Automatically Uncovering Vulnerabilities in Clouds From Mobile Apps

Dr. Zhiqiang Lin
zlin@cse.ohio-state.edu
Joint work w/ Omar Alrawi, Ruian Duan, Ranjita Kasturi, Brendan Saltaformaggio, Yinqian Zhang, Qingchuan Zhao, Chaoshun Zuo

Nov 7th, 2019
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 Apps, Web, and Clouds
Mobile Apps, Web, and Clouds
The Mobile Backend as a Service (mBaaS) Clouds

Cloud App Engines

Mobile Backend

Databases

APIs running On the Cloud

Client

Server

Mobile Backend Client Library

Push Notification
The Mobile Backend as a Service (mBaaS) Clouds
Our Focal Point: Uncovering Server Side Vulnerabilities
Our Focal Point: Uncovering Server Side Vulnerabilities
Our Focal Point: Uncovering Server Side Vulnerabilities
Our Focal Point: Uncovering Server Side Vulnerabilities

Challenges: No (source/binary) code of servers; Blackbox
Our Focal Point: Uncovering Server Side Vulnerabilities

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: Cloud Data Leakage

Verizon data of 6 million users leaked online

by Selena Larson  @selenalarson

July 12, 2017: 4:14 PM ET
Vulnerability I: 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 I: 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 I: Cloud Data Leakage

Source: CNN Business, March 29, 2018

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

by Selena Larson  @selenalarson
November 17, 2017, 12:03 PM ET
Vulnerability I: Data Leakage is Essentially an Access Control Problem
Vulnerability I: Data Leakage is Essentially an Access Control Problem
Vulnerability I: Data Leakage is Essentially an Access Control Problem
Vulnerability I: Data Leakage is Essentially an Access Control Problem
Vulnerability II: Traditional Web Vulnerabilities and Beyond

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

There was an SQL injection vulnerability at this password reset interface:

PUT /api/v9/forgotpassword?key=d12121c70dda5edfgd1df6633fdb36c0
HTTP/1.1
Content-Type: application/json
Connection: close
User-Agent: Dalvik/1.6.0 (Linux; Android 4.2)
Host: sm.shopclues.com
Accept-Encoding: gzip
Content-Length: 73

{"user_email":"testmobileserver@gmail.com","key":"d12121c70dda5edfgd1df6633fdb36c0"}
Vulnerability II: Traditional Web Vulnerabilities and Beyond

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

1. **SMV-Hunter: Large Scale, Automated Detection of SSL/TLS Man-in-the-Middle Vulnerabilities in Android Apps.** In *NDSS 2014*
2. **You Shouldn’t Collect My Secrets: Thwarting Sensitive Keystroke Leakage in Mobile IME Apps.** In *USENIX Security 2015*
3. **AutoForge: Automatic Forgery of Cryptographically Consistent Messages to Identify Security Vulnerabilities in Mobile Services.** In *NDSS 2016*
4. **SmartGen: Exposing Server URLs of Mobile Apps With Selective Symbolic Execution.** In *WWW 2017*
5. **AuthScope: Towards Automatic Identification of Vulnerable Authorizations in Online Services.** In *CCS 2017*
6. **The Unexpected Danger of UX Features: A Case of Sensitive Data Leakage of Drivers in Ride-Hailing Services.** In *NDSS 2019*
7. **LeakScope: Why Does Your Data Leak? Uncovering the Data Leakage in Cloud from Mobile Apps.** In *Oakland 2019*
8. **The Betrayal At Cloud City: An Empirical Analysis Of Cloud-Based Mobile Backends,** In *USENIX Security 2019*
9. **Automatic Fingerprinting of Vulnerable BLE IoT Devices with Static UUIDs from Mobile Apps,** In *CCS 2019*
Our Efforts on **Vulnerability Discovery** w/ Mobile App Analysis

