Browser security issues and solutions

András Gazdag
“We spend way more time looking at our browsers that we do our wives, husbands, partners, friends, pet Iguana’s etc.”

– Concise Courses
Browser History

- Opera: 1994
- Internet Explorer: 1995
- Firefox (Phoenix): 2002
- Safari: 2003
- Chrome: 2008
Security breaches

- Operating system is breached and malware is reading/modifying the browser memory space in privilege mode
- Operating system has a malware running as a background process, which is reading/modifying the browser memory space in privilege mode
- Main browser executable can be hacked
- Browser components may be hacked
- Browser plugins can be hacked
- Browser network communications could be intercepted outside the machine
Legacy

- monolithic architecture
- single protection domain
Dangers

- HTML parser
- JavaScript Virtual Machine
- Document Object Model (DOM)
Design Goals

• Remain compatible

• treats the rendering engine as a black box

• does not prevent an attacker who compromises the rendering engine […] Instead, the architecture aims to prevent an attacker from reading or writing the user’s file system

• If an attacker exploits such a vulnerability in the rendering engine, Chromium’s architecture aims to restrict the attacker to using the browser kernel interface.
Achieved Goals

- Chromium’s security architecture mitigates approximately 70% of critical browser vulnerabilities that let an attacker execute arbitrary code

- Implementation passes 99% of 10,115 compatibility tests from the WebKit project.
Threat Model

- The attacker knows an unpatched security vulnerability in the user’s browser and is able to convince the user’s browser to render malicious content
  - owns a domain name
  - has a valid HTTPS certificate
  - attacker is able to convince the user to visit his or her web site
  - The attacker knows, and is able to exploit, an unpatched arbitrary code execution vulnerability in the user’s web browser
Security Goals

• Persistent Malware: attempt to install malicious software

• Transient Keylogger: keylogger need not survive the user closing the browser.

• File Theft
“an attacker who is able to install malware is no longer constrained by the browser’s security policy and often said to “own” the user’s machine”
Out-of-Scope Goals

- Phishing
  - location bar highlights the web site’s domain name
  - black-lists known phishing sites

- Origin Isolation
  - an attacker can obtain the cookies for every web site and can read all the passwords stored in the browser’s password database
Out-of-Scope Goals

- Firewall Circumvention
  - restrict an attacker’s network access from within the browser
  - the ability to request arbitrary web URLs follows the compatibility and black-box design decisions in order to support stylesheets and image tags

- Web Site Vulnerabilities
  - does not protect an honest web site if the site contains cross-site scripting (XSS), cross-site request forgery (CSRF), or header injection vulnerabilities
  - supports HttpOnly cookies
Chromium’s Architecture

• Rendering engine: is responsible for converting HTTP responses and user input events into rendered bitmaps

• Browser kernel: responsible for interacting with the operating system.
Chromium’s Architecture

• Rendering engine: act as the web
• Browser kernel: act as the user
Rendering Engine

- interprets and executes web content by providing default behaviors
- responsible for enforcing the same-origin policy (helps prevent malicious web sites from disrupting the user’s session with honest web sites)
- interacts most directly with untrusted web content
- HTML parsing, image decoding, and JavaScript parsing
- runs in a sandbox
Browser Kernel

- responsible for managing multiple instances of the rendering engine
- implements the browser kernel API
  - tab-based windowing system
  - bookmarks, cookies, and saved passwords
  - maintains state information about the privileges it has granted to each rendering engine
  - interacts with the network
Decisions

• security, compatibility, and performance

• Exceptions (historical artifacts):
  
  • Kernel:
    
    • displays JavaScript alert
  
    • manages cookie database (because of its direct access to the file system)
  
  • Rendering Engine:
    
    • regular expressions (performance)
Decisions

• Exceptions (historical artifacts):

  • Kernel:

    • HTTP stack (could be allocated to the rendering engine, at the cost of complicating the network stack and lowering performance)

  • Both:

