AN EVALUATION OF THE GOOGLE CHROME EXTENSION SECURITY ARCHITECTURE

Nicholas Carlini, Adrienne Porter Felt, David Wagner
University of California, Berkeley
CHROME EXTENSIONS
WEB ATTACKER

servers

client-side website

client-side website

extension

browser API
- history
- bookmarks
NETWORK ATTACKER
NETWORK ATTACKER
CHROME’S SECURITY MECHANISMS
PRIVILEGE SEPARATION
ISOLATED WORLDS

servers

core extension

content script

client-side website

client-side website

browser API

history

bookmarks
PERMISSIONS
Vulnerabilities
Isolated worlds
Privilege separation
Permissions
New defenses
VULNERABILITIES
FINDING BUGS

SAMPLE
50 most popular + 50 random extensions

METHODS
Black-box testing + source code analysis

VERIFICATION
Built exploits to confirm the vulnerabilities
<table>
<thead>
<tr>
<th>Vulnerability Location</th>
<th>Web Attacker</th>
<th>Network Attacker</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core</td>
<td>5</td>
<td>50</td>
</tr>
<tr>
<td>Content Script</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Website</td>
<td>6</td>
<td>14</td>
</tr>
</tbody>
</table>

70 vulnerabilities in 40 extensions
## Vulnerable Extensions

<table>
<thead>
<tr>
<th></th>
<th>Popular</th>
<th>Random</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>22</td>
<td>18</td>
<td>40</td>
</tr>
</tbody>
</table>
EXAMPLE: SPEED DIAL
ISOLATED WORLDS
Isolated worlds:
protect content scripts
from web attackers
Vulnerability count:

3 content script vulns
DATA AS HTML

MISTAKE
Insert data as HTML, where it can execute

MITIGATION
Will execute in website’s isolated world

VULNERABILITIES
6 extensions have data-as-HTML bugs that don’t cause content script vulnerabilities
EVAL

MISTAKE
Use eval to execute untrusted data

MITIGATION
Isolated worlds does not mitigate this bug

VULNERABILITIES
2 vulnerabilities due to this mistake
CLICK INJECTION

MISTAKE
Trusting event handlers on a website

MITIGATION
Isolated worlds does not mitigate this bug

VULNERABILITIES
1 vulnerability due to this mistake
Isolated worlds is highly effective because it mitigates common bugs
PRIVILEGE SEPARATION
Privilege separation:

protect core extensions
PRIVILEGE SEPARATION
Can regular developers use privilege separation?
<table>
<thead>
<tr>
<th>Permissions</th>
<th>Extensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>All of the extensions’</td>
<td>7%</td>
</tr>
<tr>
<td>Partial: XHRs</td>
<td>15%</td>
</tr>
<tr>
<td>Partial: tab control</td>
<td>8%</td>
</tr>
<tr>
<td>Partial: other</td>
<td>8%</td>
</tr>
</tbody>
</table>

(Of the 61 extensions with content scripts)

PRIVILEGE “LEAKAGE”
Privilege separation would fully protect most core extensions, but a third of developers circumvent it.
Vulnerability count:

50 core extension vulns
METADATA ATTACK
METADATA ATTACK

5 metadata attacks

servers

client-side

website

content script

core

extension

extension

browser API

history

bookmarks

METADATA ATTACK
HTTP SCRIPTS/XHRS

client-side website

content script

core extension

servers

extension

browser API
  history
  bookmarks
HTTP SCRIPTS/XHRS

servers

client-side
website

content script

16 HTTP XHRs
28 HTTP scripts

core
extension

extension

browser API

history bookmarks
Privilege separation can be powerful, but its placement in the system matters.
Something else is needed to protect core extensions
PERMISSIONS
Permissions:
limit the scope of core vulnerabilities
PERMISSION RATE

- None: 15%
- Low: 11%
- Medium: 30%
- High: 44%

27 buggy extensions
Reduces potential for severe attacks by half
No correlation between bugs and permissions
Yes, permissions limit the scope of vulnerabilities
NEW DEFENSES
Use CSP to ban unsafe coding practices
<table>
<thead>
<tr>
<th>Restriction</th>
<th>Security Benefit</th>
<th>Broken, But Fixable</th>
<th>Broken And Unfixable</th>
</tr>
</thead>
<tbody>
<tr>
<td>No HTTP scripts in cores</td>
<td>15%</td>
<td>15%</td>
<td>0%</td>
</tr>
<tr>
<td>No inline scripts</td>
<td>15%</td>
<td>79%</td>
<td>0%</td>
</tr>
<tr>
<td>No eval</td>
<td>3%</td>
<td>30%</td>
<td>2%</td>
</tr>
<tr>
<td>No HTTP XHRs</td>
<td>17%</td>
<td>29%</td>
<td>14%</td>
</tr>
</tbody>
</table>

**POTENTIAL BANS**
<table>
<thead>
<tr>
<th>Restriction</th>
<th>Security Benefit</th>
<th>Broken, But Fixable</th>
<th>Broken And Unfixable</th>
</tr>
</thead>
<tbody>
<tr>
<td>No HTTP scripts in cores</td>
<td>15%</td>
<td>15%</td>
<td>0%</td>
</tr>
<tr>
<td>No inline scripts</td>
<td>15%</td>
<td>79%</td>
<td>0%</td>
</tr>
<tr>
<td>No eval</td>
<td>3%</td>
<td>30%</td>
<td>2%</td>
</tr>
<tr>
<td>No HTTP XHRs</td>
<td>17%</td>
<td>29%</td>
<td>14%</td>
</tr>
<tr>
<td>Restriction</td>
<td>Security Benefit</td>
<td>Broken, But Fixable</td>
<td>Broken And Unfixable</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------</td>
<td>---------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>Chrome 18 policy</td>
<td>27%</td>
<td>85%</td>
<td>2%</td>
</tr>
</tbody>
</table>
CONCLUSION

- Isolated worlds prevents common bugs
- Some developers don’t use privilege separation optimally
- Permissions reduce scope of vulns
- Recommend banning unsafe practices to protect core extensions
QUESTIONS?
adriennefelt@gmail.com

www.adrienneporterfelt.com