4. **SmartGen**: Exposing Server URLs of Mobile Apps With Selective Symbolic Execution. In WWW 2017
5. **AuthScope**: Towards Automatic Identification of Vulnerable Authorizations in Online Services. In CCS 2017
7. **LeakScope**: Why Does Your Data Leak? Uncovering the Data Leakage in Cloud from Mobile Apps. In Oakland 2019
8. **The Betrayal At Cloud City**: An Empirical Analysis Of Cloud-Based Mobile Backends, In USENIX Security 2019
9. **Automatic Fingerprinting of Vulnerable BLE IoT Devices with Static UUIDs from Mobile Apps**, In CCS 2019
Our Efforts on **Vulnerability Discovery** w/ Mobile App Analysis

4. **SmartGen**: Exposing Server URLs of Mobile Apps With Selective Symbolic Execution. In WWW 2017
5. **AuthScope**: Towards Automatic Identification of Vulnerable Authorizations in Online Services. In CCS 2017
6. **The Unexpected Danger of UX Features**: A Case of Sensitive Data Leakage of Drivers in Ride-Hailing Services. In NDSS 2019
7. **LeakScope**: Why Does Your Data Leak? Uncovering the Data Leakage in Cloud from Mobile Apps. In Oakland 2019
9. **Automatic Fingerprinting of Vulnerable BLE IoT Devices with Static UUIDs from Mobile Apps**, In CCS 2019
Vulnerability I: Data Leakage is Essentially an Access Control Problem

Authentication
Authorization

Password
Token

Bob's Data
Alice's Data
How Do Mobile Apps and mBaaS Cloud Communicate

Authentication

Authorization

User A

Cloud Resources
How Do Mobile Apps and mBaaS Cloud Communicate

Authentication

Credential A
(App Key)

Authorization

User A

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)

Authorization

Cloud Resources
How Do Mobile Apps and mBaaS Cloud Communicate

Authentication

User A

Credential A (App Key)

Authorization

User B

Credential B (App Key)

Cloud Resources
How Do Mobile Apps and mBaaS Cloud Communicate

Authentication

Authorization

Developer/Administrator

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

Authorization

- Developer/Administrator
  - Root Key
- Cloud Resources
How Do Mobile Apps and mBaaS Cloud Communicate

Authentication

Authorization

Developer/Administrator

Root Key

Cloud Resources

11 / 46
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

 User A
 User B
 Root Key
Our Discovery of Why Cloud Data was Leaked
Our Discovery of Why Cloud Data was Leaked

Authentication

Credential A (App Key)
User A
Developer/Administrator
Root Key
Credential B (App Key)
User B

Authorization

Cloud Resources

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 Hub</td>
<td>Listening Key</td>
<td>Endpoint=sb://<em>.servicebus.windows.net/; SharedAccessKeyName= DefaultListenSharedAccessSignature; SharedAccessKey=</em></td>
</tr>
<tr>
<td></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

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

Figure: A Correct Firebase Authorization Rule

{  
  "rules": {  
    "users": {  
      "$uid": {  
        "read": "true",  
        "write": "true"  
      }  
    }  
  }  
}

(a) (b)

Figure: Two Misconfigured Firebase Authorization Rules
How to Identify a Token vs the Token: A Running Example

GET /api/v1/users/21690/notifications?in_app_token=e67315b35aa38d4ac8ac3cd9c7f88ae7f576d373f 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

GET /api/v1//users/21691/notifications?in_app_token=fb153b7d8c0a
0c6ac841d7bfbd9446de627c642858 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


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=e67315b35aa38d4ac8c4c3cd9c7f88ae7f576d373f 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=f815b7d8c0a0c6ac841d7bfbd9446de627c642858 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=fb153b7d8c0a0c6ac841d7bfb9446de627c642858 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=e67315b35aa38d4ac8cac3cd97f88ae7f576d373f 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/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=e67315b35aa38d4ac8ac3cd9c7f88ae7f576d373f 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
Introducing **AuthScope** [CCS’17]

