An Empirical Study of Web Resource Manipulation in Real-world Mobile Applications

Xiaohan Zhang, Yuan Zhang, Qianqian Mo, Hao Xia, Zhemin Yang, Min Yang
XiaoFeng Wang, Long Lu, and Haixin Duan
Background

• Mobile apps are integrating more and more Web services
  • advertising
  • social sharing
  • even authentication and authorization

• Most of Web-App integrations are through light-weight **in-app** Web browsers, called WebViews
  • Android: WebView
  • iOS: UIWebView/WKWebView
Web-App Integration Security Risks

• Security risks to both sides
  • Web-to-App attacks
  • App-to-Web attacks
Web-App Integration Security Risks

• Security risks on both sides
  • Web-to-App attack
  • App-to-Web attack

• Web-to-App attacks
  • where unauthorized Web code access sensitive functions of the host apps

• Existing works
  • attacks [Luo et al. ACSAC’11], [Sooel et.al, NDSS’ 16], [OSV-Hunter, S&P ’18]
  • detections [BridgeScope, RAID’17]
  • defenses [NoFrak, NDSS’14], [Draco, CCS’16]
Web-App Integration Security Risks

• Security risks on both sides
  • Web-to-App attacks
  • App-to-Web attacks

• App-to-Web attacks
  • where the host apps manipulate sensitive resources of the Web

• Existing works
  • partially mentioned in theory [luo et al. ACSAC’11], [Eric et al. CCS’14]
  • no real-world cases

Open Questions:
1. How Web resources are manipulated by real-world apps?
2. Are there any real-world App-to-Web attacks?
### Web Resource Manipulation APIs

- Both Android and iOS provide a handful of APIs for host apps to manipulate the Web resources

<table>
<thead>
<tr>
<th>Manipulated Web Resources</th>
<th>Android WebView</th>
<th>iOS UIWebView</th>
<th>iOS WKWebView</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local Storage</td>
<td>CookieManager</td>
<td>NSHTTPCookieStorage</td>
<td>WKWebsiteDataStorage</td>
</tr>
<tr>
<td>Web Content</td>
<td>loadUrl,</td>
<td>stringByEvaluatingJavascriptFromString</td>
<td></td>
</tr>
<tr>
<td></td>
<td>evaulateJavascript</td>
<td></td>
<td>evaluateJavascript</td>
</tr>
<tr>
<td>Web Address</td>
<td>onPageFinished,</td>
<td>\</td>
<td>\</td>
</tr>
<tr>
<td></td>
<td>shouldOverrideUrlLoading</td>
<td>\</td>
<td>\</td>
</tr>
<tr>
<td>Network Traffic</td>
<td>shouldInterceptRequest</td>
<td>shouldStartLoadWithRequest</td>
<td>decidePolicyForNavigationAction, decidePolicyForNavigationResponse</td>
</tr>
</tbody>
</table>

Examples:
1. obtain cookies using `CookieManager.getCookie`
2. intercept network traffic to get user credentials using `shouldInterceptRequest`

**Is it secure?**
Motivating Example

The Website *facebook.com* is loaded into WebViews of two apps

- both apps use `CookieManager.getCookie` to get cookies of *facebook.com*

App A: Facebook’s official app
App B: Chatous, a third-party app

Observation: it is risky when security principals are crossed!
Definitions

• Two security principals involved
  • **Web Principal**, the manipulated Web resources, \( P_w \)
  • **App Principal**, the manipulating code, \( P_A \)

• Cross Principal Manipulation (XPM)

\[
P_w \neq P_A
\]

Target: to measure XPMs in real-world apps
Methodology

• Finding XPMs in real-world apps

1. locate all manipulations   2. identify $P_A$ and $P_W$   3. determine $P_A = P_W$?

```java
package com.chatous.chatous.managers;
...
if (CookieManager.getInstance().getCookie("https://facebook.com") != null) {
    // get Facebook cookies
    cookies = CookieManager.getInstance().getCookie("https://facebook.com");
    // store these cookies
    BasicCookieStore cookieStore = new BasicCookieStore();
    cookieStore.addCookie(cookies);
    ...

    // abuse these cookies to collect user privacy information.
    ...
```

$P_A \neq P_W \rightarrow \text{XPM}$

Non-trivial
Challenge 1: multiple security principals exist in the app

- the host app itself
- several third-party libraries

**Solution:** identify third-party libraries
- $P_A$ of third-party library: library name
- $P_A$ of the host app: host app’s meta-info

- library identification algorithm
  - Merkle-tree based code signature
  - please refer to our paper for more details

$P_W$: facebook.com

```javascript
// get Facebook cookies
CookieManager.getInstance().getCookie("facebook.com");
```

not XPM

FB_SDK

XPM

chatous
Determine $P_w = P_A$?

