Detecting Android Root Exploits by Learning from Root Providers

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

Android Framework

APPLICATIONS
- ALARM
- BROWSER
- CALCULATOR
- CALENDAR
- CAMERA
- CLOCK
- CONTACTS
- DIALER
- EMAIL
- HOME
- IM
- MEDIA PLAYER
- PHOTO ALBUM
- SMS/MMS
- VOICE DIAL

ANDROID FRAMEWORK
- CONTENT PROVIDERS
- MANAGERS (ACTIVITY, LOCATION, PACKAGE, NOTIFICATION, RESOURCE, TELEPHONY, WINDOW)
- VIEW SYSTEM

NATIVE LIBRARIES
- AUDIO MANAGER
- FREETYPE
- LBC
- MEDIA FRAMEWORK
- OPENGL/ES
- SQLITE
- SSL
- SURFACE MANAGER
- WEBKIT

ANDROID RUNTIME
- CORE LIBRARIES
- DALVIK VM

HAL
- AUDIO
- BLUETOOTH
- CAMERA
- DRM
- EXTERNAL STORAGE
- GRAPHICS
- INPUT
- MEDIA
- SENSORS
- TV

LINUX KERNEL
- DRIVERS (AUDIO, BINDER (IPC), BLUETOOTH, CAMERA, DISPLAY, KEYPAD, SHARED MEMORY, USB, WIFI)
- POWER MANAGEMENT
100+ CVEs →
Elevation of privilege:
1. only AOSP,
2. excluding closed-source components

Android Security Bulletins

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

Towelroot (CVE-2014-3153) Variants Count
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
    ```c
    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, version)
  - Program preconditions
    ```
    int fd = open("/dev/tty", O_RDWR);
    if (fd < 0)
      perror("open/tty");
    {
      /* read/write operations */
    }
    ```
  - Input generation problem (system call return values)

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

Key observations

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

Key observations

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

- Why not learn from them to understand exploit requirements?
  ◦ \(<\text{Exploit, Device/Model/Version, Program Preconditions}>\)
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)

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)

Effective and efficient malware detection at the end host. USENIX Security 2009
Example: what ‘line’ should be returned from readline()?

```c
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 = “aaaaaaaaaaaaaaaaaaaaaaaaaKernel 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

<table>
<thead>
<tr>
<th>Exploit</th>
<th>VirusTotal</th>
<th>RootExplorer</th>
</tr>
</thead>
<tbody>
<tr>
<td>diag</td>
<td>1/57</td>
<td>✓</td>
</tr>
<tr>
<td>exynos</td>
<td>4/57</td>
<td>✓</td>
</tr>
<tr>
<td>pingpong</td>
<td>1/57</td>
<td>✓</td>
</tr>
<tr>
<td>towelroot</td>
<td>3/57</td>
<td>✓</td>
</tr>
</tbody>
</table>

Detection rate for debug compilation

<table>
<thead>
<tr>
<th>Exploit</th>
<th>VirusTotal</th>
<th>RootExplorer</th>
</tr>
</thead>
<tbody>
<tr>
<td>diag</td>
<td>0/57</td>
<td>✓</td>
</tr>
<tr>
<td>exynos</td>
<td>1/57</td>
<td>✓</td>
</tr>
<tr>
<td>pingpong</td>
<td>0/57</td>
<td>✓</td>
</tr>
<tr>
<td>towelroot</td>
<td>1/57</td>
<td>✓</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>HTC J Butterfly</th>
<th>Fujitsu Arrows Z</th>
<th>Fujitsu Arrows X</th>
<th>Galaxy Note LTE</th>
<th>Samsung S3</th>
</tr>
</thead>
<tbody>
<tr>
<td>acdb</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>hdcp</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>msm_camera</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>put_user</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>fb_mem</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>perf_swevent</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>diag</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
</tr>
</tbody>
</table>
Different one-click root apps choose to launch different exploits against a device

<table>
<thead>
<tr>
<th>One-Click Root Apps</th>
<th>Exploit</th>
</tr>
</thead>
<tbody>
<tr>
<td>O₁</td>
<td>/dev/camera-sysram</td>
</tr>
<tr>
<td>O₂</td>
<td>/dev/graphics/fb5</td>
</tr>
<tr>
<td>O₃</td>
<td>/dev/exynos-mem</td>
</tr>
<tr>
<td>O₄</td>
<td>/dev/camera-isp</td>
</tr>
<tr>
<td>O₅</td>
<td>/dev/camera-isp</td>
</tr>
<tr>
<td>O₆</td>
<td>/dev/camera-isp</td>
</tr>
<tr>
<td>O₇</td>
<td>towelroot</td>
</tr>
</tbody>
</table>
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