SMARTGen: Exposing Server URLs of Mobile Apps with Selective Symbolic Execution

Chaoshun Zuo        Zhiqiang Lin

Department of Computer Science
University of Texas at Dallas

April 6th, 2017
Server URLs

https://www.google.com/search?q=www+2017
Server URLs

https://www.google.com/search?q=www+2017

A URL includes

1. Domain name
2. Resource path
3. Query parameters
4. ...

Security Applications

1. Hidden service identification
2. Malicious website detection
3. Server vulnerability fuzzing
4. ...

Server URLs

https://www.google.com/search?q=www+2017

A URL includes

1. Domain name
2. Resource path
3. Query parameters
4. ...

Security Applications

1. Hidden service identification
2. Malicious website detection
3. Server vulnerability fuzzing
4. ...
Server URLs

https://www.google.com/search?q=www+2017

A URL includes

1. Domain name
2. Resource path
3. Query parameters
4. ...

Security Applications

1. Hidden service identification
2. Malicious website detection
3. Server vulnerability fuzzing
4. ...

1 Domain name
2 Resource path
3 Query parameters
4 ...

Security Applications

1 Hidden service identification
2 Malicious website detection
3 Server vulnerability fuzzing
4 ...
Browsers’ URLs vs. Mobile Apps’ URLs
Browsers’ URLs vs. Mobile Apps’ URLs

Source: cloudxtension.com
Hiding the URLs may allow the servers to collect some **private sensitive information**

Mobile apps may talk to some **unwanted services** (e.g., malicious ads sites)

**False illusions** (security through obscurity) to the app developers that their services are secure (server URLs are hidden, none knows and none will attack (or fuzz) them).
Motivation
SMARTGEN Design
Applications
Evaluation
Related Work
Conclusion
References

Security Implications of the URLs in Mobile Apps

1. Hiding the URLs may allow the servers to collect some **private sensitive information**
2. Mobile apps may talk to some **unwanted services** (e.g., malicious ads sites)
3. **False illusions** (security through obscurity) to the app developers that their services are secure (server URLs are hidden, none knows and none will attack (or fuzz) them).

It is imperative to **expose the server URLs from mobile apps**

Source: cloudxtension.com
A Motivating Example: ShopClues

Figure: The password reset activity of ShopClues (between 10 million and 50 million installs).
A Motivating Example: ShopClues

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"}

There was an SQL injection vulnerability at this password reset interface.
Which Analysis We Should Use?
Static Analysis vs. Dynamic Analysis vs. Symbolic Execution

PUT /api/v9/forgotpassword?key=d12121c70dda5edfdgd1df6633f2636c0
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":"d12121c70dda5edfdgd1df6633f2636c0"}
Which Analysis We Should Use?
Static Analysis vs. Dynamic Analysis vs. Symbolic Execution

PUT /api/v9/forgotpassword?key=d12121c70dda5edfdg1df6633f6b36c0
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":"d12121c70dda5edfdg1df6633f6b36c0"}

Static Analysis
- String concatenation
- Crypto keys
Which Analysis We Should Use?
Static Analysis vs. Dynamic Analysis vs. Symbolic Execution

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"}
Which Analysis We Should Use?
Static Analysis vs. Dynamic Analysis vs. Symbolic Execution

PUT /api/v9/forgotpassword?key=d12121c70dda5edfd1df6633f6b36c0
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":"d12121c70dda5edfd1df6633f6b36c0"}

Static Analysis
- String concatenation
- Crypto keys

Dynamic Analysis
- Random inputs
- Incompleteness
- ...

Symbolic Execution
- Systematic
- Automated
- ...

