An App Centric Approach for Vulnerability Discovery in Mobile Platforms

Dr. Zhiqiang Lin
zlin@cse.ohio-state.edu

Joint work w/ Omar Alrawi, Mohamed Elsabagh, Ruian Duan, Ryan Johnson, Ranjita Kasturi, Brendan Saltaformaggio, Angelos Starvro, Yinqian Zhang, Qingchuan Zhao, Chaoshun Zuo

04/21/2020
1. Introduction

2. FirmScope: Privilege-Escalation in Firmware

3. LeakScope: Leakage in Clouds

4. SkyWalker: n-day/0-day in Servers

5. Summary
Mobile Apps Have Significantly Changed Our Lives

1. Weather, News
2. Maps, Search
3. Social, Communication
4. Games, Sports
5. Music, Media, Entertainment
6. Shopping, Retail
7. Health, Fitness
8. Food, Drink
9. ...

Source: cloudxtension.com
Mobile Ecosystems: Apps, Web, and Clouds
Mobile Ecosystems: Apps, Web, and Clouds
The Mobile Backend as a Service (mBaaS) Clouds
The Mobile Backend as a Service (mBaaS) Clouds
Our Focal Point I: Uncovering Vulnerabilities in the Phones
Our Focal Point I: Uncovering Vulnerabilities in the Phones

Challenges: Tons of existing work; we have to look for new problems.
Our Focal Point I: Uncovering Vulnerabilities in the Phones

Our Target: focusing on preinstalled apps, since there is little work on this.
Our Focal Point II: Uncovering Vulnerabilities in the Clouds
Our Focal Point II: Uncovering Vulnerabilities in the Clouds

Challenges: No (source/binary) code of servers; Blackbox
**Our Focal Point II: Uncovering Vulnerabilities in the Clouds**

**Challenges:** No (source/binary) code of servers; Blackbox

**Key Approach:** **Identifying Vulnerabilities in Remote Cloud Servers (i.e., Online Services) via Automated Analysis of Mobile Apps**
Vulnerability I: Privilege Escalation in Preinstalled Apps

The Pre-installed apps

- Framework apps
- Vendor apps
- Carrier apps

Security Issue

- Preinstalled apps have the highest Privilege
- What if they are vulnerable?
Vulnerability II: Cloud Data Leakage

Verizon data of 6 million users leaked online

by Selena Larson  @selenalarson
July 12, 2017, 4:14 PM ET
Vulnerability II: Cloud Data Leakage

Verizon data of 6 million users leaked online
by Selena Larson  @selenalarson
July 12, 2017, 4:14 PM ET

Cyber-Safe
Pentagon exposed some of its data on Amazon server
by Selena Larson  @selenalarson
November 17, 2017, 12:03 PM ET
Vulnerability II: Cloud Data Leakage

Verizon data of 2018

Biggest Cloud Security Breaches in 2018

Cyber-Safe

Pentagon exposed some of its data on Amazon server

by Selena Larson  @selenalarson

November 17, 2017, 12:03 PM ET
Vulnerability II: Cloud Data Leakage

Cloud Security Concerns in 2018: Data Breaches, Security Misconfigurations, AI and Botnets

by Selena Larson  @selenalarson
November 17, 2017, 12:03 PM ET
Vulnerability III: Traditional Web Vulnerabilities and Beyond

Figure: The password reset activity of ShopClues (between 10 million and 50 million installs).
Vulnerability III: Traditional Web Vulnerabilities and Beyond

PUT /api/v9/forgotpassword?key=d12121c70dda5edfgd1df6633fdb36c0
HTTP/1.1
Content-Type: application/json
Connection: ... gzip
Content-Length: 73
{"user_email":"testmobileserver@gmail.com","key":"d12121c70dda5edfgd1df6633fdb36c0"}
Vulnerability III: Traditional Web Vulnerabilities and Beyond

There was an SQL injection vulnerability at this password reset interface
Our Efforts on **Vulnerability Discovery** w/ Mobile App Centric Analysis

8. Automatic Fingerprinting of Vulnerable BLE IoT Devices with Static UUIDs from Mobile Apps, In CCS 2019
Our Efforts on **Vulnerability Discovery** w/ Mobile App Centric Analysis

3. **SmartGen**: Exposing Server URLs of Mobile Apps With Selective Symbolic Execution. In WWW 2017
5. **The Unexpected Danger of UX Features**: A Case of Sensitive Data Leakage of Drivers in Ride-Hailing Services. In NDSS 2019
7. **The Betrayal At Cloud City**: An Empirical Analysis Of Cloud-Based Mobile Backends, In USENIX Security 2019
8. **Automatic Fingerprinting of Vulnerable BLE IoT Devices with Static UUIDs from Mobile Apps**, In CCS 2019
1. Introduction

2. FirmScope: Privilege-Escalation in Firmware

3. LeakScope: Leakage in Clouds

4. SkyWalker: n-day/0-day in Servers

5. Summary
Detecting Privilege Escalation in Preinstalled Apps: A Running Example

**Introduction**

Detecting Privilege Escalation in Preinstalled Apps: A Running Example

Challenges.

While it can also detect privacy leakage (e.g., personally identifiable information) in pre-installed apps, we exclude it from our scope since it is well-studied in prior research.

Android OS, and execute sensitive functionalities. For example, executing an attacker-controlled command in the context of a pre-installed factory-reset app (e.g., the Settings app) and re-addressed by this work. This example is simplified from a real-world pre-installed system app that can be exploited by an insecure interface, namely class Intent.

**System App** (/system/priv-app/update.apk)

```
public abstract class e
extends BroadcastReceiver {

    public int a(String arg1) {
        new k{((Context)this, arg1);
    }
}

public class a extends e {

    public void onReceive(Context arg1, Intent arg2) {
        int v1 = this.a(v0);
    }
}
```

**User App** (/data/app/attack.apk)

```
void exploit() {
    ...
    intent.putExtra("cmd","sh /path/to/payload.sh"
   );
    sendBroadcast(intent);
    ...
}
```

**Android Apps**

Android Framework

**Figure 1**

```
4 public k(Context arg1, String arg2) {
    this.h.i = arg2;
    arg1.registerReceiver(this);
}
```

```
5 public void onReceive(Context arg1, Intent arg2) {
    int v0 = arg2.getIntExtra(BatteryManager.EXTRA_PLUGGED, -1);
    if (v0 == BatteryManager.BATTERY_PLUGGED_AC) {
        run();
    }
}
```

```
6 public void run() {
    Log.d(this.h.x);
    Process v0 = Runtime.getRuntime().exec{ this.h.i }
}
```

```
7 public class k extends BroadcastReceiver {
    private j h = new j();
    ...
}
```

At a high level, this system app exposes enormous challenges that hinder their practicality, especially as the highly-privileged system app exhibiting a command injection vulnerability.

In addition, analyzing the vulnerabilities in insecure memory management vulnerabilities for insufficient input validation in interfaces.

