BrowserShield: Vulnerability-Driven Filtering of Dynamic HTML

Charles Reis
University of Washington

John Dunagan, Helen J. Wang, Opher Dubrovsky
Microsoft

Saher Esmeir
Technion
Web Based Attacks

- Web browser exploits are common
  - Buffer overflows, ActiveX flaws, etc.

  19 critical vulns, 8 patches in 2005
  16 critical vulns, 7 updates in 2005
Patches aren’t Enough

- Patch installation often delayed
  - Reboots, application restarts, enterprise testing

- Dangerous time window
  - Attackers reverse engineer patches
Shield as a Front Line

- Vulnerability-Driven Filtering [Wang et al, 04]
  - Block dangerous traffic using protocol analysis
- Easy to deploy or roll back
  - Restarts unnecessary
Shield as a Front Line

- Vulnerability-Driven Filtering [Wang et al, 04]
  - Block dangerous traffic using protocol analysis
- Easy to deploy or roll back
  - Restarts unnecessary
**Shield as a Front Line**

- **Vulnerability-Driven Filtering**
  [Wang et al, 04]
  - Block dangerous traffic using protocol analysis
- **Easy to deploy or roll back**
  - Restarts unnecessary
Useful for Browsers?

- Shield works for static HTML
- Script code can hide exploits
- Finding exploits is undecidable
  - Can’t know deterministically until runtime

```javascript
eval(codeStr);
```
Protect at Runtime

• Rewrite code to insert runtime checks
  - Similar to Inline Reference Monitors
    [Erlingsson, Schneider 00]
  - Address challenges for JavaScript

• Protect with vulnerability policies
Script Interposition

- Focus on JavaScript
  - VBScript, Flash similar
- Can guard DOM, ActiveX, extensions
Modifying Content

- Intercept HTML and JavaScript
- Rewrite into safe equivalents
- Apply policies at runtime
Deploying BrowserShield

- Can deploy anywhere before rendering:
  - Firewall (protect many users)
  - Browser extension (can see SSL traffic)
  - Web publishers (community web sites)
Deploying BrowserShield

- Can deploy anywhere before rendering:
  - Firewall (protect many users)
  - Browser extension (can see SSL traffic)
  - Web publishers (community web sites)
Deploying BrowserShield

- Can deploy anywhere before rendering:
  - Firewall (protect many users)
  - Browser extension (can see SSL traffic)
  - Web publishers (community web sites)
Talk Outline

Motivation and Approach

Example Policy

BrowserShield Design

Evaluation
Example: IFRAME Vuln.

- MS04-040 Vulnerability
  - Buffer overrun if name and src attributes are too long
  - Affected iframe, frame, embed tags

```html
<iframe src="xxxxxxxxx...." name="xxxxxxxxx...."/>
```
IFRAME Policy

• Simple JavaScript snippet to identify exploits

• BrowserShield must apply policy to all vulnerable tags
  - No false negatives
  - No false positives

```javascript
function (tag) {
    var len = 255;
    if ((contains("name", tag.attrs) &&
        tag.attrs["name"].length > len) &&
        (contains("src", tag.attrs) &&
        tag.attrs["src"].length > len))
    {
        tag.attrs = [];
        return false; // Exploit found
    }
    return true; // Safe
}
```
Goals of BrowserShield

- Complete Interposition
- Tamper Proof
- Transparent
- Flexible Policies
Rewriting Logic

- Tokenize HTML
- Strip Exploits
- Wrap scripts for later translation
Rewriting Logic

T_{HTML}
- Tokenize HTML
- Strip Exploits
- Wrap scripts for later translation

T_{script}
- Translate scripts to access DOM via interposition layer
Rewriting Logic

\( T_{\text{HTML}} \)
- Tokenize HTML
- Strip Exploits
- Wrap scripts for later translation

\( T_{\text{script}} \)
- Translate scripts to access DOM via interposition layer

