Isolating Programs in Modern Browser Architectures

Charles Reis, Steven D. Gribble

University of Washington / Google, Inc.
Web is Evolving

- More complex, active content
- Browser now in role of OS, but not designed for it
  - Robustness and performance problems
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!
Outline

Looking for Programs

New Abstractions

Isolation in Chromium

Evaluation
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.
- **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
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

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

```javascript
w = window.open(...)
window.opener
```

Mail  Doc List  Doc
Blog  News  Article
3. Site Instances

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

Looking for Programs

New Abstractions

Isolation in Chromium

Evaluation
Multi-Process Browser

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

- Implemented in Chromium
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
Outline

- Looking for Programs
- New Abstractions
- Isolation in Chromium
- Evaluation
Robustness Benefits

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

- **Responsive** while other web programs working

![Graph showing Avg Click Delay on Blank Page]

- Monolithic Chromium
- Multi-Process Chromium

<table>
<thead>
<tr>
<th></th>
<th>With Top 5 Pages</th>
<th>With Gmail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg Click Delay (ms)</td>
<td>1,408</td>
<td>3,307</td>
</tr>
<tr>
<td>Time (ms)</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>
Other Performance Impact

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

- **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
- 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)
Conclusion

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

*Implemented in Chromium*
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
Compatibility 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
Implementation Caveats

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