**Figure 1**

```
1 public void onReceive(Context arg1, Intent arg2) {
    int v0 = arg2.getIntExtra("cmd","
        Int v1 = this.a(v0);
    }
}
```

```
2 public void run() {
    Log.d(this.h.x);
    Process v0 = Runtime.getRuntime().exec{ this.h.i }
}
```

```
3 v0 = arg2.getStringExtra("cmd");
    ...
}
```

```
5 public class k extends BroadcastReceiver {
    private j h = new j();
    ...
}
```

```
4 public k(Context arg1, String arg2) {
    this.h.i = arg2;
    arg1.registerReceiver(this);
}
```

```
6 public void onReceive(Context arg1, Intent arg2) {
    int v0 = arg2.getIntExtra(BatteryManager.EXTRA_PLUGGED, -1);
    if (v0 == BatteryManager.BATTERY_PLUGGED_AC) {
        run();
    }
}
```

```
7 public void run() {
    Log.d(this.h.x);
    Process v0 = Runtime.getRuntime().exec{ this.h.i }
}
```

```
8 System App (/system/priv-app/update.apk)

```

```
9 public abstract class e
extends BroadcastReceiver {

    public int a(String arg1) {
        new k{((Context)this, arg1);
    }
}
```

```
10 public class a extends e {

    public void onReceive(Context arg1, Intent arg2) {
        int v1 = this.a(v0);
    }
}
```

```
11 public class k extends BroadcastReceiver {
    private j h = new j();
    ...
}
```

```
12 public k(Context arg1, String arg2) {
    this.h.i = arg2;
    arg1.registerReceiver(this);
}
```

```
13 public void onReceive(Context arg1, Intent arg2) {
    int v0 = arg2.getIntExtra(BatteryManager.EXTRA_PLUGGED, -1);
    if (v0 == BatteryManager.BATTERY_PLUGGED_AC) {
        run();
    }
}
```

```
14 public void run() {
    Log.d(this.h.x);
    Process v0 = Runtime.getRuntime().exec{ this.h.i }
}
```

**Figure 1**

```
15 System App (/system/priv-app/update.apk)
```

```
16 public abstract class e
extends BroadcastReceiver {

    public int a(String arg1) {
        new k{((Context)this, arg1);
    }
}
```

```
17 public class a extends e {

    public void onReceive(Context arg1, Intent arg2) {
        int v1 = this.a(v0);
    }
}
```

```
18 public class k extends BroadcastReceiver {
    private j h = new j();
    ...
}
```

```
19 public k(Context arg1, String arg2) {
    this.h.i = arg2;
    arg1.registerReceiver(this);
}
```

```
20 public void onReceive(Context arg1, Intent arg2) {
    int v0 = arg2.getIntExtra(BatteryManager.EXTRA_PLUGGED, -1);
    if (v0 == BatteryManager.BATTERY_PLUGGED_AC) {
        run();
    }
}
```

```
21 public void run() {
    Log.d(this.h.x);
    Process v0 = Runtime.getRuntime().exec{ this.h.i }
}
```

**Android Apps**

Android Framework

**Figure 1**

```
1 public void onReceive(Context arg1, Intent arg2) {
    int v0 = arg2.getIntExtra("cmd","
        Int v1 = this.a(v0);
    }
}
```

```
2 public void run() {
    Log.d(this.h.x);
    Process v0 = Runtime.getRuntime().exec{ this.h.i }
}
```

```
3 v0 = arg2.getStringExtra("cmd");
    ...
}
```

```
4 public k(Context arg1, String arg2) {
    this.h.i = arg2;
    arg1.registerReceiver(this);
}
```
Introduction

While it can also detect privacy leakage (e.g., personally identifiable information) in pre-installed apps, we exclude any visible to the user in a pre-installed app and execute sensitive functionalities. For example, and [\[WUD("),Q-A33CRQWURO)ORZ CURVV-A33CRQWURO)ORZ]

... power and runtime performance. At a high level, these challenges mainly stem from (i) the semantics of Java

method... call which, in turn, executes the contents of the string as a command... sent from any app co-located on the device (step ... run callback in class... arrives at the... system... an attacker-controlled string is extracted from the incoming string to the constructor of... a... Intent... sent from any app co-located on the device (step... receives an... system... as the privileged... system app exposes... as the highly-privileged... command... run... field... a... Intent... details for clarity. At a high level, this system app exposes... a real-world pre-installed system app that can be exploited by an attacker. As... Amandroid [\([WUD("));... ], Amandroid [\([WUD("));... Droid [\([WUD("));... Challeng... FPG... Insecure Memory Management Vulnerabilities

Figure 1

3 Challenges and Key Insights

We start by giving a running example in

\n\textbf{System App (/system/priv-app/update.apk)}

public abstract class e extends BroadcastReceiver {
    public int a(String arg1) {
        new k((Context)this, arg1);
        ...
    }
}

public class k extends BroadcastReceiver {
    public k(Context arg1, String arg2) {
        this.h.i = arg2;
        ...
    }
    public void onReceive(Context arg1, Intent arg2) {
        int v0 = arg2.getStringExtra("cmd");
        int v1 = this.a(v0);
        ...
    }
}

\n\textbf{User App (/data/app/attack.apk)}

\begin{verbatim}
void exploit() {
    ...
    intent.putExtra("cmd", "sh /path/to/payload.sh");
    sendBroadcast(intent);
}
\end{verbatim}

\\begin{verbatim}
public class a extends e {
    public void onReceive(Context arg1, Intent arg2) {
        int v0 = arg2.getStringExtra("cmd");
        int v1 = this.a(v0);
        ...
    }
}
\end{verbatim}
Detecting Privilege Escalation in Preinstalled Apps: A Running Example

Figure 1: System App (/system/priv-app/update.apk)

```java
public abstract class e extends BroadcastReceiver {
    public int a(String arg1) {
        new k{((Context)this, arg1);
    }
}
```

```java
public class k extends BroadcastReceiver {
    private j h = new j();

    public k(Context arg1, String arg2) {
        h.i = arg1.registerReceiver(this);
    }

    public void onReceive(Context arg1, Intent arg2) {
        if (v0 == BatteryManager.BATTERY_PLUGGED_AC)
            run();
    }

    public void run() {
        Log.d(this.h.i);
        Process v0 = Runtime.getRuntime().exec( this.h.i );
    }
```

User App (/data/app/attack.apk)

```java
void exploit() {
    intent.putExtra(
        "cmd",
        "sh /path/to/payload.sh"
    );
    sendBroadcast(intent);
}
```

Android Apps

Android Framework

Introduction

LeakScope: Leakage in Clouds

FirmScope: Privilege-Escalation in Firmware

SkyWalker: n-day/0-day in Servers

Summary
Detecting Privilege Escalation in Preinstalled Apps: A Running Example

System App (/system/priv-app/update.apk)

```java
public abstract class e extends BroadcastReceiver {
    public int a(String arg1) {
        new k((Context)this, arg1);
    }
    ...
}
```

```java
public class k extends BroadcastReceiver {
    private int h = new int();
    ...
    public k(Context arg1, String arg2) {
        ...
        - arg1.registerReceiver(this);
    }
    ...
}
```

User App (/data/app/attack.apk)

```java
void exploit() {
    ...
    intent.putExtra("cmd", "sh /path/to/payload.sh");
    sendBroadcast(intent);
    ...
}
```

Android System

Android Apps

Android Framework

```
public void onReceive(Context arg1, Intent arg2) {
    if (v0 == BatteryManager.EXTRA_PLUGGED, -1);
    run();
}
```

```
public void run() {
    Log.d(this.h.x);
    Process v0 = Runtime.getRuntime().exec{ this.h.i };
    ...
}
```
Challenges

1. Tracking Flows Through Class Fields
2. Full vs. Partial Object Sensitivity
3. Handling the Android Runtime Framework APIs
4. Handling Asynchronous Callbacks
5. Handling Inter-component Communication
FirmScope: Privilege-Escalation in Firmware

LeakScope: Leakage in Clouds

SkyWalker: n-day/0-day in Servers

Summary

Preprocessing
- Disassemble & lift to IL
- Extract & canonicalize apps
- Unpack firmware

Firmware Image

Static Taint Analysis
- Build inter-procedural CFGs
- Build inter-procedural DFGs
- Perform custom taint analysis

