The Bridge between Web Applications and Mobile Platforms is Still Broken

Philipp Beer, Lorenzo Veronese, Marco Squarcina, Martina Lindorfer
TU Wien
Contributions

Malicious application

Android Custom Tab

Attack similar to XS state inference and CSRF

UI flaw

Achieves Stealthiness

Malicious website

Android WebView Attack

Accesses to user’s microphone/camera
### Related Work

**Attacks on WebView in the Android System**

Tongbo Lu, Hao Hao, Wenhong Du, Yile Wang, and Hong Yin
Dept. of Electrical Engineering & Computer Science, Syracuse University, Syracuse, New York, USA

**ABSTRACT**

WebView is a crucial component in both Android platforms, enabling synchronization and taking up much less processing overhead than loading full web pages. However, WebView provides a number of APIs allowing WebView to be attacked and used for a variety of malicious purposes, such as payload download and distribution in mobile apps. Using these features, apps can embed WebView to their interfaces to be exploited in the real world. In this work, we analyze WebView and current vulnerabilities. We find that 784 applications (8.9%) from 129 different categories are vulnerable. The results show that the attacks can be performed not only in the WebView and WebViewLoader classes but also in the relevant third-party apps. We exploit 16 vulnerabilities and evaluate the risk of these vulnerabilities.

**A View To A Kill: WebView Exploitation**

Matthew Naumann and Shanthi Neelesh
Secure Systems Lab, Nebraska Ivanovo State University
Email: remm@nevasecurity.org

Abstract—WebView is a technique to embed web and native applications into mobile devices. The fact that access to native apps requires explicit data sharing, and that WebView is devoid of its own security context, makes WebView vulnerable to attacks.

In this paper, we present a detailed analysis of WebView and several practical exploits that can be performed against it. We utilize this analysis to develop a WebView-based mobile application that we use to break into a vulnerable system.

**A Large-Scale Study of Mobile Web App Security**

Patrick McDaniel, Arian Doukaz, John Mitchell, Chris Kruegel, and Giovanni Vigna

Brandeis University
Email: {pm26, amo4, cmitch, ckruegel}@brandeis.edu

Exante University
Email: dasgupta@exante.edu

University of California, Santa Barbara
Email: cix, vigna@cs.ucsb.edu

**Abstract**

Recent advances in mobile Web app technology have led to the growth of mobile web applications. Many mobile apps use web views, which provide a means to navigate to a remote website or access sensitive data from the web. These apps are often embedded within the main app or a web view container. In this work, we study three vulnerabilities in mobile Web apps, leading to sensitive information exposure, and analyze the risk of each vulnerability.

1. Introduction

Mobile operating systems allow third-party developers to create applications ("apps") that run on a mobile device, and apps can access mobile applications ("apps") that run on a mobile device, and apps can access sensitive data from the TimeSpan, which can be used to gain unauthorized access to the mobile device.

2. Threat Model

A fundamental requirement for exploiting a WebView app is to gain control over the web content that is requested by the app. This requires an attack on the WebView and the underlying HTML. The attacker needs to exploit the WebView security flaw that is subsequently executed by the app.

3. Server compromise

If the attacker manages to manipulate the contents sent over the server, the attack leverage is very

**Bifocal:** Analyzing WebView Vulnerabilities in Android Applications

Erika Chin and David Wagner
University of California, Berkeley
Email: erika, dawagner@berkeley.edu

**Abstract**

WebView allows Android developers to embed a web page within an application, essentially merging native applications with web browsers. WebView is widely used in mobile applications and games on Android devices. However, many mobile platforms, especially smartphones, are vulnerable to WebView-based attacks.

In this paper, we explore two WebView vulnerabilities across android applications. These vulnerabilities can be exploited to execute arbitrary JavaScript code or to access sensitive data from the application.

We build a tool, Bifocal, to detect these vulnerabilities and demonstrate the presence of vulnerable code. We find significant security issues with WebView-based applications, and exploit vulnerabilities in real-world applications. We also develop a system to protect against WebView-based attacks.

**Keywords:** Security, webviews, android applications, static analysis.
Integrating Web Content in Mobile Apps

• Serve as in-app browsers
  - Android
    - WebView
    - Custom Tab
    - Trusted Web Activities
  - iOS
    - WKWebView
    - SFSafariViewController
Custom Tab

- Report navigation **callbacks** to host application
- Custom Tabs **share state** with browser
- Useful for e.g. SSO
Custom Tab Attack

- Features enable attack similar to XS-leak to infer user information
- Malicious app uses event sequence to infer user data
- Three approaches
  - Status code-based approach
  - Redirection-based approach
  - Timing-based approach
Status code-based approach

- Additional failed event triggered on 4xx/5xx response code and empty response body
Redirection-based approach

- Finished/failed event triggered for every JS/meta redirection
Timing-based approach

- Measure time between NAVIGATION_STARTED and NAVIGATION_FINISHED
Stealthiness

Normal Custom Tab launch

Hiding the Custom Tab
Mitigation

Custom Tab Provider:
Prevent callbacks on redirection
(prevents redirection-based attack)

Android OS:
Restrict callbacks to Custom Tabs in the foreground
(prevents stealthy attack)
Security Implications

- Opening website in Custom Tab is **top-level navigation**
- Cross-origin attack-targeted mitigations useless
- Allows to bypass
  - SameSite cookies
  - Framing Protection
  - Cross-Origin-Opener-Policy
  - Fetch Metadata