Related Work
Conclusion
References
Symbolic Execution
Generating Inputs Based on Program Code
Various Constraints in Mobile Apps

Various Constraints

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 SMARTGEN

- Automated
- Systematic
- Scalable
Introducing SMARTGEN

- Static analysis
- Selective symbolic execution
- Dynamic analysis
Using soot [soo] framework
Building extended call graph (ECG)
EdgeMiner [CFB+15] for callbacks
Selective Symbolic Execution

- Data flow analysis (w/ FlowDroid [ARF+14])
- Extract the path constraints
- Solve them w/ Z3-str [ZZG13]
Selective Symbolic Execution

- Data flow analysis (w/ FlowDroid [ARF+14])
- Extract the path constraints
- Solve them w/ Z3-str [ZZG13]

Why Selective: only on the execution path of network sending APIs (to trigger the request messages)
Runtime Instrumentation

- System code static rewriting
- Repackaging the apps
- System debugging tool adb
Runtime Instrumentation

- System code static rewriting
- Repackaging the apps
- System debugging tool adb

A new approach that leverages API hooking and Java reflection
Runtime Instrumentation
Runtime Instrumentation

Original APP Code

```java
onCreate(...){
    this.et = findViewById();
    this.bt = findViewById();
    v.yl = new y(et);
    bt.setOnClickListener(yl);
    //init other UI
}
```

Activity Running

```
Forgot Password
We will email you the password!
```

Instrumentation Code
(API Hooking)

```java
handleLoadPackage(...){
    hookOnCreate();
    hookOnResume();
}
```

```
...afterhookedmethod(ctx){
    insert_analysis_code(ctx);
    exec_analysis_thread(ctx);
}
```

Analysis Code

```java
exec_analysis_thread(ctx){
    // Analyze UI
    ui = URAnalysis(ctx);
    // Get Button
    bt = ui.Button;
    // Get OnClick Handler
    y = reflect(bt, "onClick");
    // Get Field "b" of y
    b = reflect(y, "b");
    // Set Text
    b.setText("EMAILSTRING");
    // Perform Click
    trigger(y);
}
```
Runtime Instrumentation

Original APP Code
```java
onCreate(...) {
    this.et = findViewById();
    this.bt = findButton();
    y1l = new y(et);
    bt.setOnClickListener(y1l);
    //init other UI
}
```

Instrumentation Code (API Hooking)
```java
handleLoadPackage(...) {
    hookOnCreat();
    hookOnResume();
    ...
}
```
```
afterHookedMethod(ctx) {
    insert_analysis_code(ctx);
    exec_analysis_thread(ctx);
}
```

Analysis Code
```java
exec_analysis_thread(ctx) {
    // Analyze UI
    ui = UiAnalysis(ctx);
    Get Button
    bt = ui.getButton();
    Get OnClick Handler
    y = reflect(bt, "onclick");
    Get Field "b" of Y
    y = reflect(y, "b");
    b = reflect(y, "b");
    Set Text
    b.setText(EMAILSTRING);
    // Perform Click
    trigger(y);
    ...
```
Runtime Instrumentation

Original APP Code

```
onCreate(...) {
    this.et = findViewById();
    this.bt = findViewById();
    y y1 = new y(et);
    bt.setOnClickListener(y1);
    //init other UI
}
```

Instrumentation Code (API Hooking)

```
handleLoadPackage(...) {
    hookOnCreate();
    hookOnResume();
}
```

Activity Running

```
this.et
this.bt
```

Analysis Code

```
exec_analysis_thread(ctx) {
    // Analyze UI
    u = UAnalysis(ctx);
    Get Button
    b = u.getButton();
    Get OnClick Handler
    y = reflect(b, "onClick");
    y.Get Field "b" of Y
    b = reflect(Y, "b");
    Set Text
    b.setText(EMAILSTRING);
    // Perform Click
    trigger(y);
}
```
Security Applications

- SQL Injection
- Cross Site Scripting
- Others (e.g., malicious URL detection)
SQL Injection

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

Malware sites
Compromised sites
VirusTotal provides services for these detections
## Overall Experimental Results