1. Alice’s Request₁
2. Alice’s Request₂
3. Bob’s Request

**Field Recognition and Substitution**

1. Alice’s Request₁
2. Alice’s Request₂
3. Bob’s Request

**Response Message Labeling**

4. Alice’s Response₁
5. Alice’s Response₂
6. Bob’s Response

**Server Response Messages for the Field-Substituted Request**

7. Field-Substituted Alice’s Request Messages (for Bob)

8. Server Response Messages for the Field-Substituted Request

**Post-Authentication Message Generation**

Smartphone

Man-in-the-Middle Proxy

Cloud
Experiment Setup and Overall Result

- Top 10% Apps (200,000) of Google Play Store (March 2017)
- 33,950 contains Facebook library
- 4,838 Facebook Login
## Experiment Setup and Overall Result

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Sigma$ # Apps</td>
<td>4,838</td>
</tr>
<tr>
<td>$\Sigma$ Time to perform the test (hours)</td>
<td>562.4</td>
</tr>
<tr>
<td>$\Sigma$ # Request messages</td>
<td>3,220,886</td>
</tr>
<tr>
<td>$\Sigma$ # HTTP Messages</td>
<td>178,539</td>
</tr>
<tr>
<td>$\Sigma$ Size of the messages (G-bytes)</td>
<td>59.2</td>
</tr>
<tr>
<td>$\Sigma$ Time of activity exploration before authentication (hours)</td>
<td>169.9</td>
</tr>
<tr>
<td>$\Sigma$ # Explored activities after authentication</td>
<td>20,704</td>
</tr>
<tr>
<td>$\Sigma$ # Mutated fields</td>
<td>57,736</td>
</tr>
<tr>
<td>$\Sigma$ # Vulnerable apps</td>
<td>306</td>
</tr>
</tbody>
</table>

Table: Overall Experimental Result.
## Experiment Setup and Overall Result

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Σ # Apps</td>
<td>4,838</td>
</tr>
<tr>
<td>Σ Time to perform the test (hours)</td>
<td>562.4</td>
</tr>
<tr>
<td>Σ # Request messages</td>
<td>3,220,886</td>
</tr>
<tr>
<td>Σ # HTTP Messages</td>
<td>178,539</td>
</tr>
<tr>
<td>Σ Size of the messages (G-bytes)</td>
<td>59.2</td>
</tr>
<tr>
<td>Σ Time of activity exploration before authentication (hours)</td>
<td>169.9</td>
</tr>
<tr>
<td>Σ # Explored activities after authentication</td>
<td>20,704</td>
</tr>
<tr>
<td>Σ # Mutated fields</td>
<td>57,736</td>
</tr>
<tr>
<td>Σ # Vulnerable apps</td>
<td>306</td>
</tr>
</tbody>
</table>

Table: Overall Experimental Result.
## Experiment Setup and Overall Result

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Σ # Apps</td>
<td>4,838</td>
</tr>
<tr>
<td>Σ Time to perform the test (hours)</td>
<td>562.4</td>
</tr>
<tr>
<td>Σ # Request messages</td>
<td>3,220,886</td>
</tr>
<tr>
<td>Σ # HTTP Messages</td>
<td>178,539</td>
</tr>
<tr>
<td>Σ Size of the messages (G-bytes)</td>
<td>59.2</td>
</tr>
<tr>
<td>Σ Time of activity exploration before authentication (hours)</td>
<td>169.9</td>
</tr>
<tr>
<td>Σ # Explored activities after authentication</td>
<td>20,704</td>
</tr>
<tr>
<td>Σ # Mutated fields</td>
<td>57,736</td>
</tr>
<tr>
<td>Σ # Vulnerable apps</td>
<td>306</td>
</tr>
</tbody>
</table>

Table: Overall Experimental Result.
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)
Introducing **LeakScope** [Oakland’19]

**Authentication**
- Credential A (App Key)
- Root Key
- Developer/Administrator
- User A
- User B

**Authorization**
- APKs
- Cloud API Identification

**Cloud API**
- String Value Analysis

**Vulnerability Identification**
- SDK APIs
- Vulnerabilities
## 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>
## Distributions of the Testing Apps