    • both the browser kernel and the rendering engine parse URLs
<table>
<thead>
<tr>
<th>Rendering Engine</th>
<th>Browser Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTML parsing</td>
<td>Cookie database</td>
</tr>
<tr>
<td>CSS parsing</td>
<td>History database</td>
</tr>
<tr>
<td>Image decoding</td>
<td>Password database</td>
</tr>
<tr>
<td>JavaScript interpreter</td>
<td>Window management</td>
</tr>
<tr>
<td>Regular expressions</td>
<td>Location bar</td>
</tr>
<tr>
<td>Layout</td>
<td>Safe Browsing blacklist</td>
</tr>
<tr>
<td>Document Object Model</td>
<td>Network stack</td>
</tr>
<tr>
<td>Rendering</td>
<td>SSL/TLS</td>
</tr>
<tr>
<td>SVG</td>
<td>Disk cache</td>
</tr>
<tr>
<td>XML parsing</td>
<td>Download manager</td>
</tr>
<tr>
<td>XSLT</td>
<td>Clipboard</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Both</strong></td>
<td></td>
</tr>
<tr>
<td>URL parsing</td>
<td></td>
</tr>
<tr>
<td>Unicode parsing</td>
<td></td>
</tr>
</tbody>
</table>
Process Granularity

• Chromium uses a separate instance of the rendering engine for each tab

• fault tolerance in the case of a rendering engine crash

• main exception to this pattern is the Web Inspector
Plug-ins

- each plug-in runs in a separate host process
- outside both the rendering engines and the browser kernel
- at most one instance of a plug-in for the entire web browser
- by default, each plug-in runs outside of the sandbox and with the user’s full privileges
- Chromium also contains an option to run existing plug-ins inside the sandbox (- -safe-plugins)
Sandbox

- Chromium runs each rendering engine in a sandbox
- restricts issuing some system calls
- force the rendering engine to use the browser kernel API to interact with the outside world (XMLHttpRequest)
Sandbox

• Limitations:

• The FAT32 file system does not support access control lists (USB thumb drive)

• Misconfigured Objects: if an object has a NULL discretionary access control list (DACL), the Windows security manager will grant access without considering the accessing security token.

• TCP/IP: the rendering engine could create a TCP/IP socket on Windows XP because the low-level system calls to open a socket do not appear to require OS handles or to perform access checks
Browser Kernel Interface

- the rendering engine relies on the browser kernel API

- must be carefully designed not to grant more privileges than are necessary

(in the X Window System, the ability to create a window on an X server also implies the ability to monitor all of the user’s keystrokes)
Browser Kernel Interface

- Rendering: instead of granting the rendering engine direct access to a window handle, the rendering engine draws into an off-screen bitmap.

- User Input: instead of delivering user input events directly to the rendering engine, the operating system delivers these events to the browser kernel.
Browser Kernel Interface

- Persistent Storage: the sandbox is responsible for ensuring that the rendering engine cannot access the user’s file system directly
  
  - Uploading:
    
    - The browser kernel is responsible for displaying the file picker dialog
    
    - These authorizations last for the lifetime of the rendering engine
  
  - Downloading:
    
    - the rendering engine uses the browser kernel API to download URLs
    
    - blacklists certain kinds of file names
    
    - blacklists certain kinds of file names
Browser Kernel Interface

• Networking:

  • the rendering engine retrieves URLs from the network via the browser kernel

  • the browser kernel prevents most rendering engines from requesting URLs with the “file://“ scheme

• development: these documents are rendered in a dedicated rendering engine
Security Evaluation

- between July 1, 2007 and July 1, 2008
- Internet Explorer patched 19 vulnerabilities, Firefox patched 60 vulnerabilities, and Safari patched 50 vulnerabilities

<table>
<thead>
<tr>
<th></th>
<th>Browser</th>
<th>Renderer</th>
<th>Unclassified</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet Explorer</td>
<td>4</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>Firefox</td>
<td>17</td>
<td>40</td>
<td>3</td>
</tr>
<tr>
<td>Safari</td>
<td>12</td>
<td>37</td>
<td>1</td>
</tr>
</tbody>
</table>
Security Evaluation

- the rendering engine is more complex than the browser kernel!

<table>
<thead>
<tr>
<th></th>
<th>Browser</th>
<th>Renderer</th>
<th>Unclassified</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet Explorer</td>
<td>1</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>Firefox</td>
<td>5</td>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td>Safari</td>
<td>5</td>
<td>10</td>
<td>0</td>
</tr>
</tbody>
</table>

Rendering Engine: 1,000,000 lines of code
Kernel: 700,000 lines of code

- Chromium’s architecture helps mitigate these vulnerabilities!
Vulnerabilities

- the majority (8 of 11) of these vulnerabilities were caused by insufficient validation of inputs to system calls and not by buffer overflows or other memory-safety issues.

- other techniques are required to mitigate these issues!