Policies
- Apply policies on all script actions
- Recursively apply \( T_{\text{HTML}} \) and \( T_{\text{script}} \)
doc.write(obj[str]);
**Script Example**

```javascript
doc.write(obj[str]);
```

```javascript
document.write(arg);
```
**T****s**cript **E****x**ample

```javascript
doc.write(obj[str]);
```

Object alias?

```javascript
document.write(arg);
```
Script Example

document.write(arg);

Object alias?  Method alias?

doc.write(obj[str]);
**Script Example**

```javascript
// Object alias? Method alias?

// Complete Interposition

document.write(arg);
```
T_script Example

(Object alias?  Method alias?)

(Complete Interposition)

bshield.invokeMeth(doc, “write”, obj[str]);
Tscript Example

```
Object alias?
Method alias?
Reflection?
```

```
doc.write(obj[str]);
```

```
bshield.invokeMeth(doc, "write", obj[str]);
```

(Complete Interposition)
**Script Example**

```java
bshield.invokeMeth(doc, "write", obj[str]);
```

- **Object alias?**
- **Method alias?**

(Complete Interposition)

```java
bshield.invokeMeth(doc, "write", obj[str]);
```

- **Reflection?**

(Transparent)

```java
bshield.invokeMeth(doc, "write", bshield.propRead(obj, str));
```
```
Object alias?
Method alias?
```

(Complete Interposition)

```
Reflection?
```

(Transparent)

```
bshield.invokeMeth(doc, “write”, obj[16][str]);
```

```
bshield.invokeMeth(doc, “write”,
  bshield.propRead(obj, str));
```

```
bshield.invokeMeth(doc, “write”,
  bshield.propRead(obj, str));
```

```
Same name space
```

(Tamper-Proof)
**T**script Example

```javascript
bshield.invokeMeth(doc, "write", obj[str]);
```

Object alias?  Method alias?  Reflection?

(Complete Interposition)  (Transparent)

```javascript
bshield.invokeMeth(doc, "write", obj[str]);
```

```javascript
bshield.invokeMeth(doc, "write", bshield.propRead(obj, str));
```

Same name space  Call THTML

(Tamper-Proof)  (Flexible Policies)
Complete Interposition

- Rewrite and apply policy to:
  - Function and method calls
  - Object property reads/writes
  - Object creations
Tamper Proof & Transparent

• Hide BrowserShield code
  - Rename variables, handle reflection
• Shadow copies of untranslated code
• Preserve context for “this”
Other Applications

• Useful beyond security policies:
  - Link translation
  - Dynamic content sandboxing
  - Anti-phishing mechanisms
Talk Outline

Motivation and Approach

Example Policy

BrowserShield Design

Evaluation
Implementation

- Firewall-based prototype:
  - ISA plugin: 2700 lines of C++
  - Client library: 3500 lines of JavaScript
  - Handled 3 types of vulnerabilities (HTML, script, and ActiveX)
**Vulnerability Coverage**

- Studied all 19 IE vulns (8 patches) in 2005

<table>
<thead>
<tr>
<th></th>
<th>HTTP filter + Antivirus</th>
<th>BrowserShield + HTTP + AV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vulnerability Coverage</td>
<td>5</td>
<td>19</td>
</tr>
<tr>
<td>Patch Equivalence</td>
<td>1</td>
<td>8</td>
</tr>
</tbody>
</table>
Performance Overhead

• **Firewall**: 22% increase in CPU
• **Client**:  
  - Typical interpreter behavior  
  - 250 pages weighted by popularity, measured 70 pages that worked
On average, 94% increase  (216% worst case)
- JavaScript-heavy pages still a challenge
Conclusions

- Script rewriting can protect web clients
  - Vulnerability-driven filtering
  - Transforms content, not browsers
- General framework
Acknowledgments

Valuable feedback from Ulfar Erlingsson, Bill Weihl, Alec Wolman, Steve Gribble, and anonymous reviewers