<table>
<thead>
<tr>
<th>Services Combination</th>
<th>Total #Apps</th>
<th>Non-Obfuscated #Apps</th>
<th>Obfuscated #Apps</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>w/ Cloud API</strong></td>
<td>107,081</td>
<td>85,357</td>
<td>21,724</td>
</tr>
<tr>
<td>w/ AWS only</td>
<td>4,799</td>
<td>4,548</td>
<td>251</td>
</tr>
<tr>
<td>w/ Azure only</td>
<td>899</td>
<td>720</td>
<td>179</td>
</tr>
<tr>
<td>w/ Firebase only</td>
<td>99,186</td>
<td>78,475</td>
<td>20,711</td>
</tr>
<tr>
<td>w/ AWS &amp; Azure</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>w/ AWS &amp; Firebase</td>
<td>1,973</td>
<td>1,427</td>
<td>546</td>
</tr>
<tr>
<td>w/ Azure &amp; Firebase</td>
<td>210</td>
<td>174</td>
<td>36</td>
</tr>
<tr>
<td>w/ Three Services</td>
<td>11</td>
<td>11</td>
<td>0</td>
</tr>
</tbody>
</table>
## Statistics of The Detected Vulnerabilities

<table>
<thead>
<tr>
<th>The Root Cause</th>
<th>Non-Obfuscated</th>
<th>Obfuscation</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><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>#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

```
# master (#65)

 seguler committed on Dec 21, 2017

1 parent 191f888 commit d99c3b49312e77c22c911c8f55a37be9947454c4

Showing 1 changed file with 7 additions and 0 deletions.

7  microsoft-azure-storage-samples/src/com/microsoft/azure/storage/samples/MainActivity.java

```
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
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. ...
Introducing \textbf{SMARTGen [WWW'17]}

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

Static Analysis
- Static analysis
- Selective symbolic execution
- Dynamic analysis

![Diagram showing the process of analyzing APKs and generating request messages](image)

- APK
- Building ECG
- Extracting Path Constraints
- Solving the Constraints
- Request Message Generation
- Runtime Instrumentation
- Real Phone
- Request Messages

**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
Introducing SkyWalker [USENIX Security 2019]

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 [Sec18a], 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>Vulnerabilities</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># OS</td>
<td># SS</td>
<td># AS</td>
</tr>
<tr>
<td>Books &amp; Reference</td>
<td>332</td>
<td>15</td>
<td>49</td>
</tr>
<tr>
<td>Business</td>
<td>145</td>
<td>5</td>
<td>22</td>
</tr>
<tr>
<td>Entertainment</td>
<td>1,177</td>
<td>36</td>
<td>108</td>
</tr>
<tr>
<td>Games</td>
<td>1,283</td>
<td>34</td>
<td>81</td>
</tr>
<tr>
<td>Lifestyle</td>
<td>363</td>
<td>20</td>
<td>50</td>
</tr>
<tr>
<td>Misc</td>
<td>199</td>
<td>6</td>
<td>21</td>
</tr>
<tr>
<td>Tools</td>
<td>792</td>
<td>19</td>
<td>84</td>
</tr>
<tr>
<td>Video &amp; Audio</td>
<td>689</td>
<td>24</td>
<td>46</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>4,980</td>
<td>121</td>
<td>356</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></td>
<td>Expired Lifecycle for Linux OS (various)</td>
<td>124</td>
</tr>
<tr>
<td>OS</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></td>
<td>Vulnerable PHP Version</td>
<td>357</td>
</tr>
<tr>
<td>SS</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></td>
<td>XSS (various)</td>
<td>262</td>
</tr>
<tr>
<td>AS</td>
<td>SQLi (various)</td>
<td>160</td>
</tr>
<tr>
<td></td>
<td>XXE (various)</td>
<td>86</td>
</tr>
<tr>
<td></td>
<td>Support for Vulnerable SSL Version 2 and 3</td>
<td>997</td>
</tr>
<tr>
<td>CS</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.
Recommendations for Mobile App Developers (4D)

1. First, developers should **delegate** as much of the backend functionality to reputable third-party backends and minimize the number of features and functions their backend needs to support.

2. Second, developers should **dedicate** personnel to manage and maintain their backends including the routine maintenance of *OS*, *SS* and *CS*, and timely fixes of known vulnerabilities affecting their cloud backends and mobile apps using patching tools [Sec18b, AK, DBJ⁺].