Rules

Vulnerabilities
Dataset: 2,017 firmware form more than 100 Android vendors

<table>
<thead>
<tr>
<th>Vendor</th>
<th># Firmware</th>
<th># Apps</th>
<th>v4</th>
<th>v5</th>
<th>v6</th>
<th>v7</th>
<th>v8</th>
<th>v9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alcatel</td>
<td>31</td>
<td>4,390</td>
<td>15</td>
<td>3</td>
<td>9</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Alps</td>
<td>48</td>
<td>9,557</td>
<td>15</td>
<td>7</td>
<td>22</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>ASUS</td>
<td>93</td>
<td>17,944</td>
<td>16</td>
<td>24</td>
<td>21</td>
<td>19</td>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>BLU</td>
<td>132</td>
<td>16,355</td>
<td>32</td>
<td>17</td>
<td>58</td>
<td>20</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>Coolpad</td>
<td>29</td>
<td>3,429</td>
<td>12</td>
<td>7</td>
<td>3</td>
<td>1</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>Doogee</td>
<td>25</td>
<td>3,310</td>
<td>3</td>
<td>3</td>
<td>10</td>
<td>9</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Elephone</td>
<td>23</td>
<td>2,840</td>
<td>4</td>
<td>10</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Google</td>
<td>372</td>
<td>54,057</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>175</td>
<td>142</td>
<td>54</td>
</tr>
<tr>
<td>HTC</td>
<td>39</td>
<td>9,361</td>
<td>15</td>
<td>11</td>
<td>11</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Huawei</td>
<td>63</td>
<td>9,143</td>
<td>19</td>
<td>21</td>
<td>19</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Infinix</td>
<td>29</td>
<td>4,476</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>8</td>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>Lenovo</td>
<td>82</td>
<td>9,209</td>
<td>52</td>
<td>14</td>
<td>12</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Motorola</td>
<td>65</td>
<td>11,101</td>
<td>5</td>
<td>17</td>
<td>13</td>
<td>19</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>Panasonic</td>
<td>21</td>
<td>2,963</td>
<td>6</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Samsung</td>
<td>219</td>
<td>61,457</td>
<td>9</td>
<td>1</td>
<td>65</td>
<td>71</td>
<td>55</td>
<td>18</td>
</tr>
<tr>
<td>TCL</td>
<td>33</td>
<td>5,309</td>
<td>6</td>
<td>6</td>
<td>16</td>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Tecno</td>
<td>55</td>
<td>8,057</td>
<td>21</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>XBO</td>
<td>72</td>
<td>8,264</td>
<td>24</td>
<td>35</td>
<td>13</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Xiaomi</td>
<td>102</td>
<td>21,331</td>
<td>11</td>
<td>10</td>
<td>36</td>
<td>14</td>
<td>20</td>
<td>11</td>
</tr>
<tr>
<td>ZTE</td>
<td>73</td>
<td>10,557</td>
<td>12</td>
<td>13</td>
<td>24</td>
<td>24</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Other</td>
<td>411</td>
<td>58,232</td>
<td>126</td>
<td>82</td>
<td>83</td>
<td>55</td>
<td>65</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>2,017</td>
<td>331,342</td>
<td>403</td>
<td>291</td>
<td>441</td>
<td>449</td>
<td>350</td>
<td>83</td>
</tr>
</tbody>
</table>

We used [graph-tool](https://www.graph-tool.de) for efficient graph storage and in [37 KSLOC](https://example.com) of Cython, followed by performance benchmarks and comparisons with closely related work in [34 KSLOC](https://example.com) of Cython.

§5.1 Dataset and Experiment Setup

- We collected 2,017 firmware images from more than 100 Android vendors, each containing 150 GiB of RAM. We implemented a pipeline using GNU Parallel to analyze each firmware image in full, regardless of whether some of its apps might have appeared in other analyzed images. We deployed the code in 37 KSLOC of Cython, followed by performance benchmarks and comparisons with closely related work in 34 KSLOC of Cython.

<table>
<thead>
<tr>
<th>Vendor</th>
<th># Firmware</th>
<th># Apps</th>
<th>v4</th>
<th>v5</th>
<th>v6</th>
<th>v7</th>
<th>v8</th>
<th>v9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google</td>
<td>372</td>
<td>54,057</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>175</td>
<td>142</td>
<td>54</td>
</tr>
<tr>
<td>HTC</td>
<td>39</td>
<td>9,361</td>
<td>15</td>
<td>11</td>
<td>11</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Huawei</td>
<td>63</td>
<td>9,143</td>
<td>19</td>
<td>21</td>
<td>19</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Infinix</td>
<td>29</td>
<td>4,476</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>8</td>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>Lenovo</td>
<td>82</td>
<td>9,209</td>
<td>52</td>
<td>14</td>
<td>12</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Motorola</td>
<td>65</td>
<td>11,101</td>
<td>5</td>
<td>17</td>
<td>13</td>
<td>19</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>Panasonic</td>
<td>21</td>
<td>2,963</td>
<td>6</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Samsung</td>
<td>219</td>
<td>61,457</td>
<td>9</td>
<td>1</td>
<td>65</td>
<td>71</td>
<td>55</td>
<td>18</td>
</tr>
<tr>
<td>TCL</td>
<td>33</td>
<td>5,309</td>
<td>6</td>
<td>6</td>
<td>16</td>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Tecno</td>
<td>55</td>
<td>8,057</td>
<td>21</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>XBO</td>
<td>72</td>
<td>8,264</td>
<td>24</td>
<td>35</td>
<td>13</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Xiaomi</td>
<td>102</td>
<td>21,331</td>
<td>11</td>
<td>10</td>
<td>36</td>
<td>14</td>
<td>20</td>
<td>11</td>
</tr>
<tr>
<td>ZTE</td>
<td>73</td>
<td>10,557</td>
<td>12</td>
<td>13</td>
<td>24</td>
<td>24</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Other</td>
<td>411</td>
<td>58,232</td>
<td>126</td>
<td>82</td>
<td>83</td>
<td>55</td>
<td>65</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>2,017</td>
<td>331,342</td>
<td>403</td>
<td>291</td>
<td>441</td>
<td>449</td>
<td>350</td>
<td>83</td>
</tr>
</tbody>
</table>

We report the vulnerable app metadata, the vulnerable components, and the vulnerability semant. For instance, to detect the following privilege-escalation vulnerabilities:

- [Master Clear](https://example.com)
- [System Settings Modification](https://example.com)
- [Device Recording](https://example.com)
- [SMS](https://example.com)
- [Screen Recording](https://example.com)
- [Wireless Settings Modification](https://example.com)
- [System Commands](https://example.com)
- [Command Injection](https://example.com)
- [Data Injection](https://example.com)
- [Broadcast Intent](https://example.com)

We collected 331,342 apps with unique package names and versions. The details of the dataset are shown in Table 1.

### Table 1: Total 2,017 firmware images are shown.

<table>
<thead>
<tr>
<th>Vendor</th>
<th># Firmware</th>
<th># Apps</th>
<th>v4</th>
<th>v5</th>
<th>v6</th>
<th>v7</th>
<th>v8</th>
<th>v9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google</td>
<td>372</td>
<td>54,057</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>175</td>
<td>142</td>
<td>54</td>
</tr>
<tr>
<td>HTC</td>
<td>39</td>
<td>9,361</td>
<td>15</td>
<td>11</td>
<td>11</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Huawei</td>
<td>63</td>
<td>9,143</td>
<td>19</td>
<td>21</td>
<td>19</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Infinix</td>
<td>29</td>
<td>4,476</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>8</td>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>Lenovo</td>
<td>82</td>
<td>9,209</td>
<td>52</td>
<td>14</td>
<td>12</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Motorola</td>
<td>65</td>
<td>11,101</td>
<td>5</td>
<td>17</td>
<td>13</td>
<td>19</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>Panasonic</td>
<td>21</td>
<td>2,963</td>
<td>6</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Samsung</td>
<td>219</td>
<td>61,457</td>
<td>9</td>
<td>1</td>
<td>65</td>
<td>71</td>
<td>55</td>
<td>18</td>
</tr>
<tr>
<td>TCL</td>
<td>33</td>
<td>5,309</td>
<td>6</td>
<td>6</td>
<td>16</td>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Tecno</td>
<td>55</td>
<td>8,057</td>
<td>21</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>XBO</td>
<td>72</td>
<td>8,264</td>
<td>24</td>
<td>35</td>
<td>13</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Xiaomi</td>
<td>102</td>
<td>21,331</td>
<td>11</td>
<td>10</td>
<td>36</td>
<td>14</td>
<td>20</td>
<td>11</td>
</tr>
<tr>
<td>ZTE</td>
<td>73</td>
<td>10,557</td>
<td>12</td>
<td>13</td>
<td>24</td>
<td>24</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Other</td>
<td>411</td>
<td>58,232</td>
<td>126</td>
<td>82</td>
<td>83</td>
<td>55</td>
<td>65</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>2,017</td>
<td>331,342</td>
<td>403</td>
<td>291</td>
<td>441</td>
<td>449</td>
<td>350</td>
<td>83</td>
</tr>
</tbody>
</table>

