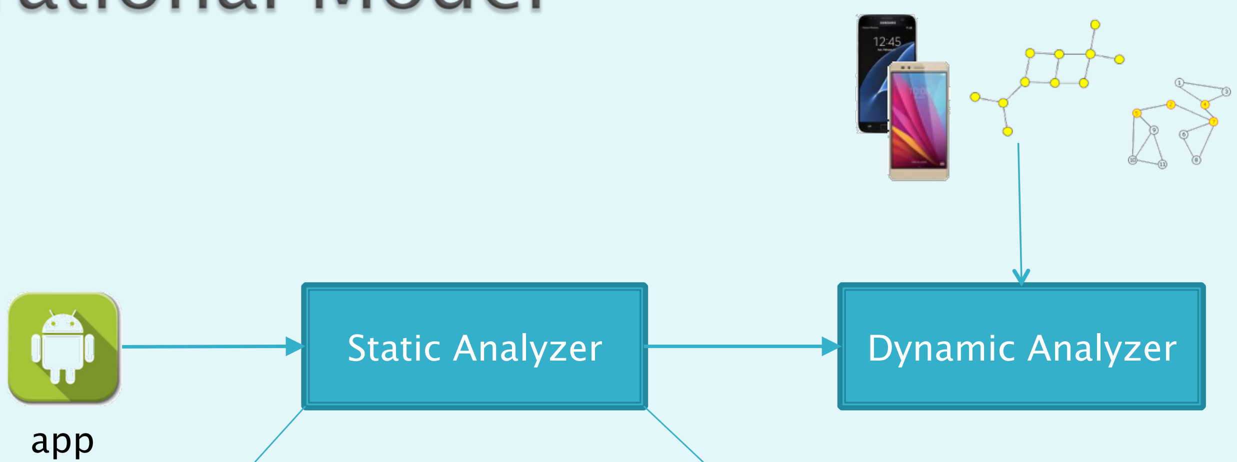
Questions?

zhiyunq@cs.ucr.edu

# Contributions

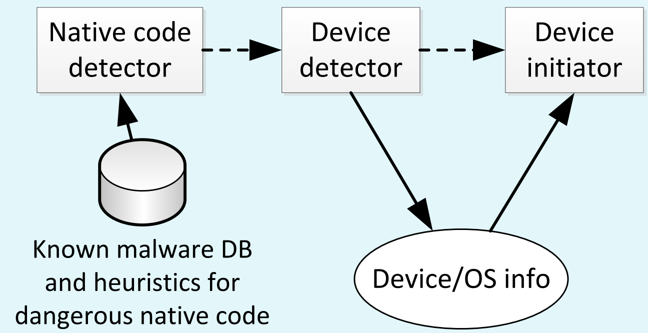
- **Detection of Android root exploits that target a diverse set of Android devices**
  - Based on commercial one-click root apps
  - What environmental features are sought
  - What pre-conditions need to be met for a root exploit to be triggered
- **Design and implement *RootExplorer***
  - Detects malwares that contain root exploits
  - Uses static analysis to understand the environment and attack profile of the exploits
  - It utilizes the learned information to construct proper analysis environments
- **Evaluation of *RootExplorer***
  - Successfully detects malwares with root exploits
  - Result in no false positives in our test case
  - Found an application in Android market that contains root exploits

# Operational Model




Pre-screening for native code, packing, dynamic code loading, device environment detector

# Static Analyzer




- **Native Code Detector**
  - App matches signatures of known malwares
  - App has any native code/dynamically loading native code
  - Custom heuristics to decide if it contains root exploits
- **Device Detector**
  - Parses APK and finds (A) methods that contain code that check the version of Android or the device name
  - Parses APK and finds (B) methods that contain code that run native executable code
  - If there is a path between (A) to (B), a new appropriate device is created
- **Device Initiator**
  - Modifies Android's property system accordingly
  - Modifies system files (/proc/version, etc) accordingly

# Dynamic Analyzer

- **Loadable Kernel Module**
  - Hooks every system call
  - Tracks only a specific app
  - Communicates with background service
- **Background Service**
  - Decides which action to take
  - Deploys decoy objects
  - Chooses behavior graphs and preconditions accordingly