Showcase Showdown
Browser Security Edition

Actionable Metrics for Web Browser Security

Shawn Moyer
Practice Manager

Ryan Smith
Chief Scientist
Hi, BlackHat.

- Quick overview of browser security research
  - Released in late 2011
  - Evaluated security of Internet Explorer 9, Chrome 12 & 13, Firefox 5, on Windows 7 (32-bit)
- Collaborative effort by the entire Labs R&D team:
  - Drake, Mehta, Miller, Moyer, Smith, Valasek
- Some key points and a nickel tour.
- Paper, etc: http://www.accuvantlabs.com
We’ve come a long way...

- The browser is the most critical application we use today
  - In some cases it may be the only application we use
  - Especially true as we move to SaaS / cloud / etc

- Most common entry point for viruses, malware, client-side exploitation
No maps for these territories

- Metrics / bakeoffs thus far have been narrowband
  - Focused on some single, easy-to-measure test case
  - Bar charts are not the end goal of security “research”

- We took a more holistic view.
  - Defined shared attack surface on 3 major browsers
    - Specific focus on exploitation/persistence defense
  - Our goal was to create measurable, agnostic criteria
  - Public release of all test data and tool chains to foster an open dialogue
Browser Security Ecosystem

- We defined the browser security ecosystem as:
  - Browser Process Security Architecture
  - Add-On Security (Plugins, Extensions)
  - Exploit Mitigation and Sandboxing
  - Malware Detection / Blacklisting
  - Historical Vulnerability Metrics

- Again, our focus was on commonalities.
Process Security Architecture

- Common across all modern browsers:
  - Multi-process / multi-threaded architecture
  - Security barriers, trust zones, integrity models

- Integrity models in Windows 7:
  - System
  - High
  - Medium
  - Low
IE Process Architecture

- “Loosely Coupled” model
  - UI frame, tabs (low integrity) largely independent

- Medium integrity broker process
  - Creates low integrity tabs:
    - General Browsing and Rendering
    - ActiveX controls and other plugins
    - GPU acceleration
    - Tab-independent: downloads, toolbars, etc
Chrome Process Architecture

- Uses a medium integrity broker process
  - Manages the UI
  - Creates separate low integrity processes for:
    - Rendering tabs
    - Out-of-process hosting for plugins, extensions
    - GPU acceleration
    - Named pipes created by broker for IPC
Firefox Process Architecture

- Single, medium integrity browser process
  - Contains entire browsing session in a single address space
    - All tabs
    - All add-ons
    - GPU acceleration
    - etc.
  - One exception: Flash and Silverlight plugins
    - Hosted out-of-process at medium integrity
Why Architecture Matters

- Process architecture determines if an exploit will:
  - Succeed or fail
  - Attain persistence
  - Have access to other in-browser data
  - Communicate with other processes / plugins

- Along with sandboxing, key criteria for true exploitability
Sandboxing

- Why is sandboxing important?
  - There will always be bugs (until Skynet takes over)
  - Assume attackers will find a method for exploitation
  - Limit what damage can be done

- We’ve accepted compromise, hence emphasis on limitations post-mortem

- Ultimately if a sandbox bypass is required to land a payload, attacker complexity is increased
Sandboxing (cont.)

- General effectiveness of sandboxes

<table>
<thead>
<tr>
<th>Sandbox Result</th>
<th>Chrome</th>
<th>Internet Explorer</th>
<th>Firefox</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Files</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Write Files</td>
<td>✓</td>
<td>❌</td>
<td>❌</td>
</tr>
<tr>
<td>Read Registry Keys</td>
<td>✓</td>
<td>❌</td>
<td>❌</td>
</tr>
<tr>
<td>Write Registry Keys</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Network Access</td>
<td>✓</td>
<td>❌</td>
<td>❌</td>
</tr>
<tr>
<td>Resource Monitoring</td>
<td>✓</td>
<td>❌</td>
<td>❌</td>
</tr>
<tr>
<td>Thread Access</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Process Access</td>
<td>✓</td>
<td>❌</td>
<td>❌</td>
</tr>
<tr>
<td>Process Creation</td>
<td>✓</td>
<td>❌</td>
<td>❌</td>
</tr>
<tr>
<td>Clipboard Access</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>System Parameters</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
</tr>
<tr>
<td>Broadcast Messages</td>
<td>✓</td>
<td>❌</td>
<td>❌</td>
</tr>
<tr>
<td>Desktop &amp; Windows Station Access</td>
<td>✓</td>
<td>❌</td>
<td>❌</td>
</tr>
<tr>
<td>Windows Hooks</td>
<td>❌*</td>
<td>❌</td>
<td>❌</td>
</tr>
<tr>
<td>Named Pipes Access</td>
<td>✓</td>
<td>❌</td>
<td>❌</td>
</tr>
</tbody>
</table>

- Action was blocked
- Action was partially blocked
- Action was allowed

*Isolated Desktop and Window Station
Sandboxing (cont.)