Ensuring type-compatibility...
### Vulnerabilities: 3,483 vulnerabilities (147 confirmed CVEs)

<table>
<thead>
<tr>
<th>Vulnerability</th>
<th># Total</th>
<th># Unique</th>
<th>% Firmware</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command Injection</td>
<td>1,420</td>
<td>211</td>
<td>41%</td>
</tr>
<tr>
<td>Wireless Settings Modification</td>
<td>901</td>
<td>212</td>
<td>26%</td>
</tr>
<tr>
<td>SMS Injection</td>
<td>232</td>
<td>63</td>
<td>7%</td>
</tr>
<tr>
<td>Screen Recording</td>
<td>207</td>
<td>63</td>
<td>6%</td>
</tr>
<tr>
<td>Factory Reset</td>
<td>169</td>
<td>48</td>
<td>5%</td>
</tr>
<tr>
<td>System Properties Modification</td>
<td>160</td>
<td>54</td>
<td>5%</td>
</tr>
<tr>
<td>App (Un)Installation</td>
<td>153</td>
<td>54</td>
<td>5%</td>
</tr>
<tr>
<td>Full Logcat Leakage</td>
<td>110</td>
<td>85</td>
<td>4%</td>
</tr>
<tr>
<td>Microphone Audio Recording</td>
<td>61</td>
<td>38</td>
<td>2%</td>
</tr>
<tr>
<td>AT Command Injection</td>
<td>55</td>
<td>17</td>
<td>2%</td>
</tr>
<tr>
<td>Code Injection</td>
<td>15</td>
<td>5</td>
<td>1%</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>3,483</td>
<td>850</td>
<td>77%</td>
</tr>
</tbody>
</table>
Vulnerabilities: 3,483 vulnerabilities (147 confirmed CVEs))

<table>
<thead>
<tr>
<th>Vulnerability</th>
<th># Total</th>
<th># Unique</th>
<th>% Firmware</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command Injection</td>
<td>1,420</td>
<td>211</td>
<td>41%</td>
</tr>
<tr>
<td>Wireless Settings Modification</td>
<td>901</td>
<td>212</td>
<td>26%</td>
</tr>
<tr>
<td>SMS Injection</td>
<td>232</td>
<td>63</td>
<td>7%</td>
</tr>
<tr>
<td>Screen Recording</td>
<td>207</td>
<td>63</td>
<td>6%</td>
</tr>
<tr>
<td>Factory Reset</td>
<td>169</td>
<td>48</td>
<td>5%</td>
</tr>
<tr>
<td>System Properties Modification</td>
<td>160</td>
<td>54</td>
<td>5%</td>
</tr>
<tr>
<td>App (Un)Installation</td>
<td>153</td>
<td>54</td>
<td>5%</td>
</tr>
<tr>
<td>Full Logcat Leakage</td>
<td>110</td>
<td>85</td>
<td>4%</td>
</tr>
<tr>
<td>Microphone Audio Recording</td>
<td>61</td>
<td>38</td>
<td>2%</td>
</tr>
<tr>
<td>AT Command Injection</td>
<td>55</td>
<td>17</td>
<td>2%</td>
</tr>
<tr>
<td>Code Injection</td>
<td>15</td>
<td>5</td>
<td>1%</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>3,483</td>
<td>850</td>
<td>77%</td>
</tr>
</tbody>
</table>
Vulnerabilities: 3,483 vulnerabilities (147 confirmed CVEs))

<table>
<thead>
<tr>
<th>Vendor</th>
<th># Total Vuln. per firmware</th>
<th># Unique Vuln. per firmware</th>
<th>System Properties Modification</th>
<th>Microphone Audio Recording</th>
<th>Wireless Settings Modification</th>
<th>Full Logcat Leakage</th>
<th>Full Logcat Leakage</th>
<th>AT Command Injection</th>
<th>Code Injection</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALCATEL</td>
<td>1.3</td>
<td>23</td>
<td>15</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>34</td>
</tr>
<tr>
<td>Alps</td>
<td>1.1</td>
<td>20</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>22</td>
</tr>
<tr>
<td>ASUS</td>
<td>3.7</td>
<td>132</td>
<td>41</td>
<td>53</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>BLU</td>
<td>2.2</td>
<td>63</td>
<td>43</td>
<td>5</td>
<td>7</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Coolpad</td>
<td>3</td>
<td>54</td>
<td>22</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>Doogee</td>
<td>3.3</td>
<td>48</td>
<td>26</td>
<td>2</td>
<td>6</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Elephone</td>
<td>2.7</td>
<td>36</td>
<td>26</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Google</td>
<td>0.6</td>
<td>21</td>
<td>0</td>
<td>3</td>
<td>18</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>HTC</td>
<td>1.5</td>
<td>27</td>
<td>4</td>
<td>15</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Huawei</td>
<td>1.2</td>
<td>22</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>Infinix</td>
<td>0.6</td>
<td>8</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Lenovo</td>
<td>1.2</td>
<td>44</td>
<td>21</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Motorola</td>
<td>0.6</td>
<td>24</td>
<td>0</td>
<td>7</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Panasonic</td>
<td>2.3</td>
<td>34</td>
<td>27</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Samsung</td>
<td>3.3</td>
<td>178</td>
<td>16</td>
<td>50</td>
<td>1</td>
<td>15</td>
<td>29</td>
<td>0</td>
<td>30</td>
</tr>
<tr>
<td>TCL</td>
<td>1.4</td>
<td>33</td>
<td>20</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Tecno</td>
<td>1.2</td>
<td>28</td>
<td>17</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>XBO</td>
<td>2.2</td>
<td>37</td>
<td>29</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Xiaomi</td>
<td>2.2</td>
<td>118</td>
<td>46</td>
<td>27</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>ZTE</td>
<td>0.6</td>
<td>23</td>
<td>11</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>Other</td>
<td>1.7</td>
<td>239</td>
<td>82</td>
<td>50</td>
<td>20</td>
<td>23</td>
<td>6</td>
<td>20</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. Command Injection
2. Wireless Settings Modification
3. SMS Injection
4. Screen Recording
5. Factory Reset
6. System Properties Modification
7. App (Un)Installation
8. Full Logcat Leakage
9. Microphone Audio Recording
10. AT Command Injection
11. Code Injection
FirmScope vs. Other Frameworks

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>FlowDroid</th>
<th>Amandroid</th>
<th>DroidSafe</th>
<th>FirmScope</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FP</td>
<td>FN</td>
<td>FP</td>
<td>FN</td>
</tr>
<tr>
<td>Aliasing</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AndroidSpecific</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>ArraysAndLists</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Callbacks</td>
<td>2</td>
<td>1</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>EmulatorDetection</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>FieldAndObjectSensitivity</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>GeneralJava</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>InterComponentComm.</td>
<td>0</td>
<td>8</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Lifecycle</td>
<td>0</td>
<td>9</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>Reflection</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Threading</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>Total (lower is better)</td>
<td>10</td>
<td>24</td>
<td>20</td>
<td>30</td>
</tr>
</tbody>
</table>

We avoid discussing the overall precision and recall of the comparants. DroidBench bundles an extensive set of benchmarks implemented as DIALDRoid-Bench, a representative sample of 30 minimalist apps. (DroidSafe took more than 20 min, FlowDroid and Amandroid timed out on 12 apps, Amandroid on 9 apps, and DroidSafe on 7 apps.) Apps took about $8 \text{ min}$ on average despite the large number of firmware images in the market. Although this captures results on the standard Android 9 Settings app (one of the largest apps that come pre-installed on every device) where results show that FirmScope outperforms prior work in terms of both detection power and scalability to large apps.

