Improving the Security and Robustness of Modern Web Browsers

Charles Reis
General Examination
May 14, 2007
Is Browsing Safe?
Is Browsing Safe?

Web Mail

Movie Rentals

Search Results
Is Browsing Safe?

Web Mail

Search Results

Movie Rentals

Install Malware
Is Browsing Safe?

Web Mail

Send Spam

Movie Rentals

Search Results

? Install Malware
Is Browsing Safe?

Web Mail

Movie Rentals

Search Results

Change Address

Install Malware

Send Spam
Is Browsing Safe?

Web Mail

Movie Rentals

Search Results

Install Malware

Send Spam

Crash Browser
How did we get here?
How did we get here?

- Browsers have evolved
How did we get here?

- Browsers have evolved
How did we get here?

- Browsers have evolved
- Now analogous to OS
How did we get here?

- Browsers have evolved

Runtime Environment

- Now analogous to OS
- Convenience has trumped security
Hypothesis

Mechanisms from OS research can improve the security and robustness of modern web browsers.
Hypothesis

Mechanisms from OS research can improve the security and robustness of modern web browsers.

- Focus on Isolation and Interposition
Hypothesis

Mechanisms from OS research can improve the security and robustness of modern web browsers.

• Focus on Isolation and Interposition
• Evaluate Safety, Backwards Compatibility, and Efficiency
I. Browser Vulnerabilities
1. Browser Vulnerabilities

- Can run arbitrary code
- Windows .ANI bug
1. Browser Vulnerabilities

- Can run arbitrary code
- Windows .ANI bug
- Patches get delayed
1. Browser Vulnerabilities

- Can run arbitrary code
- Windows .ANI bug
- Patches get delayed

**Existing Proposals:**
- VM sandboxes \([\text{Tahoma}]\),
- pre-screening \([\text{SpyProxy}]\),
- legal teams \([\text{HoneyMonkey}]\)
1. Browser Vulnerabilities

- Can run arbitrary code
- Windows .ANI bug
- Patches get delayed
- **Existing Proposals:**
  - VM sandboxes \textit{Tahoma],
  - pre-screening \textit{SpyProxy},
  - legal teams \textit{HoneyMonkey]
- **Proposal**: filter exploits
2. Script Injection (XSS)
2. Script Injection (XSS)
2. Script Injection (XSS)

- Can subvert trusted sites
- Yahoo Mail, MySpace
2. Script Injection (XSS)

- Can subvert trusted sites
- Yahoo Mail, MySpace
- Input validation is hard
2. Script Injection (XSS)

- Can subvert trusted sites
- Yahoo Mail, MySpace
- Input validation is hard
- **Existing Proposals:**
  - Server-side analysis \([Xie 06]\),
  - client firewalls \([Noxes]\)
2. Script Injection (XSS)

- Can subvert trusted sites
- Yahoo Mail, MySpace
- Input validation is hard

**Existing Proposals:**
- Server-side analysis [Xie 06]
- Client firewalls [Noxes]

**Proposal:** whitelists
3. Cross-Site Request Forgery (CSRF)
3. Cross-Site Request Forgery (CSRF)

- Can abuse user credentials
- Netflix, GMail
3. Cross-Site Request Forgery (CSRF)

- Can abuse user credentials
- Netflix, GMail
- Add tokens to web forms
3. Cross-Site Request Forgery (CSRF)

- Can abuse user credentials
- Netflix, GMail
- Add tokens to web forms
- **Existing Proposals:**
  - Proxies [Jovanovic 06], block suspicious requests [RequestRodeo]
3. Cross-Site Request Forgery (CSRF)

- Can abuse user credentials
- Netflix, GMail
- Add tokens to web forms
- **Existing Proposals:**
  - Proxies [Jovanovic 06], block suspicious requests [RequestRodeo]
- **Proposal:** browser sessions
4. Resource Contention

Browser
4. Resource Contention
4. Resource Contention

- Unresponsiveness, crashes
4. Resource Contention

- Unresponsiveness, crashes
- **Existing Proposals:** Separate VMs with manifests

Browser
4. Resource Contention

- Unresponsiveness, crashes
- Existing Proposals: Separate VMs with manifests
- Proposal: OS processes
Isolating Web Content
Isolating Web Content

- Some threats addressed by using multiple browsers
Isolating Web Content

- Some threats addressed by using multiple browsers

3. CSRF
Isolating Web Content

- Some threats addressed by using multiple browsers
  
  3. CSRF
  
  4. Resource contention
Isolating Web Content

- Some threats addressed by using multiple browsers
  3. CSRF
  4. Resource contention

**Goal:** capture this idea within a single browser
Isolation in OS

• Processes

• Lightweight fault domains
  [Wahbe 93, Swift 03]

• Covert channels
  [Lampson 73]
Proposal:

Decompose Browser
Proposal:

Decompose Browser

A.com

A.com

A.com

B.com

A.com
Proposal:

Decompose Browser

Runtime Environments

A.com ➔ A.com ➔ A.com ➔ B.com

User Interface
Proposal:

Host

A.com  A.com  A.com  B.com
Host

- All documents from a hostname

Proposal:
Proposal:

Host

- All documents from a hostname
- **Preliminary work**: process per host
Host

- All documents from a hostname
- **Preliminary work:** process per host
- **Proposal:** partition of storage
- Cache [Felten 00], visited links [Jackson 06], persistent cookies
Proposal:

Session

A.com  A.com  A.com  B.com
Proposal:

Session

- All documents from a host with either:
  - Navigational relationship
  - Parent-child relationship
Proposal:

Session

• All documents from a host with either:
  • Navigational relationship
  • Parent-child relationship
• Separate session cookies *(no exp. date)*
Session

Proposal:

- All documents from a host with either:
  - Navigational relationship
  - Parent-child relationship
  - Separate **session cookies** (no exp. date)
  - Separate runtime environments
Isolation with Processes

Proposal:
Methodology

- Implement with KDE web browser

Challenges:
- What to isolate or share across sessions?
- How to keep overhead low?
- Will use of cookies need to change?
Proposal:

Evaluation
Evaluation

- **Safety**: test CSRF and contention attacks
- Use both crafted and observed pages
Proposal:

Evaluation

- **Safety**: test CSRF and contention attacks
  - Use both crafted and observed pages
- **Back Compat**: test popular content
  - Compare loaded objects, JS errors
  - Characterize use of cookies
Proposal:

Evaluation

- **Safety**: test CSRF and contention attacks
  - Use both crafted and observed pages

- **Back Compat**: test popular content
  - Compare loaded objects, JS errors
  - Characterize use of cookies

- **Efficiency**: overhead of sessions, calls
  - Are lightweight domains needed?
Interposition & Policies

- Fixed policy doesn’t block all threats
- **Extensible security architecture**
  [Wallach 97]

1. **Defend browser vulnerabilities**
2. **Block XSS attacks**
Interposition in OS

- System call interposition
  [Goldberg 96, Garfinkel 04]

- Code rewriting
  [Erlingsson 00]

- Vulnerability Filtering
  [Wang 04]
Preliminary Work:

**BrowserShield**

- Policies
- BrowserShield Rewriter
- Browser
Preliminary Work:

**BrowserShield**

- **Code Rewriting**
- **Interpose on HTML and JavaScript code**

```
BrowserShield Rewriter

Policies

Browser
```
Preliminary Work:

BrowserShield

- **Code Rewriting**
- Interpose on HTML and JavaScript code

- **Vulnerability Policies**
- Block all exploits of known vulnerabilities

BrowserShield Rewriter

Policies

Browser
Preliminary Work:

Script Whitelists

[Diagram showing a script flowing into a JavaScript Engine]
Preliminary Work:

Script Whitelists

- Intercept JavaScript
- Independent of quirky HTML parsing
Script Whitelists

- Intercept JavaScript
- Independent of quirky HTML parsing
- Page provides whitelist
- Prevents XSS attacks
- Practical to deploy

Preliminary Work:
Proposal:

Interposition Layer

- HTML
- JS

Interposition Layer

- Renderer
- JS Engine

DOM

Interposition Layer
Proposal:

Interposition Layer

- Interpose within browser

Diagram:

- HTML
- JS
- Interposition Layer
- Renderer
- JS Engine
- Interposition Layer
- DOM
Proposal:

Interposition Layer

- Interpose *within* browser
- Uniform policies for web content of all formats
Proposal:

Interposition Layer

- Interpose within browser
- Uniform policies for web content of all formats
Proposal:

Interposition Layer

- Interpose within browser
- Uniform policies for web content of all formats
- Expose hooks for policies:
  - Raw input (e.g., HTML), DOM access, Communication

Diagram:

- HTML
- JS
- Flash

Interposition Layer

- Renderer
- JS Engine
- Flash, etc.

DOM
Security Policies

- **Policy hierarchy**
  - Browser, extensions, sessions, documents

- **Build sample policies**
  - Vulnerability shields
  - Script whitelists
Methodology

- Implement in Firefox (extension API)

Challenges:
- How to specify policies?
- Expressive with low overhead?
- How much can plugins be confined?
Proposal:

Evaluation
Evaluation

- **Safety**: can it support safer policies?
- Test defense against exploits, XSS
Proposal:

Evaluation

- **Safety**: can it support safer policies?
  - Test defense against exploits, XSS

- **Back Compatt**: do policies break pages?
  - Same popular content tests
Proposal:

**Evaluation**

- **Safety**: can it support safer policies?
  - Test defense against exploits, XSS

- **Back Compatability**: do policies break pages?
  - Same popular content tests

- **Efficiency**: overhead of layer and policies?
  - Micro/macro benchmarks
Future Directions
Future Directions

- **Communication** between sites
- *Better support for “mashups”*
Future Directions

• Communication between sites

• Better support for “mashups”

• Support for new phishing defenses

• Visual indicators of sessions
Future Directions

- **Communication** between sites
  - Better support for “mashups”
- Support for new **phishing defenses**
  - **Visual indicators of sessions**
- Platform for deploying **security research**
  - **Distribute as policies**
Conclusion

• OS mechanisms can improve web browsers
  • **Isolation** prevents interference
  • **Interposition** allows flexible policies

• Will prevent threats with few costs