**Challenge 2:** semantic gaps between $P_w$ and $P_A$

- “chatous” and “facebook”
- “google” and “youtube”
- abbreviation: “fb” and “facebook”

```java
// get Facebook cookies
CookieManager.getInstance().getCookie(
    "facebook.com");
```
Determine $P_w = P_A$ ?

**Challenge 2:** semantic gaps between $P_w$ and $P_A$

- “chatous” and “facebook”
- “google” and “youtube”
- abbreviation: “fb” and “facebook”

**Solution:** ask search engine

- e.g. “facebook” and “fb” have more than 80% similarity in google search result

**searching-based classifier**

- normalize search results into $W$ and $A$ using bag-of-words model
- similarity distances between these two vectors

\[
isXPM(P_w, P_a) := \text{sim\_distance}(P_w, P_a) \geq \theta
\]
XPMChecker Design & Implementation

Implementation is based on *Soot* and *FlowDroid*
- with customized ICFG
- API-specific data flow analysis

(please refer to our paper for more technical details)
XPMChecker Evaluation

• Dataset
  • **84,712** apps from Google Play during Jul 2017, with at least 5,000 installations across 48 categories.

• Performance
  • **95.3%** of all apps (80,694/84,712) are successfully analyzed
  • 233 hours with 9 processes, **10 seconds/app**.
    • CentOS 7.4 64-bit server, 64 CPU cores (2GHz), 188 GB memory
    • 9 processes, 20 minutes timeout

• Effectiveness
  • with 200 manually labeled ground truth
  • **98.9%** precision and **97.9%** recall ($\theta = 0.3134$)
Finding: XPM Prevalence

• **XPMs are very popular in real-world apps**
  • **4.8%** (3,858/80,694) of all apps contain XPMs

<table>
<thead>
<tr>
<th></th>
<th># of Apps (% in all apps)</th>
<th># of manipulations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apps that manipulate Web</td>
<td>13,599 (16.9%)</td>
<td>29,448</td>
</tr>
<tr>
<td>resources</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Apps with XPMs</td>
<td>3,858 (4.8%)</td>
<td><strong>14,476 (49.2%)</strong></td>
</tr>
</tbody>
</table>

• **49.2%** (14,776/29,448) of all Web resource manipulations are cross-principal.
Finding: XPM Location

• A large part of XPMs are from libraries.
  • 63.6% of XPMs originate from 88 libraries in our dataset

• Reflections on current defensive work
  • works that consider the app as a single principal is not fine-grained enough nor accurate
    [WIREFRAME, AsiaCCS’17]
Finding: XPM Intents

• More than 90% XPMs provide normal utilities
  • Inject JS to customize Web services to improve user experience
    • add navigation controls
    • customize Google Cloud Print
  • Monitor Web addresses to invoke local apps
Malicious XPM Intents

• Confirm malicious XPMs in real-world for the first time
  • find 22 malicious XPMs in 21 apps, with up to 130M installations
  • report to Google and the malicious intents are removed
  • 4 iOS apps with such malicious XPMs are also confirmed

• Three categories:

<table>
<thead>
<tr>
<th>Malicious behavior</th>
<th># of apps</th>
</tr>
</thead>
<tbody>
<tr>
<td>impersonating legitimate relying party in OAuth</td>
<td>2</td>
</tr>
<tr>
<td>stealing user credentials</td>
<td>6</td>
</tr>
<tr>
<td>stealing and abusing cookies</td>
<td>14</td>
</tr>
</tbody>
</table>
Case Study 1. Impersonating relying party in OAuth

- App instaview impersonates “Tinder” in Instagram OAuth
  - a profile tracker for users to see their Instagram visiting statistics
  - 1,000,000-5,000,000 installations

```java
package com.instaview.app;
...
public class LoginActivity extends Activity{
...
    // get Tinder’s client ID
    String clientId = getTinderClientId();
    ...
    this.webview.setWebViewClient(new WebViewClient()
    {
        public boolean shouldOverrideUrlLoading(WebView arg1, String url) {
            ...
            // check if url is Instagram’s OAuth API and extract the access token for Tinder
            if (url.startsWith("api.instagram.com/oauth") && contains("code=")) {
                String accessToken = url.substring(url.indexOf("code=") + 5, url.length());
                // then use this token to access user’s profile info
            }
        }
    });
...`
```
Case Study 2. Stealing user credentials

adkingkong steals user’s Google account credentials

• an advertising app with 500,000 – 1,000,000 installations

```java
package co.kr.adkingkong.libs.autoinstall;
...
public class GoogleWebLogin extends RelativeLayout {
    ...
    // load Google login Web page
    this.webview.loadUrl("accounts.google.com");
    ...
    this.webview.setWebViewClient(new WebViewClient() {
        public void onPageFinished(WebView arg1, String url) {
            ...
            // inject JS to steal users' email and password
            arg1.loadUrl("javascript:
                if (document.getElementById('gaia_loginform') !== null) {
                    document.getElementById('gaia_loginform').onsubmit = function onSubmit(form) {
                        // extract email and password from the login form
                        email = document.getElementById('email-display').innerHTML;
                        passwd = document.getElementById('Passwd').value;
                        ...
                    };
                }
            ...";
        }
    });
    ...
```
App **chatous** steals Facebook cookies and abuses them to collect sensitive user info and send spams