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td># Apps</td>
<td>5,000</td>
</tr>
<tr>
<td>Size of the Dataset (G-bytes)</td>
<td>126.2</td>
</tr>
<tr>
<td>Time of the first two phases analyses (s)</td>
<td>90,143 (25 hours)</td>
</tr>
<tr>
<td># Targeted API Calls</td>
<td>147,327</td>
</tr>
<tr>
<td># Constraints</td>
<td>47,602</td>
</tr>
<tr>
<td># UI Configuration files generated</td>
<td>25,030</td>
</tr>
<tr>
<td>Time of Dynamic Analysis (s)</td>
<td>486,446 (135 hours)</td>
</tr>
<tr>
<td># Request Messages</td>
<td>257,755</td>
</tr>
<tr>
<td># Exposed URLs</td>
<td>297,780</td>
</tr>
<tr>
<td># Unique Domains</td>
<td>18,193</td>
</tr>
<tr>
<td>Logged Message Size (G-bytes)</td>
<td>24.0</td>
</tr>
<tr>
<td><strong>Σ Malicious URLs</strong></td>
<td><strong>8,634</strong></td>
</tr>
</tbody>
</table>
## Overall Experimental Results

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td># Apps</td>
<td>5,000</td>
</tr>
<tr>
<td>Size of the Dataset (G-bytes)</td>
<td>126.2</td>
</tr>
<tr>
<td>Time of the first two phases analyses (s)</td>
<td>90,143 (25 hours)</td>
</tr>
<tr>
<td># Targeted API Calls</td>
<td>147,327</td>
</tr>
<tr>
<td># Constraints</td>
<td>47,602</td>
</tr>
<tr>
<td># UI Configuration files generated</td>
<td>25,030</td>
</tr>
<tr>
<td>Time of Dynamic Analysis (s)</td>
<td>486,446 (135 hours)</td>
</tr>
<tr>
<td># Request Messages</td>
<td>257,755</td>
</tr>
<tr>
<td># Exposed URLs</td>
<td>297,780</td>
</tr>
<tr>
<td># Unique Domains</td>
<td>18,193</td>
</tr>
<tr>
<td>Logged Message Size (G-bytes)</td>
<td>24.0</td>
</tr>
<tr>
<td>Σ Malicious URLs</td>
<td>8,634</td>
</tr>
</tbody>
</table>
## Statistics on the Extracted String Constraints

<table>
<thead>
<tr>
<th>Constraints Name</th>
<th># Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not null</td>
<td>25,855</td>
</tr>
<tr>
<td>String_length</td>
<td>13,858</td>
</tr>
<tr>
<td>String_isEmpty</td>
<td>377</td>
</tr>
<tr>
<td>String_contains</td>
<td>196</td>
</tr>
<tr>
<td>String_contentEquals</td>
<td>43</td>
</tr>
<tr>
<td>String_equals</td>
<td>3,087</td>
</tr>
<tr>
<td>String_equalsIgnoreCase</td>
<td>991</td>
</tr>
<tr>
<td>String_matches</td>
<td>448</td>
</tr>
<tr>
<td>String_endsWith</td>
<td>11</td>
</tr>
<tr>
<td>String_startsWith</td>
<td>64</td>
</tr>
<tr>
<td>TextUtils_isEmpty</td>
<td>2,355</td>
</tr>
<tr>
<td>Matcher_matches</td>
<td>317</td>
</tr>
</tbody>
</table>
Table 1: Summary of the Performance of SMARTGEN

<table>
<thead>
<tr>
<th>Item</th>
<th># Apps</th>
<th># Targeted API Calls</th>
<th># UI Configuration Files</th>
<th># Exposed URLs</th>
<th>Size of the Dataset (G-bytes)</th>
<th># Unique Domains</th>
<th># Constraints</th>
<th># Logged Message Size (G-bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative Performance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Execution Time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Request Messages</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exposed URLs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unique Domains</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logged Message Size</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Comparison w/ Monkey [mon]

Figure 6: Comparison between SMARTGEN and Monkey

- SMARTGEN Design
- Applications
- Evaluation
- Related Work
- Conclusion
- References
## Security Application: Malicious URL detection