<table>
<thead>
<tr>
<th>Headers</th>
<th>Chromecast</th>
</tr>
</thead>
<tbody>
<tr>
<td>connection</td>
<td>close</td>
</tr>
<tr>
<td>accept-language</td>
<td>en-US, en;q=0.9, de-AT;q=0.8, de;q=0.7, en-AT;q=0.7, en-DE;q=0.6, en-US;q=0.5, fr;q=0.4, it;q=0.3, de;q=0.2, de;q=0.1, xx;q=0.05, *;q=0.01</td>
</tr>
<tr>
<td>accept-encoding</td>
<td>gzip, deflate, br</td>
</tr>
<tr>
<td>sec-fetch-dest</td>
<td>document</td>
</tr>
<tr>
<td>sec-fetch-mode</td>
<td>navigate</td>
</tr>
<tr>
<td>sec-fetch-site</td>
<td>none</td>
</tr>
<tr>
<td>accept</td>
<td>text/html, application/xhtml+xml, application/*, <em>/</em></td>
</tr>
<tr>
<td>user-agent</td>
<td>Mozilla/5.0 (Linux; Android 11; AC2003) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.177 Mobile Safari/537.36</td>
</tr>
<tr>
<td>upgrade-insecure-requests</td>
<td>1</td>
</tr>
<tr>
<td>sec-ch-ua</td>
<td>&quot;Android&quot;</td>
</tr>
<tr>
<td>sec-ch-ua-mobile</td>
<td>&quot;?1&quot;</td>
</tr>
<tr>
<td>sec-ch-vendor</td>
<td>Not A;Brand;v=&quot;99&quot;, &quot;Chromium:v=&quot;101&quot;, ...</td>
</tr>
<tr>
<td>host</td>
<td>webhook.site</td>
</tr>
<tr>
<td>content-length</td>
<td>content-length</td>
</tr>
<tr>
<td>content-type</td>
<td>content-type</td>
</tr>
</tbody>
</table>
Custom Tab CSRF

- 10.3% of state-changing requests still implemented using GET
- ... sensitive state-changing POST requests can be exploited by changing to GET requests (e.g. IMDB, PayPal and Meetup)
- No detectable attack
- Allows to bypass even SameSite strict cookies on Chrome!
Web View Attack 1/2

- Vulnerability in two popular WebView plugins for Android frameworks
  - React Native WebView
  - unity-webview
- Websites in WebView can access camera/microphone, if
  - Application has permission
  - Application grants WebView permission
- Default: WebView permission denied
Web View Attack 2/2

- Two plugins *by default* grant permission to WebView
- Attacker *loads malicious website* into WebView of vulnerable app
- Access to camera & microphone
- Mitigation
  - Deny access *by default*
  - Implement *access control mechanism* by plugin developers
  - Show indicator when camera/microphone is used
Conclusion

- Custom Tab Attack
  - Abuse Custom Tab for XS-like attacks (state inference & CSRF)
  - Doesn’t trigger user-observable events
  - Defeats existing mitigations for XS attacks

- Web View Attack
  - Implementation flaw in Android framework plugins allows microphone/camera access to web attacker
Thank you!
Questions?

@beerphilipp
Backup: Preliminary Evaluation

- Analysed top 250 downloaded free applications on Google Play (247 successfully)
- 85 (34%) use Custom Tabs
- 57 (23%) use Custom Tabs Callback
- Web View attack app vulnerability:

<table>
<thead>
<tr>
<th>Permissions</th>
<th>RN WebView</th>
<th>unity-webiew</th>
<th>Others</th>
</tr>
</thead>
<tbody>
<tr>
<td>✛ ✛ ▼ ▼ ▼</td>
<td>0</td>
<td>1 (&lt; 1%)</td>
<td>113 (46%)</td>
</tr>
<tr>
<td>✛ ▼ ▼ ▼ ▼</td>
<td>0</td>
<td>0</td>
<td>28 (11%)</td>
</tr>
<tr>
<td>▼ ▼ ▼ ▼ ▼</td>
<td>2 (&lt; 1%)</td>
<td>0</td>
<td>32 (13%)</td>
</tr>
<tr>
<td>▼ ▼ ▼ ▼ ▼</td>
<td>5 (2%)</td>
<td>0</td>
<td>66 (27%)</td>
</tr>
<tr>
<td>▼ ▼ ▼ ▼ ▼</td>
<td>7 (3%)</td>
<td>0</td>
<td>126 (51%)</td>
</tr>
</tbody>
</table>
Backup: Custom Tab Attack Code

```kotlin
val callback = object : CustomTabsCallback() {
    override fun onNavigationEvent(navigationEvent: Int, extras: Bundle?) {
        when (navigationEvent) {
            TAB_SHOWN -> {
                startActivity(Intent(this, OverlayActivity::class.java))
            }
            NAVIGATION_STARTED -> {
                onNavigationStarted()
            }
            NAVIGATION_FINISHED -> {
                onLoadingFinished()
            }
            NAVIGATION_FAILED -> {
                onLoadingFailed()
            }
            else -> {
            }
        }
    }
}

val connection = object : CustomTabsServiceConnection() {
    override fun onCustomTabsServiceConnected(name: ComponentName, client: CustomTabsClient) {
        session = client.newSession(callback)
        client.warmup(0)
    }
    override fun onServiceDisconnected(componentName: ComponentName?) {
    }
}

CustomTabsClient.bindCustomTabsService(context, packageName, connection)
cctIntent.launchUrl(context, Uri.parse(url))
```