We used the same configuration related studies [20] by setting a maximum execution time of 30 min (details are given in [21]). Apps took about $8 \text{ min}$ on average for all uncaught exceptions (e.g., using java.lang.Thread.setDefaultUncaughtExceptionHandler(...), which do not model all the semantics of throwing exceptions. For certain constructs that are not handled (e.g., nested reflection and writes. Writing a tainted value to a container eventually results in tainting the entire container (e.g., once the container is modified).

The total start-to-finish runtime of images was approximately 5.3. In terms of per firmware server, this start-to-finish time is inverse proportional to the number of servers. In practice, especially given the large number of firmware images and apps in the market, marked solutions, incurring only a few FPs is paramount (note that we had only three cases were FPs, coming to a total of 451 findings out of 483 (12.95%). This accounts for FirmScope's low number of FPs is paramount. An important number of FPs in our real-world findings. We computed the worst-case scenario false discovery rate (FDR) in our findings as sample reference cases to arrive at a rough estimate of the number of FPs in our real-world findings. We computed the worst-case scenario false discovery rate (FDR) in our findings.
Outline

1. Introduction
2. FirmScope: Privilege-Escalation in Firmware
3. LeakScope: Leakage in Clouds
4. SkyWalker: n-day/0-day in Servers
5. Summary
Vulnerability II: Data Leakage is Essentially an Access Control Problem
Vulnerability II: Data Leakage is Essentially an Access Control Problem
Vulnerability II: Data Leakage is Essentially an Access Control Problem
Vulnerability II: Data Leakage is Essentially an Access Control Problem

Authentication
Authorization

Password

Token

Bob

Alice

Bob's Data

Alice's Data

Approved

[Diagram showing authentication and authorization processes with Bob and Alice, passwords, tokens, and data access control]

Bob

Alice

Bob's Data

Alice's Data

Password

Token

[Diagram showing authentication and authorization processes with Bob and Alice, passwords, tokens, and data access control]
How Do Mobile Apps and mBaaS Cloud Communicate
How Do Mobile Apps and mBaaS Cloud Communicate

Authentication

User A

Credential A (App Key)

Authorization

Cloud Resources
How Do Mobile Apps and mBaaS Cloud Communicate

Authentication

User A

Credential A (App Key)

Authorization

Cloud Resources
How Do Mobile Apps and mBaaS Cloud Communicate

**Authentication**

User A

Credential A
(App Key)

Cloud Resources

**Authorization**
How Do Mobile Apps and mBaaS Cloud Communicate

Authentication

User A
Credential A (App Key)

User B
Credential B (App Key)

Authorization

SkyWalker: n-day/0-day in Servers
LeakScope: Leakage in Clouds
FirmScope: Privilege-Escalation in Firmware

Introduction

Summary
How Do Mobile Apps and mBaaS Cloud Communicate

Authentication

Developer/Administrator

Authorization

Cloud Resources
How Do Mobile Apps and mBaaS Cloud Communicate

Authentication

Authorization

Developer/Administrator

Root Key

Cloud Resources
How Do Mobile Apps and mBaaS Cloud Communicate

Authentication

Developer/Administrator

Root Key

Authorization

Cloud Resources
How Do Mobile Apps and mBaaS Cloud Communicate

Authentication

Developer/Administrator

Root Key

Authorization

Cloud Resources
How Do Mobile Apps and mBaaS Cloud Communicate

Authentication

User A
Credential A (App Key)

Developer/Administrator
Root Key

User B
Credential B (App Key)

Authorization

Cloud Resources
Our Discovery of Why Cloud Data was Leaked

The Root Causes of the Cloud Data Leakage

1. Misuse of Various Keys in Authentication
   - Microsoft Azure Storage
   - Microsoft Azure Notification Hubs
2. Misconfiguration of User Permissions in Authorization
   - Google Firebase
   - Amazon AWS

Credential A (App Key)
Developer/Administrator
Authentication
Credential B (App Key)
User B
Authorization
Cloud Resources
User A

Authentication
Authorization
Our Discovery of Why Cloud Data was Leaked

The Root Causes of the Cloud Data Leakage

1. Misuse of Various Keys in Authentication
   - Microsoft Azure Storage
   - Microsoft Azure Notification Hubs
   - Amazon AWS

2. Misconfiguration of User Permissions in Authorization
   - Google Firebase
   - Amazon AWS
Our Discovery of Why Cloud Data was Leaked

The Root Causes of the Cloud Data Leakage

1. Misuse of Various Keys in Authentication
   - Microsoft Azure Storage
   - Microsoft Azure Notification Hubs
   - Amazon AWS

2. Misconfiguration of User Permissions in Authorization
   - Google Firebase
   - Amazon AWS
## Misuse of Root Keys in Microsoft Azure

<table>
<thead>
<tr>
<th>Service</th>
<th>Key Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Azure Storage</td>
<td>Account Key</td>
<td>DefaultEndpointsProtocol=https; AccountName=<em>;AccountKey=</em></td>
</tr>
<tr>
<td></td>
<td>SAS</td>
<td>https://<em>.blob.core.windows.net/</em>?sv=<em>&amp;st=</em>&amp;se=<em>&amp;sr=b&amp;sp=rw&amp;sip=</em>&amp;spr=https&amp;sig=*</td>
</tr>
<tr>
<td>Notification</td>
<td>Listening Key</td>
<td>Endpoint=sb://<em>.servicebus.windows.net/; SharedAccessKeyName= DefaultListenSharedAccessSignature; SharedAccessKey=</em></td>
</tr>
<tr>
<td>Hub</td>
<td>Full Access Key</td>
<td>Endpoint=sb://<em>.servicebus.windows.net/; SharedAccessKeyName= DefaultFullSharedAccessSignature; SharedAccessKey=</em></td>
</tr>
</tbody>
</table>

**Table:** Examples of the Keys Used in Mobile App Development with Microsoft Azure Cloud. We use symbol * to anonymize those sensitive data in the keys.
Misconfiguration of User Permissions in Google Firebase

```json
{
    "rules": {
        "users": {
            "$uid": {
                "read": "$uid === auth.uid",
                "write": "$uid === auth.uid"
            }
        }
    }
}
```

Figure: A Correct Firebase Authorization Rule
Misconfiguration of User Permissions in Google Firebase

Figure: A Correct Firebase Authorization Rule

```
{
  "rules": {
    "users": {
      "$uid": {
        "read": "$uid === auth.uid",
        "write": "$uid === auth.uid"
      }
    }
  }
}
```

Figure: Two Misconfigured Firebase Authorization Rules

```
(a) (b)
```

```
{
  "rules": {
    "read": "auth != null",
    "write": "auth != null"
  }
}
{
  "rules": {
    "read": true,
    "write": true
  }
}
(a) (b)
```
How to Identify a Token vs the Token: A Running Example

GET /api/v1//users/\[21690\]/notifications?in_app_token=e67315b35aa38d4ac8cac3cd9c7f88ae7f576d373f HTTP/1.1
Host: api.w****.com
Connection: close

HTTP/1.1 200 OK
Cache-Control: max-age=0, private, must-revalidate
Content-Type: application/json
ETag: W/"5319d96924bb6d0a761b5f13b248919c"
Server: nginx/1.6.2
X-Request-Id: 5775d45e-cc3b-4665-8bc6-c2c7a2c9180d
X-Runtime: 0.027840
Content-Length: 191
Connection: Close


Alice’s first request and response message after login
How to Identify a Token vs the Token: A Running Example

Bob’s first request and response message after login
How to Identify a Token vs the Token: A Running Example

Alice’s first request message after login

```
GET /api/v1/users/21690/notifications?in_app_token=21691/notifications?in_app_token=21690/notifications?in_app_token=e67315b35aa38d4ac8cac3cd9c7f88ae7f576d373f HTTP/1.1
Host: api.w****.com
Connection: close
```

Bob’s first request message after login

```
GET /api/v1/users/21691/notifications?in_app_token=fb153b7d8c0a0c6ac841d7bfbd9446de627c642858 HTTP/1.1
Host: api.w****.com
Connection: close
```
How to Identify a Token vs the Token: A Running Example

GET /api/v1/users/21690/notifications?in_app_token=e67315b35aa38d4ac8cac3cd9c7f88ae7f576d373f HTTP/1.1
Host: api.w****.com
Connection: close