- Google Chrome prevents processes in the sandbox from doing much of anything
  - Even if permission is granted, it is limited to the alternate desktop

- Microsoft Internet Explorer allows read access to most objects on the operating system
  - Deters a handful of system modifications

- Mozilla Firefox, on the other hand, is only limited by standard medium integrity
  - Permitting read, write and system change capabilities associated with regular, non-administrator users
  - If current user can do it, so can FF
JavaScript JIT Hardening

• JIT engines emit native code that can weaken security

• ASLR and DEP already exist for compiled binaries, but are not effective protections for JIT engines because
  • JIT compilation bridges the distinction between data and code
  • Predictable executable memory can turn a previously un-exploitable bug into a trivial exploit

• JIT hardening prevents the abuse of the JIT engine itself
## JIT Hardening Comparison

<table>
<thead>
<tr>
<th>JIT Hardening Techniques</th>
<th>Chrome</th>
<th>Internet Explorer</th>
<th>Firefox</th>
</tr>
</thead>
<tbody>
<tr>
<td>Codebase Alignment Randomization</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Instruction Alignment Randomization</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Constant Folding</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Constant Blinding</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Resource Constraints</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Memory Page Protection</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Additional Randomization</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Guard Pages</td>
<td>✗*</td>
<td>✓</td>
<td>✗</td>
</tr>
</tbody>
</table>

- ✓ Technique was implemented
- ✗ Technique was not implemented
- ✗ Technique was not necessary

* Chrome 14
URL Blacklisting Services

- Intent: Early warning system for fast-flux malware
  - IE: MS Phishing filter -> MS URS / SmartScreen Filter
  - Google SBL, used by Chrome, FF, Safari

- Similar goals, some implementation differences
  - SBL: Sourced from crawl data, public submissions
  - MS URS: Numerous private feeds, public submissions

- We tested both services against public malware URL feeds
  - BLADE, MalwareBlacklist, MalwareDomains, MalwarePatrol
  - We wanted to use public, attributable sources
Blacklisting Services (cont.)

- 3086 average unique live URLs per day
  - 404 vs 405 matches for SBL vs URS
  - Interestingly, 42 SBL URLs also in URS
  - No URS URLs in SBS

<table>
<thead>
<tr>
<th>Date</th>
<th>7/23</th>
<th>7/24</th>
<th>7/25</th>
<th>7/26</th>
<th>7/27</th>
<th>7/28</th>
<th>7/29</th>
<th>7/30</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google SBL Matches</td>
<td>409</td>
<td>411</td>
<td>411</td>
<td>422</td>
<td>393</td>
<td>396</td>
<td>397</td>
<td>404</td>
<td>405</td>
</tr>
<tr>
<td>Microsoft URS Matches</td>
<td>361</td>
<td>336</td>
<td>364</td>
<td>371</td>
<td>401</td>
<td>447</td>
<td>499</td>
<td>450</td>
<td>404</td>
</tr>
<tr>
<td>Total URLs</td>
<td>5684</td>
<td>5724</td>
<td>5738</td>
<td>6128</td>
<td>6145</td>
<td>6089</td>
<td>6149</td>
<td>6025</td>
<td>5960</td>
</tr>
<tr>
<td>Live URLs</td>
<td>2993</td>
<td>2948</td>
<td>3040</td>
<td>3416</td>
<td>3128</td>
<td>3043</td>
<td>3115</td>
<td>3003</td>
<td>3086</td>
</tr>
</tbody>
</table>
Blacklisting Services (cont.)

- Both only ID a fraction of our sample set. What gives?
  - Apparently, malware SIGINT is really hard
  - Sharing info / collaboration could help
  - Still, it’s clear neither of these services is a panacea
Vulnerability Statistics

- Difficult if not impossible to make clear comparisons here
  - Privately disclosed bugs, rollups, internal discoveries
  - Timelines and vagaries, severity metrics
  - We discarded what wasn’t clearly measurable, normalized the data
Vuln Stats (cont.)

- One fairly reliable and interesting metric is time to patch
  - Again, based only on what we could normalize

![Average Time to Patch Chart]

- Firefox: 158
- Internet Explorer: 214
- Chrome: 53
Conclusions

- Every browser has improved over the last 4 years
  - Diversity and the browser wars have benefited end users
- Most of the yardsticks are broken
  - Security models are hard to make charts from
- We believe, that the best defended browser is the most payload-hostile one
Conclusions (cont.)

- In the long run, no disinfectant like sunlight
  - Without transparency, there’s no real debate on this topic
  - We shared our tools and data, anyone is welcome to debate the merit of our work, regardless of funding

- We’re proud of the dialogue and conversation we created
  - We hope we’ve set a precedent in publishing our test data
  - Please expand our research! We might even help!