Building a Safer Web

Charles Reis

University of Washington CSE
Web is Evolving

- More complex, active content
- Browser now in role of OS, but not yet safe
  - Browsers aren’t built for programs
- Web content faces real challenges
# My Contributions

<table>
<thead>
<tr>
<th>Problems</th>
<th>Projects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program Interference</td>
<td>Multi-Process Browsers</td>
</tr>
<tr>
<td>In-Flight Page Changes</td>
<td>Web Tripwires</td>
</tr>
<tr>
<td>Poor Program Support</td>
<td>Architectural Principles</td>
</tr>
<tr>
<td>XSS</td>
<td>Script Whitelists</td>
</tr>
<tr>
<td>Browser Exploits</td>
<td>BrowserShield</td>
</tr>
</tbody>
</table>

*Note: [EuroSys ’09], [NSDI ’08], [OSDI ’06], [HotNets ’07]*
# Range of Project Types

<table>
<thead>
<tr>
<th>Program Interference</th>
<th>Multi-Process Browsers</th>
<th>Practical, deployed in Google Chrome</th>
</tr>
</thead>
<tbody>
<tr>
<td>In-Flight Page Changes</td>
<td>Web Tripwires</td>
<td>Measurement study of 50,000 clients</td>
</tr>
<tr>
<td>Poor Program Support</td>
<td>Architectural Principles</td>
<td>Position paper</td>
</tr>
<tr>
<td>XSS</td>
<td>Script Whitelists</td>
<td>Research prototype</td>
</tr>
<tr>
<td>Browser Exploits</td>
<td>BrowserShield</td>
<td>Prototype, influenced Web Sandbox</td>
</tr>
</tbody>
</table>
Consider OS Landscape

- Performance isolation
- Resource management
- Failure isolation
- Clear program abstraction
Browsers Fall Short

- Unresponsiveness
- Jumbled accounting
- Browser crashes
- Unclear what a program is!
Thesis: Learn from the OS

- Improve browser and web content architecture
  - Define a precise program abstraction
  - Isolate programs from each other
  - Make it possible to authorize program code
  - Interpose on program behavior
Outline

- Browser Architecture: Chromium
  - Define program abstractions
  - Isolate programs from each other

Web Tripwires

Previous Work

Future Directions
Programs in the Browser

Consider an example browsing session

- Several independent programs
Monolithic Browsers

- Most browsers put all pages in one process
- Poor performance isolation
- Poor failure isolation
- Poor security
- Should re-architect the browser
Process per Window?

- **Breaks pages** that directly communicate
- **Shared access to data structures, etc.**
- **Connected pages** from **same-origin**
- **Fails as a program abstraction**
Need a Program Abstraction

- Aim for **new groupings** that:
  - Match our intuitions
  - Preserve compatibility
- Take cues from browser’s existing rules
- Isolate each grouping in an OS process
- Will get **performance and failure isolation**, but not security between sites
Outline

- Browser Architecture
- Program Abstractions
- Program Isolation
- Evaluation
Ideal Abstractions

- **Web Program**
  - Set of pages and sub-resources providing a service

- **Web Program Instance**
  - Live copy of a web program in the browser
  - Will be isolated in the browser’s architecture

*Intuitive, but how to define concretely?*
Compatible Abstractions

Three ways to group pages into processes:

1. **Site:** based on browser’s access control policies

2. **Browsing Instance:** communication channels between pages

3. **Site Instance:** intersection of the first two
1. Sites

- Sites
- **Same Origin Policy** dictates some isolation (host+protocol+port)
  - Pages can change document.domain
  - Registry-controlled domain name limit
  - **Site**: RCDN + protocol

<image>

<example>

- mail.zoho.com
- docs.zoho.com
- https://zoho.com
- http://bbc.co.uk

<diagram>

<summary>

- Mail
- Doc List
- Doc
- Blog
- News
- Article

</summary>
2. Browsing Instances

- Not all pages can talk
- References between “related” windows
- Parents and children
- Lifetime of window
- **Browsing Instance**: connected windows, regardless of site
3. Site Instances

- **Site Instance:** Intersection of site & browsing instance
- Safe to isolate from any other pages
- Compatible notion of a web program instance
Abstractions Recap

- **Site**
  - e.g., All pages from https://bbc.co.uk

- **Browsing Instance**
  - Windows with script references to each other

- **Site Instance**
  - Connected, same-site pages
Compatiblity Compromises

- Coarse granularity
  - Some logical apps grouped together (instances help)
- Imperfect isolation
  - Shared cookies, some window-level JS calls
- Not a secure boundary
  - Must still rely on renderer to prevent certain leaks
Most Browsers are Monolithic

- All browser parts in one process
- Could divide into separate modules
  - **Isolate with OS processes:** address spaces, concurrency, failure isolation
Multi-Process Browser

- **Browser Kernel**
  - Storage, network, UI
- **Rendering Engines**
  - Web program and runtime environment
- **Plug-ins**
Implementations

- **Konqueror Prototype** (2006)
  - Proof of concept on Linux

- **Chromium** (Google Chrome, 2008)
  - Added support for Site Instance isolation (including creating processes during navigations)
Chromium Process Models