- remaining vulnerabilities would not have been mitigated by additional sandboxing, suggesting that assigning more tasks to the rendering engine would not significantly improve security.
Case Study: Xml eXternal Entity (XXE) attack

• The attacker can then read the content, circumventing a confidentiality goals of the browser’s security policy
Case Study: Xml eXternal Entity (XXE) attack

• rendering engine’s use of libXML was vulnerable to XXE attacks

• from the “file://“ scheme, then the browser kernel blocked the request
XXE Discussion

- By parsing web content in the sandboxed rendering engine, Chromium’s security architecture mitigated an unknown vulnerability.
- Sandbox did not completely defend against the XXE vulnerability.
- The architecture also helps mitigate less-severe vulnerabilities that lead to partial compromises of the rendering engine.
Other modular browser architectures

- SubOS isolates web pages with different URLs

- SubOS incompatible with many web sites.

- DarpaBrowser: uses an object capability discipline to grant an untrusted rendering engine a limited set of capabilities

- non-black-box architecture prevents the DarpaBrowser from being compatible with many web sites (e.g., those that navigate using JavaScript)
Other modular browser architectures

• Tahoma runs each “site” in a separate protection domain, isolated using a virtual machine monitor

• To make use of these isolation features, a web site operator must opt-in by publishing a manifest file

• If the user incorrectly approves a malicious web site, that web site can steal confidential documents

• OP browser uses a separate protection domain for each web page and implements

• the JavaScript interpreter, the network stack, and the cookie store in separate modules.

• does not support a number of widely used browser features, such as inter-frame scripting, downloads, and uploads.
Other modular browser architectures

- Internet Explorer 8: runs tabs in separate processes, each of which runs in protected mode
  - does not seek to protect the confidentiality of the user’s file system
- VMware browser appliance
  - prevents from reading any of the user’s files, even files the user wishes to upload to web sites
Extension Security

- Privilege separation: Every Chrome extension is composed of two types of components: scripts and extension
  - Content scripts read and modify websites as needed
  - Core extensions can access Chrome’s extension API
Firefox

- Page DOM
  - Attacker
  - Isolated World
- Content Script
- Extension Core
  - Tabs
  - Bookmarks
- Native Binary
  - Process Creation
  - File System
Extension Security

• Isolated worlds: The isolated worlds mechanism is intended to protect content scripts from web attackers

• Permissions: By default, extensions cannot use parts of the browser API that impact users’ privacy or security
Extension Security

• isolated worlds mechanism is highly effective

• permissions can have a significant positive impact on system security

• multiple threats should be considered when initially designing a system

• the most dangerous class of extensions (extensions with the privilege to execute arbitrary code) are not permitted in the gallery unless the developer signs a contract with Google
September 2013.
Efficiency

- multi-process architecture if you have a multi-core CPU
- single-process architecture
Sandbox

✓

∅
Just-In-Time Hardening
(keeps the browser from compiling JavaScript that cannot be run on the user's computer)
Do Not Track privacy on multiple platforms

∅  vs  ✓
Conclusion

• “Use telnet…” (by Pinyó)
• Chrome
• IE 10
Browser hardening

• Hardening: Browsing the Internet as a least-privilege user account

• NoScripts

• HTTPS Everywhere
“at PacSec Tokyo 2013. The first day of competition brought iOS and the Samsung Galaxy S4 down”

– Larry Seltzer
Latest News: Mobile Pwn2Own 2013
(November 13, 2013)

• iOS:
  • “…by getting the user to visit a web site, the attackers were able to steal the cookie database from the browser. From this they retrieved the user's Facebook credentials and logged in using them on a different computer.”
  • “The iOS 7.0.3 exploit relied on a flaw in the permissions model. Once the user visited a page, the attackers were able to steal a photo from the phone.”
Latest News: Mobile Pwn2Own 2013
(November 13, 2013)

• “Neither phone was jailbroken. But Keen was not able to break out of the sandbox, so their award was limited to $27,500."
Latest News: Mobile Pwn2Own 2013
(November 13, 2013)

• Android Samsung Galaxy S4:
  • “Japan’s very own Team MBSD, […] have demonstrated exploits against several applications installed by default on the Samsung Galaxy S4. Combined, these bugs allow the covert installation of a malicious application and the theft of sensitive data. The spoils for their hard work? A cool $40,000.”
Browser security issues and solutions

András Gazdag