3. Third, developers should **develop** an audit plan and a mitigation plan and be familiar with it to execute during an incident or vulnerability disclosure.

4. Finally, developers should utilize **defense** tools like web app firewalls (WAF), DDoS mitigation, and crawler/scanner blockers to protect from internet scanners, DDoS threats, and web app attacks (SQLi, authentication bypass, etc.).
**Related Work**

1. **Protocol Reverse Engineering.** A large body of research focusing on protocol reverse engineering [Bed, MLK+06, CKW07, CS07, WMKK08, LJXZ08, MWKK09, CPKS09]

Vulnerability Discovery w/ Mobile Apps:

- **Client Side:** TaintDroid [EGC+10], PiOS [EKKV11], CHEX [LLW+12], SMV-Hunter [SSG+14].
- **Server Side:** AutoForge [ZWWL16], SmartGen [ZL17], AuthScope [ZZL17], and LeakScope [ZLZ19].

2 Misconfiguration Vulnerability Identification: FIREMAN [YMS+06], ConfErr [KUC08], ConfAid [AF10], SPEX [XZH+13].
Related Work

Measurement Studies

- Durumeric et al. [DKBH] conducted a comprehensive internet-wide study of the HTTPS certificate ecosystem. Later, Durumeric et al. [DLK+] carried out a similar internet-wide study for the Heartbleed vulnerability [hea].
- Perez-Botero et al. [PBSL] presented an in-depth study characterizing hypervisor vulnerabilities in cloud services.
- Zuo et al. [ZL17] proposed a system to identify mobile app URLs and examine their reputation with public blacklists to detect malicious apps.
AutHScoPe [CCS’17]

AutHScoPe

- A fully automated automated system to analyze mobile apps, and reveal post-authentication messages
- Can be used to test various post authentication vulnerabilities

Experimental Result w/ 5K apps

- Each app has large number of (e.g., 1,000,000+) installs
- There were close to 300 apps whose servers contain vulnerable authorization services
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
- Responsible disclosures were made to the cloud providers
**SMARTGen** [WWW’17]

- **SMARTGen**
  - A fully automated mobile app execution framework via symbolic execution
  - Can be used to test various security vulnerabilities in mobile systems

**Experimental Result w/ 5,000 apps**
- 340 SQL Injection Vulnerabilities
- These apps actually talk to 2,071 phishing sites, 3,722 malware sites, and 3,228 malicious sites
A fully automated mobile app backend analysis framework

Can be used to vet various security vulnerabilities in mobile backends

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
Future Works: Security in IoT (e.g., BLEScope [CCS'19])
Future Works: Security in IoT (e.g., BLEScope [CCS'19])
Future Works: Security in IoT (e.g., BLEScope [CCS'19])
Automatically Uncovering Vulnerabilities in Clouds From Mobile Apps

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

Joint work w/ Omar Alrawi, Ruian Duan, Ranjita Kasturi, Brendan Saltaformaggio, Yinqian Zhang, Qingchuan Zhao, Chaoshun Zuo

Nov 7th, 2019
References I


Jeff Arnold and M Frans Kaashoek, Ksplice: Automatic rebootless kernel updates.


References II


Ruian Duan, Ashish Bijlani, Yang Ji, Omar Alrawi, Yiyuan Xiong, Moses Ike, Brendan Saltaformaggio, and Wenke Lee, *Automating patching of vulnerable open-source software versions in application binaries*.

Zakir Durumeric, James Kasten, Michael Bailey, and J Alex Halderman, *Analysis of the https certificate ecosystem*.

Zakir Durumeric, Frank Li, James Kasten, Johanna Amann, Jethro Beekman, Mathias Payer, Nicolas Weaver, David Adrian, Vern Paxson, Michael Bailey, et al., *The matter of heartbleed*.


References III


Diego Perez-Botero, Jakub Szefer, and Ruby B. Lee, *Characterizing hypervisor vulnerabilities in cloud computing servers*. 