GET /api/v1/users/21691/notifications?in_app_token=fb153b7d8c0a0c6ac841d7bfbd9446de627c642858 HTTP/1.1
Host: api.w****.com
Connection: close
How to Identify a Token vs the Token: A Running Example

```
GET /api/v1/users/21690/notifications?in_app_token=e67315b35aa38d4ac8cac3cd9c7f88ae7f576d373f HTTP/1.1
Host: api.w****.com
Connection: close
```

```
GET /api/v1/users/21691/notifications?in_app_token=fbl53b7d8c0a0c6ac841d7bfbd9446de627c642858 HTTP/1.1
Host: api.w****.com
Connection: close
```
How to Identify a Token vs the Token: A Running Example

GET /api/v1/users/21691/notifications?in_app_token=e67315b35aa38d4ac8cac3cd9c7f88ae7f576d373f HTTP/1.1
Host: api.w****.com
Connection: close
How to Identify a Token vs the Token: A Running Example

GET /api/v1//users/21691/notifications?in_app_token=e67315b35aa3
8d4ac8c3cd9c7f88a7f576d373f HTTP/1.1
Host: api.w****.com
Connection: close

HTTP/1.1 200 OK
Cache-Control: max-age=0, private, must-revalidate
Content-Type: application/json
ETag: W/"6ee365b32e7f3e145d5c74778ea243cd"
Server: nginx/1.6.2
X-Request-Id: 4970cafb-9438-4a70-96e0-ca2f789f0d5d
X-Runtime: 0.022889
Content-Length: 192
Connection: Close


Alice reads Bob’s notifications
Problem Statement: How to automatically detect the cloud leakage at scale

Challenges

1. How to systematically identify various keys used by mobile apps (Cloud APIs)
2. How to identify the relevant key strings that are used by mobile apps (String Analysis)
3. How to design an obfuscation-resilient approach to identify cloud APIs and key strings of our interest (Obfuscation-Resilient)
4. How to determine the vulnerability without leaking sensitive information in the cloud (Vulnerability Confirmation)
Our **LeakScope** [Oakland’19]
## Distributions of the Testing Apps

<table>
<thead>
<tr>
<th></th>
<th>Total #Apps</th>
<th>%</th>
<th>Non-Obfuscated #Apps</th>
<th>%</th>
<th>Obfuscated #Apps</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>w/ Cloud API</td>
<td>107,081</td>
<td>-</td>
<td>85,357</td>
<td>79.71</td>
<td>21,724</td>
<td>20.29</td>
</tr>
<tr>
<td>w/ AWS only</td>
<td>4,799</td>
<td>4.48</td>
<td>4,548</td>
<td>5.33</td>
<td>251</td>
<td>1.16</td>
</tr>
<tr>
<td>w/ Azure only</td>
<td>899</td>
<td>0.84</td>
<td>720</td>
<td>0.84</td>
<td>179</td>
<td>0.82</td>
</tr>
<tr>
<td>w/ Firebase only</td>
<td>99,186</td>
<td>92.63</td>
<td>78,475</td>
<td>91.94</td>
<td>20,711</td>
<td>95.34</td>
</tr>
<tr>
<td>w/ AWS &amp; Azure</td>
<td>3</td>
<td>0.00</td>
<td>2</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
</tr>
<tr>
<td>w/ AWS &amp; Firebase</td>
<td>1,973</td>
<td>1.84</td>
<td>1,427</td>
<td>1.67</td>
<td>546</td>
<td>2.51</td>
</tr>
<tr>
<td>w/ Azure &amp; Firebase</td>
<td>210</td>
<td>0.20</td>
<td>174</td>
<td>0.20</td>
<td>36</td>
<td>0.17</td>
</tr>
<tr>
<td>w/ Three Services</td>
<td>11</td>
<td>0.01</td>
<td>11</td>
<td>0.01</td>
<td>0</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Total: 24 / 38
### Distributions of the Testing Apps

<table>
<thead>
<tr>
<th></th>
<th>Total #Apps</th>
<th>%</th>
<th>Non-Obfuscated #Apps</th>
<th>%</th>
<th>Obfuscated #Apps</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>w/ Cloud API</td>
<td>107,081</td>
<td>-</td>
<td>85,357</td>
<td>79.71</td>
<td>21,724</td>
<td>20.29</td>
</tr>
<tr>
<td>w/ AWS only</td>
<td>4,799</td>
<td>4.48</td>
<td>4,548</td>
<td>5.33</td>
<td>251</td>
<td>1.16</td>
</tr>
<tr>
<td>w/ Azure only</td>
<td>899</td>
<td>0.84</td>
<td>720</td>
<td>0.84</td>
<td>179</td>
<td>0.82</td>
</tr>
<tr>
<td>w/ Firebase only</td>
<td>99,186</td>
<td>92.63</td>
<td>78,475</td>
<td>91.94</td>
<td>20,711</td>
<td>95.34</td>
</tr>
<tr>
<td>w/ AWS &amp; Azure</td>
<td>3</td>
<td>0.00</td>
<td>2</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
</tr>
<tr>
<td>w/ AWS &amp; Firebase</td>
<td>1,973</td>
<td>1.84</td>
<td>1,427</td>
<td>1.67</td>
<td>546</td>
<td>2.51</td>
</tr>
<tr>
<td>w/ Azure &amp; Firebase</td>
<td>210</td>
<td>0.20</td>
<td>174</td>
<td>0.20</td>
<td>36</td>
<td>0.17</td>
</tr>
<tr>
<td>w/ Three Services</td>
<td>11</td>
<td>0.01</td>
<td>11</td>
<td>0.01</td>
<td>0</td>
<td>0.00</td>
</tr>
</tbody>
</table>
### Statistics of The Detected Vulnerabilities

<table>
<thead>
<tr>
<th>The Root Cause</th>
<th>Non-Obfuscated</th>
<th>Obfuscated</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#Apps</td>
<td>%</td>
</tr>
<tr>
<td><strong>Azure</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Account Key Misuse</td>
<td>85</td>
<td>9.37</td>
</tr>
<tr>
<td>Full Access Key Misuse</td>
<td>101</td>
<td>11.14</td>
</tr>
<tr>
<td><strong>AWS</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Root key Misuse</td>
<td>477</td>
<td>7.97</td>
</tr>
<tr>
<td>“Open” S3 Storage</td>
<td>916</td>
<td>15.30</td>
</tr>
<tr>
<td><strong>Firebase</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>“Open” Database</td>
<td>5,166</td>
<td>6.45</td>
</tr>
<tr>
<td>No Permission Check</td>
<td>6,855</td>
<td>8.56</td>
</tr>
</tbody>
</table>
# Statistics of The Detected Vulnerabilities

<table>
<thead>
<tr>
<th>The Root Cause</th>
<th>Non-Obfuscated</th>
<th>Obfuscated</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#Apps</td>
<td>%</td>
</tr>
<tr>
<td>Azure</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Account Key Misuse</td>
<td>85</td>
<td>9.37</td>
</tr>
<tr>
<td>Full Access Key Misuse</td>
<td>101</td>
<td>11.14</td>
</tr>
<tr>
<td>AWS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Root key Misuse</td>
<td>477</td>
<td>7.97</td>
</tr>
<tr>
<td>“Open” S3 Storage</td>
<td>916</td>
<td>15.30</td>
</tr>
<tr>
<td>Firebase</td>
<td></td>
<td></td>
</tr>
<tr>
<td>“Open” Database</td>
<td>5,166</td>
<td>6.45</td>
</tr>
<tr>
<td>No Permission Check</td>
<td>6,855</td>
<td>8.56</td>
</tr>
</tbody>
</table>
### Statistics of The Detected Vulnerabilities