- a random chatting app with 10,000,000 to 50,000,000 installations

```java
package com.chatous.chatous.managers;
...
public class FacebookManager extends Manager {
    ...
    if (CookieManager.getInstance().getCookie("https://facebook.com") != null) {
        // get Facebook cookies
        cookies = CookieManager.getInstance().getCookie("https://facebook.com");
        // use these cookies to access user’s Facebook homepage
        DefaultHttpClient httpclient = new DefaultHttpClient();
        httpclient.setCookieStore(cookieStore);
        HttpResponse response = httpclient.execute(new HttpGet("https://facebook.com/first_degree.php?" + ...));
        ...
        // get user’s friend list and send spam invitations
        List<String> friends = parse_response(response);
        for (friend: friends) {
            send_invitations(friend);
        }
    }
```
Finding: XPM Targets and Their Awareness

• More than 70% of XPMs target top popular Web services  
  • such as Google, Facebook, YouTube, Twitter, etc.

• However, most of them are **unaware** of such risks  
  • all the above providers except Goolge allow sensitive Web services to be loaded into WebView of any apps.  
  • Google **are unable** to effectively prevent users from using WebView to do OAuth.  
  • [Google announcement](#), Aug 2016
Conclusion

• Measurement tool: automatically find Cross Principal Manipulation (XPM)

• First large scale empirical study on XPM in real-world
  • better understanding of the threat and development of countermeasures
  • confirm malicious App-to-Web attacks on both Android and iOS that already affect a large number of devices

• Dataset released: https://xhzhang.github.io/XPMChecker/
Thanks!

Xiaohan Zhang
xh_zhang@fudan.edu.cn
Backup slides
Future work

1. What are other channels besides WebViews?
   - Hybrid frameworks, such as Cordova
   - Customized browser, such as Tencent X5
   - Other methods besides manipulation APIs

2. Can we directly detect malicious XPM behaviors?
   - Currently we rely on manual effort to confirm malicious XPM behaviors
   - Heuristic rules based on current findings

3. Defensive works
API Models

• Three types of APIs based on the source of manipulated Web URL
  • Type I: URL from a parameter
  • Type II: URL from base WebView instance
  • Type III: URL from a callback parameter (runtime URL)

Listing 1: Type I, URL from a parameter.
```java
CookieManager cm = new CookieManager();
cm.getCookie("www.google.com");
```

Listing 2: Type II, URL from base WebView instance.
```java
WebView wv = new WebView(this);
// some code
wv.loadUrl("www.google.com");
// some other code
wv.evaluateJavaScript("JS_CODE", ..);
```

Listing 3: Type III, URL from a callback parameter.
```java
boolean shouldOverrideUrlLoading(WebView webView, String url)
{
  if(url.startsWith("www.google.com"))
  {
    // some code
  }
  else if(url.equals("www.facebook.com");
  {
    // some other code
  }
  // other code
}
```
Identifying third-party libraries

- **idea:** library code must exist in more than one apps with different developers
  - extra benefits: name recovery even some apps obfuscate their code

- Merkle-tree based code signature

[libPecker SANER’18]
Similarity Distance

\[ \text{sim\_distance}(P_w, P_a) = \cos(P_w, P_a) = \frac{\sum_{i=1}^{n} A_i W_i}{\sqrt{\sum_{i=1}^{n} A_i^2 \sum_{i=1}^{n} W_i^2}} \]

where \( P_w \) and \( P_A \) is the Web principal and Code principal, \( W \) and \( A \) is the searching vectors, with top \( n \) terms and their frequencies.
Determine the threshold $\theta$

- Manually labeled 1200 $<P_w, P_A>$ pairs
  - 1000 are used to determine threshold $\theta$
  - 200 are used to test the performance

- ROC & EER point
  - $\theta = 0.3134$, AUC = 97.8%

- 27% improvement in precision comparing to simple word similarity

Figure 4: ROC curve for varied $\theta$ in XPMClassifier with 1000 manipulation points.
Case 3 More details

- User complaints on malicious Chatous app