1. Monolithic

2. Process-per-Browsing-Instance
   - New window = new renderer process

3. Process-per-Site-Instance (default)
   - Create renderer process when navigating cross-site

4. Process-per-Site
   - Combine instances: fewer processes, less isolation
Implementation Caveats

- Sites may sometimes share processes
  - Not all cross-site navigations change processes
  - Frames still in parent process
  - Process limit (20), then randomly re-used
Robustness Benefits

- Failure Isolation
- Accountability
- Memory Management

- Some additional security (e.g., Chromium’s sandbox)
Performance Isolation

- **Responsive** while other web programs working
- No click latency

![Performance Chart]

<table>
<thead>
<tr>
<th></th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>With Top 5 Pages</td>
<td>Monolithic Chromium: 1,408</td>
</tr>
<tr>
<td>With Gmail</td>
<td>6</td>
</tr>
</tbody>
</table>

**Avg Click Delay on Blank Page**
Other Performance Impact

- **Speedups**
  - More work done concurrently, leveraging cores
  - e.g., Session restore of several tabs

- **Process Latency**
  - 100 ms, but masked by other speedups in practice
Memory Overhead

- Robustness benefits do have a cost
- Reasonable for many real users
Compatibility Evaluation

- No known compat bugs due to architecture
- Distributed tests check top million pages
- Some minor behavior changes
  - e.g., **Narrower scope of window names:** browsing instance, not global
Related Architecture Work

- Internet Explorer 8
  - Multi-process architecture, no program abstractions
- Gazelle
  - Like Chromium, but values security over compatibility
- Other research: OP, Tahoma, SubOS
  - Break compatibility (isolation too fine-grained)
Summary

- Browsers must recognize programs to support them
  - Site Instances capture this
  - Compatible with existing web content
  - Can prevent interference with process isolation
Outline

- Web Tripwires
  - Simple integrity checks to protect programs

Browser Architecture

Previous Work

Future Directions
Web Program Integrity

- Can users or publishers trust web program contents?
  - HTTP can be modified in-flight
  - Changes become part of the site instance
Is this a concern?

- Measurements say it is!
  - Of 50,000 clients, 1% saw in-flight changes (653)
  - Ads, exploits, broken pages, new vulnerabilities
Detecting Page Changes

- Can detect with JavaScript

- Built a **Web Tripwire:**
  - Runs in client’s browser
  - Finds most changes to HTML
  - Reports to user & server

http://vancouver.cs.washington.edu
Measurement Study

- Wanted view of many clients on many networks
  - Posted to Slashdot, Digg, etc.
    - Visits from over 50,000 unique IP addresses
    - 653 reported changes

http://vancouver.cs.washington.edu
Diverse Changes Observed

Ad Injection (Free wireless, NebuAd, etc)

Security Checks (Enterprises)

Exploits (ARP poisoning)

Ad / Popup Blockers (on client)

http://vancouver.cs.washington.edu
The best intentions...

- Bugs introduced
  - Web forums broken by popup blockers

- Vulnerabilities introduced
  - Ad blocker code vulnerable to XSS
  - User’s web programs are the victims!

http://vancouver.cs.washington.edu
Web Tripwires for Publishers

- HTTPS too costly for some sites
- Can detect changes with JavaScript
- Easy for publishers to deploy
  - Configurable toolkit
  - Web tripwire service

http://vancouver.cs.washington.edu
Summary

- Not safe to blindly patch code of web programs
- Many parties with incentives to do so
- Publishers can detect it with **web tripwires**
Outline

Browser Architecture

Web Tripwires

Previous Work

Future Directions
BrowserShield [OSDI ’06]

- **Block exploits** of known browser vulnerabilities
- Interpose to enforce flexible policies
- Rewrites JavaScript code in-flight
- Has influenced Live Labs’ Web Sandbox
Earlier Research

- **Wireless Networking**
  - Study low-level 802.11 behavior [EWIND ‘05]
  - Predict behavior from measurements [SIGCOMM ‘06]

- **Education with DrJava**
  - Teach production programming [SIGCSE ‘03]
  - Simplify Eclipse for students [SIGCSE ‘04]
Outline

Browser Architecture

Web Tripwires

Previous Work

Future Directions
Short Term Directions

- **Secure + Compatible isolation** of Site Instances
  - Better ways to evaluate compatibility
- **Opt-in mechanisms** for secure web apps
  - e.g., Alternatives to Same Origin Policy
- **Enforcing policies** on content, plug-ins, extensions
Long Term Directions

- What will **networked applications** look like?
  - How will browsers & OSes evolve to support them?
- How will **trust models** change?
  - How to grant some programs more rights?
- **Robust and secure systems** in general
Conclusion

- Web is becoming an application platform
- Browser architectures must support programs
- Web publishers must protect content
- Great opportunity to reshape the web
Relevant for security?

- Pages are free to embed objects from any site
  - Scripts, images, plugins
  - Carry user’s credentials
  - Inaccessible info within each Site Instance
- Compatibility makes us rely on internal logic