<table>
<thead>
<tr>
<th>#Downloads</th>
<th># Non-Vulnerable Apps</th>
<th># Vulnerable Apps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Azure</td>
<td>AWS</td>
</tr>
<tr>
<td>1,000,000,000 - 5,000,000,000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>500,000,000 - 1,000,000,000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>100,000,000 - 500,000,000</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>50,000,000 - 100,000,000</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>10,000,000 - 50,000,000</td>
<td>2</td>
<td>35</td>
</tr>
<tr>
<td>5,000,000 - 10,000,000</td>
<td>3</td>
<td>32</td>
</tr>
<tr>
<td>1,000,000 - 5,000,000</td>
<td>16</td>
<td>136</td>
</tr>
<tr>
<td>500,000 - 1,000,000</td>
<td>10</td>
<td>105</td>
</tr>
<tr>
<td>100,000 - 500,000</td>
<td>65</td>
<td>356</td>
</tr>
<tr>
<td>50,000 - 100,000</td>
<td>42</td>
<td>249</td>
</tr>
<tr>
<td>10,000 - 50,000</td>
<td>167</td>
<td>679</td>
</tr>
<tr>
<td>5,000 - 10,000</td>
<td>82</td>
<td>369</td>
</tr>
<tr>
<td>1,000 - 5,000</td>
<td>272</td>
<td>976</td>
</tr>
<tr>
<td>0 - 1,000</td>
<td>464</td>
<td>3,844</td>
</tr>
</tbody>
</table>

Table: The Number of Apps that Have Used the Cloud APIs in Each of The Accumulated Download Category.
## Statistics of The Detected Vulnerabilities

<table>
<thead>
<tr>
<th>#Downloads</th>
<th># Non-Vulnerable Apps</th>
<th># Vulnerable Apps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Azure</td>
<td>AWS</td>
</tr>
<tr>
<td>$1,000,000,000 – 5,000,000,000$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$500,000,000 – 1,000,000,000$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$100,000,000 – 500,000,000$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$50,000,000 – 100,000,000$</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>$10,000,000 – 50,000,000$</td>
<td>2</td>
<td>35</td>
</tr>
<tr>
<td>$5,000,000 – 10,000,000$</td>
<td>3</td>
<td>32</td>
</tr>
<tr>
<td>$1,000,000 – 5,000,000$</td>
<td>16</td>
<td>136</td>
</tr>
<tr>
<td>$500,000 – 1,000,000$</td>
<td>10</td>
<td>105</td>
</tr>
<tr>
<td>$100,000 – 500,000$</td>
<td>65</td>
<td>356</td>
</tr>
<tr>
<td>$50,000 – 100,000$</td>
<td>42</td>
<td>249</td>
</tr>
<tr>
<td>$10,000 – 50,000$</td>
<td>167</td>
<td>679</td>
</tr>
<tr>
<td>$5,000 – 10,000$</td>
<td>82</td>
<td>369</td>
</tr>
<tr>
<td>$1,000 – 5,000$</td>
<td>272</td>
<td>976</td>
</tr>
<tr>
<td>$0 – 1,000$</td>
<td>464</td>
<td>3,844</td>
</tr>
</tbody>
</table>

Table: The Number of Apps that Have Used the Cloud APIs in Each of The Accumulated Download Category.
Engaging with the Cloud Providers

**Disclosed** all the vulnerabilities we have identified. Cloud providers further notified the app developers.

1. **Microsoft** immediately corrected the wrong documentation
2. **Google** plans to provide more user-friendly SDKs when configuring user permissions in authorization.
3. **Amazon** added new permission checks with its S3 storage in November 2017 (two weeks before we disclosed our details to them)
Engaging with the Cloud Providers

Disclaimer on the use of account key

* MODIFY THIS!

* Stores the storage connection string.

* Only use Shared Key authentication (Account Key) for testing purposes!

* Your account name and account key, which give full read/write access to the associated Storage account,

* will be distributed to every person that downloads your app.

* This is not a good practice as you risk having your key compromised by untrusted clients.

* Please consult following documents to understand and use Shared Access Signatures instead.

* https://docs.microsoft.com/en-us/rest/api/storageservices/delegating-access-with-a-shared-access-signature

* https://docs.microsoft.com/en-us/azure/storage/common/storage-dotnet-shared-access-signature-part-1

* public static final String storageConnectionString = "DefaultEndpointsProtocol=https;"
Engaging with the Cloud Providers

Google’s Update

1. “The big additions on our side are tools for local emulation and writing tests against the database products including their security rules, which we expect to have a marked improvement on the ability of customers to test and validate security rules.”

2. “Additionally, we have alerting for customers (sent every few weeks) for anyone using the Realtime Database or Cloud Firestore with open rules.”

3. “We’re exploring more options, but those are a start.”
Generating the **IPs/URLs** of Servers for Vulnerability Discovery

```java
package com.shopclues;

class y implements View$OnClickListener {
    EditText b;
    ...
    public void onClick(View arg5) {
        String v0 = this.b.getText().toString().trim();
        if (v0.equalsIgnoreCase("")) {
            Toast.makeText(this.a, "Email Id should not be empty", 1).show();
        } else if (!al.a(v0)) {
            Toast.makeText(this.a, "The email entered is not a valid email", 1).show();
        } else if (al.b(this.a)) {
            this.a.c = new ac(this.a, v0);
            this.a.c.execute(new Void[0]);
        } else {
            Toast.makeText(this.a, "Please check your internet connection", 1).show();
        }
    }
}
```
Generating the **IPs/URLs** of Servers for Vulnerability Discovery

**Various Constraints in Mobile Apps**

1. Two text-box’s inputs need to be equivalent
2. The “age” needs to be greater than 18
3. A “zip code” needs to be a five digit sequence
4. A “phone number” needs to be a phone number
5. A file name extension needs to be some type (e.g., .jpg)
6. ...
Our SMARTGen [WWW’17]

- Automated
- Systematic
- Scalable
Our **SMARTGen** [WWW’17]

- Static analysis
- Selective symbolic execution
- Dynamic analysis
SQL Injection Fuzzing

- "SELECT PG_SLEEP(5);",
- "SELECT PG_SLEEP(10);"
- "'WAITFOR DELAY '0:0:5'--"
- ";SELECT COUNT(*) FROM SYSIBM.SYSTABLES"
Malicious URL Detection [WWW’17]

- Malware sites
- Compromised sites
- VirusTotal provides services for these detections
Other Security Vulnerabilities

- **n-day vulnerabilities** by fingerprinting servers (e.g., Hypervisors, OS, Networking daemons, and libraries such as vulnerable SSL/TLS, Apache, and PHP)

- **0-day vulnerabilities**, e.g., Cross Site Scripting (XSS), XML External Entity (XXE) Vulnerabilities, in addition to SQLi
Figure: **SkyWalker** Overview. Phase 1 (Binary Analysis) extracts backend URLs through a dynamic binary instrumentation technique. Phase 2 labels backends into first-party, third-party, and hybrid based on IP and public available information. Phase 3 discovers and fingerprints the backend services to collect cloud layer information. Phase 4 (vulnerability analysis) uses the fingerprints and correlates them with public vulnerabilities to identify vulnerable backends.
Implementation

1. For the **binary analysis** tool implementation, we relied on Soot [soo], SmartGen [ZL17], FlowDroid [ARF⁺], Z3-str [ZZG], and Xposed [xpo] with custom code written in Java (7,000 lines of code) and Python (900 lines of code).

2. For our **backend labeling** implementation, we relied on Team Cymru IP-to-ASN [Cym18], MaxMind Geolocation [Max18], Alexa ranking [ale], ipcat list [Gal19], and Domaintools WHOIS [Dom18] with custom code written in Python (480 lines of code).