### Table 3: Statistics of Harmful URLs Detected by Each Engine

<table>
<thead>
<tr>
<th>Detection Engine</th>
<th>#Phishing Sites</th>
<th>#Malware</th>
<th>#Malicious Sites</th>
<th>Σ #Harmful URLs</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADMINUSLabs</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>AegisLab WebGuard</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>AutoShun</td>
<td>0</td>
<td>0</td>
<td>863</td>
<td>863</td>
</tr>
<tr>
<td>Avira</td>
<td>2062</td>
<td>941</td>
<td>0</td>
<td>3003</td>
</tr>
<tr>
<td>BitDefender</td>
<td>0</td>
<td>191</td>
<td>0</td>
<td>191</td>
</tr>
<tr>
<td>Blueliv</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>CLEAN MX</td>
<td>0</td>
<td>0</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>CRDF</td>
<td>0</td>
<td>0</td>
<td>150</td>
<td>150</td>
</tr>
<tr>
<td>CloudStat</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Dr.Web</td>
<td>0</td>
<td>0</td>
<td>2330</td>
<td>2330</td>
</tr>
<tr>
<td>ESET</td>
<td>0</td>
<td>75</td>
<td>0</td>
<td>75</td>
</tr>
<tr>
<td>Emsisoft</td>
<td>1</td>
<td>43</td>
<td>0</td>
<td>44</td>
</tr>
<tr>
<td>Fortinet</td>
<td>8</td>
<td>469</td>
<td>0</td>
<td>477</td>
</tr>
<tr>
<td>Google Safebrowsing</td>
<td>0</td>
<td>13</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>Kaspersky</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Malwarebytes hpHosts</td>
<td>0</td>
<td>1103</td>
<td>0</td>
<td>1103</td>
</tr>
<tr>
<td>ParetoLogic</td>
<td>0</td>
<td>800</td>
<td>0</td>
<td>800</td>
</tr>
<tr>
<td>Quick Heal</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Quttera</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>SCUMWARE.org</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>Sophos</td>
<td>0</td>
<td>0</td>
<td>56</td>
<td>56</td>
</tr>
<tr>
<td>Sucuri SiteCheck</td>
<td>0</td>
<td>0</td>
<td>248</td>
<td>248</td>
</tr>
<tr>
<td>ThreatHive</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Trustwave</td>
<td>0</td>
<td>0</td>
<td>80</td>
<td>80</td>
</tr>
<tr>
<td>Websense ThreatSeeker</td>
<td>0</td>
<td>0</td>
<td>56</td>
<td>56</td>
</tr>
<tr>
<td>Yandex Safebrowsing</td>
<td>0</td>
<td>173</td>
<td>0</td>
<td>173</td>
</tr>
</tbody>
</table>

Σ #Harmful URLs: 2071 3818 3826 9715

Σ #Unique Harmful URLs: 2071 3722 3228 8634
Related Work


2 Symbolic Execution. Symbolic execution in app testing in general [MMP+12], path exploration [ANHY12], and malware analysis [WL16]. Closely related work IntelliDroid but it only focuses on malware and lacks generality of UI rich mobile app analysis.
**Mobile App Vulnerability Discovery.** A large body of efforts have focused on discovering vulnerabilities in mobile apps. TaintDroid [EGC+10], PiOS [EKKV11], CHEX [LLW+12], SMV-Hunter [SSG+14].

**Remote Server Vulnerability Discovery.** Few efforts (e.g., AUTOForge [ZWWL16]) including smartgen [ZL17] have been focusing on identifying the vulnerabilities in app’s server side.
SMARTGen [ZL17]  
A Fully Automated, Symbolic Execution Based, Mobile App Execution Framework

**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**  
- Each app has 1,000,000 installs  
- These apps actually talk to 2,071 phishing sites, 3,722 malware sites, and 3,228 malicious sites
Thank You

Acknowledgement
- AFOSR, NSF
- VirusTotal (premium services)

first name.last name@utdallas.edu
References


References III