3. For **fingerprinting**, we relied on the Nessus scanner and commercial plugins [Sec18], sqlmap [sql], and Acunetix [Acu18]. We used Nessus plugins and custom Python code (1010 lines of code) to perform the vulnerability analysis.
Overall Experimental Results

<table>
<thead>
<tr>
<th>Category</th>
<th># Mobile Apps</th>
<th># OS</th>
<th># SS</th>
<th># AS</th>
<th># CS</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Books &amp; Reference</td>
<td>332</td>
<td>15</td>
<td>49</td>
<td>55</td>
<td>71</td>
<td>190</td>
</tr>
<tr>
<td>Business</td>
<td>145</td>
<td>5</td>
<td>22</td>
<td>10</td>
<td>37</td>
<td>74</td>
</tr>
<tr>
<td>Entertainment</td>
<td>1,177</td>
<td>36</td>
<td>108</td>
<td>158</td>
<td>170</td>
<td>472</td>
</tr>
<tr>
<td>Games</td>
<td>1,283</td>
<td>34</td>
<td>81</td>
<td>147</td>
<td>106</td>
<td>368</td>
</tr>
<tr>
<td>Lifestyle</td>
<td>363</td>
<td>20</td>
<td>50</td>
<td>79</td>
<td>72</td>
<td>221</td>
</tr>
<tr>
<td>Misc</td>
<td>199</td>
<td>6</td>
<td>21</td>
<td>45</td>
<td>46</td>
<td>118</td>
</tr>
<tr>
<td>Tools</td>
<td>792</td>
<td>19</td>
<td>84</td>
<td>184</td>
<td>115</td>
<td>402</td>
</tr>
<tr>
<td>Video &amp; Audio</td>
<td>689</td>
<td>24</td>
<td>46</td>
<td>89</td>
<td>98</td>
<td>257</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>4,980</strong></td>
<td><strong>121</strong></td>
<td><strong>356</strong></td>
<td><strong>655</strong></td>
<td><strong>506</strong></td>
<td><strong>1,638</strong></td>
</tr>
</tbody>
</table>

Table: An overview of the vulnerable mobile apps per category along with the raw counts of vulnerabilities: Operating System (OS), Software Services (SS), Application Software (AS), Communication Services (CS)
## Overall Experimental Results

<table>
<thead>
<tr>
<th>Category</th>
<th># Mobile Apps</th>
<th># OS</th>
<th>SS</th>
<th>AS</th>
<th>CS</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Books &amp; Reference</td>
<td>332</td>
<td>15</td>
<td>49</td>
<td>55</td>
<td>71</td>
<td>190</td>
</tr>
<tr>
<td>Business</td>
<td>145</td>
<td>5</td>
<td>22</td>
<td>10</td>
<td>37</td>
<td>74</td>
</tr>
<tr>
<td>Entertainment</td>
<td>1,177</td>
<td>36</td>
<td>108</td>
<td>158</td>
<td>170</td>
<td>472</td>
</tr>
<tr>
<td>Games</td>
<td>1,283</td>
<td>34</td>
<td>81</td>
<td>147</td>
<td>106</td>
<td>368</td>
</tr>
<tr>
<td>Lifestyle</td>
<td>363</td>
<td>20</td>
<td>50</td>
<td>79</td>
<td>72</td>
<td>221</td>
</tr>
<tr>
<td>Misc</td>
<td>199</td>
<td>6</td>
<td>21</td>
<td>45</td>
<td>46</td>
<td>118</td>
</tr>
<tr>
<td>Tools</td>
<td>792</td>
<td>19</td>
<td>84</td>
<td>184</td>
<td>115</td>
<td>402</td>
</tr>
<tr>
<td>Video &amp; Audio</td>
<td>689</td>
<td>24</td>
<td>46</td>
<td>89</td>
<td>98</td>
<td>257</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>4,980</strong></td>
<td><strong>121</strong></td>
<td><strong>356</strong></td>
<td><strong>655</strong></td>
<td><strong>506</strong></td>
<td><strong>1,638</strong></td>
</tr>
</tbody>
</table>

Table: An overview of the vulnerable mobile apps per category along with the raw counts of vulnerabilities: Operating System (OS), Software Services (SS), Application Software (AS), Communication Services (CS)
Other Statistics

<table>
<thead>
<tr>
<th>Comp.</th>
<th>Vulnerability (Top 3)</th>
<th>#Apps</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS</td>
<td>Expired Lifecycle for Linux OS (various)</td>
<td>124</td>
</tr>
<tr>
<td></td>
<td>Windows Server RCE (MS15-034)</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>Expired Lifecycle for Windows Server</td>
<td>9</td>
</tr>
<tr>
<td>SS</td>
<td>Vulnerable PHP Version</td>
<td>357</td>
</tr>
<tr>
<td></td>
<td>Expired Lifecycle for Web Server (various)</td>
<td>181</td>
</tr>
<tr>
<td></td>
<td>Vulnerable Apache Version</td>
<td>76</td>
</tr>
<tr>
<td>AS</td>
<td>XSS (various)</td>
<td>262</td>
</tr>
<tr>
<td></td>
<td>SQLi (various)</td>
<td>160</td>
</tr>
<tr>
<td></td>
<td>XXE (various)</td>
<td>86</td>
</tr>
<tr>
<td>CS</td>
<td>Support for Vulnerable SSL Version 2 and 3</td>
<td>997</td>
</tr>
<tr>
<td></td>
<td>OpenSSH Bypass (CVE-2015-5600)</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>Vulnerable OpenSSL (various)</td>
<td>15</td>
</tr>
</tbody>
</table>

Table: The top three vulnerabilities found per cloud layer along with the number of affected mobile apps.
## Other Statistics

<table>
<thead>
<tr>
<th># Installs</th>
<th># Apps</th>
<th># SQLi</th>
<th># XSS</th>
<th># XXE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1B</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>500M</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>100M</td>
<td>116</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>50M</td>
<td>131</td>
<td>4</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>10M</td>
<td>1,049</td>
<td>25</td>
<td>85</td>
<td>15</td>
</tr>
<tr>
<td>5M</td>
<td>1,047</td>
<td>54</td>
<td>89</td>
<td>9</td>
</tr>
<tr>
<td>1M</td>
<td>2,621</td>
<td>132</td>
<td>316</td>
<td>17</td>
</tr>
</tbody>
</table>

Table: The number of 0-day vulnerabilities found per install category.
Outline

1. Introduction
2. FirmScope: Privilege-Escalation in Firmware
3. LeakScope: Leakage in Clouds
4. SkyWalker: n-day/0-day in Servers
5. Summary
**FirmScope**

- **A fully automated** tool to detect privilege escalation vulnerabilities
- Can also detect **PII leakage**

---

**Experimental Result w/ 2K firmware**

- 331,342 pre-installed apps
- 3,483 vulnerabilities
- 147 confirmed CVEs
FirmScope [USENIX Security 2020]

**FirmScope**
- A **fully automated** tool to detect privilege escalation vulnerabilities
- Can also detect **PII leakage**

**Experimental Result w/ 2K firmware**
- 331,342 pre-installed apps
- 3,483 vulnerabilities
- 147 confirmed CVEs

Alternatives: github.com/secure-software-engineering/FlowDroid, github.com/MIT-PAC/droidsafe-src
**LeakScope** [Oakland’19]

- A static analysis to identify server side data leakage vulnerabilities
- It performs cloud API identification, string value analysis to identify the vulnerabilities

**Experimental Result w/ 100K apps**
- 15,098 apps' cloud servers are vulnerable
- 200 Azure, 1,600 AWS, 13,200 Firebase
- Source code is available at [github.com/OSUSecLab/LeakScope](https://github.com/OSUSecLab/LeakScope)
SkyWalker [USENIX Security 2019]

SkyWalker

▶ A fully automated mobile app backend analysis framework
▶ Can be used to vet various security vulnerabilities in mobile backends

Experimental Result w/ 4,980 apps
▶ 983 N-day and 655 0-day spanning across the software layers (OS, software services, communication, and web apps) of cloud backends
▶ https://MobileBackend.vet
An App Centric Approach for Vulnerability Discovery in Mobile Platforms

Dr. Zhiqiang Lin
zlin@cse.ohio-state.edu

Joint work w/ Omar Alrawi, Mohamed Elsabagh, Ruian Duan, Ryan Johnson, Ranjita Kasturi, Brendan Saltaformaggio, Angelos Starvro, Yinqian Zhang, Qingchuan Zhao, Chaoshun Zuo

04/21/2020


Tenable Security, *Nessus Professional*,

*A framework for analyzing and transforming java and android apps*,
[https://sable.github.io/soot/](https://sable.github.io/soot/).


