Advanced Systems Security: Web Security

Trent Jaeger
Systems and Internet Infrastructure Security (SIIS) Lab
Computer Science and Engineering Department
Pennsylvania State University

April 22, 2010
Problem

- The Internet
  - World-wide Web enabled anyone anywhere to talk
- What happens next?
What the Web Means to Me

- Server Side
- Web Server is a portal to a wide variety of application content
  - Web server design needs to be as general as possible
  - Otherwise, it would limit application developers
  - But, this means that the web server is like an OS for the web applications
- But is it?
What the Web Means to Me

- **Client Side**

- Web Client is a portal to a wide variety of application content
  - Web client design needs to be as general as possible
  - Otherwise, it would limit application developers
  - But, this means that the web client is like an OS for the web applications

- **But is it?**
Web Server Systems

- They started out so simple
- Servers produced static content on demand
- Clients render the content (no executable content)
Web Server Systems

- Then things got complicated
- Web servers compute dynamic contents
- And offload some of the computation to the client
Web Server Systems

- **Server-side**
- Receives input
  - From anyone in the world
- Submits it to a web application component
  - Processing defined by others
- Which may or may not protect itself from malicious input
  - But, web applications may have valuable data (your credit card numbers)
- What then…
Web Server Systems

- **Client-side**
- Receives input
  - From anyone in the world
- Some input is executable
  - Often, not clear what
- So, need to find and isolate execution
  - Except when executables need to interact
    - Need a policy to describe this (*Same-origin policy*)
- What then…
### Same Origin Policy

<table>
<thead>
<tr>
<th>Content Type</th>
<th>Fetch</th>
<th>Read</th>
<th>Modify</th>
<th>Execute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Images</td>
<td>Yes</td>
<td>SO</td>
<td>SO</td>
<td>No</td>
</tr>
<tr>
<td>HTML</td>
<td>Yes</td>
<td>SO</td>
<td>SO</td>
<td>No</td>
</tr>
<tr>
<td>Raw Data</td>
<td>Yes</td>
<td>SO</td>
<td>SO</td>
<td>No</td>
</tr>
<tr>
<td>JavaScript</td>
<td>Yes</td>
<td>Yes(^1)</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Styles</td>
<td>Yes</td>
<td>Yes(^1)</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Audio/Video (Plugins)</td>
<td>Yes</td>
<td>Plugin Dependent</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Audio/Video (HTML5)</td>
<td>Yes</td>
<td>SO</td>
<td>SO</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 3.1. Current JavaScript access to content loaded by the web browser (e.g., JavaScript may always fetch images, but can only read or modify the contents of an image if is tagged with the same origin (SO) as the JavaScript attempting the operation).
Some Web System Bugs

- Cross-site scripting

- Server receives malicious input, which it distributes to clients (as if from itself)
  - Web application does not adequately sanitize

- Same-origin policy does not prevent – why not?
Some Web System Bugs

- Cross-site request forgery
- Malicious (or compromised) server sends malicious content to client
  - That tries to trick the user to interact with a target
- Same-origin policy does not prevent – why not?
Some Web System Bugs

- **Clickjacking**

- Malicious (or compromised) server sends malicious content to client
  - That generates web requests to a target

- Same-origin policy does not prevent – why not?
Reference Monitor with MPS

• OK, so we can solve all problems with a reference monitor
  ▷ What is missing here?

• But, we need to enforce an MPS
  ▷ What is missing wrt that?

• Other problems?
  ▷ Hint: Not exactly satisfying Biba integrity
Securing Your Web Server

- 2001: http://windows.stanford.edu/docs/IISsecchecklist.htm
- 2005: http://www.petefreitag.com/item/505.cfm
- 2010: http://krebsonsecurity.com/2010/01/firm-to-release-database-web-server-0days/
• **Problem**: Web applications perform mission critical tasks, are accessible to attackers, and are built with ad hoc input validation
  - What integrity principles does this violate?

• **Solution**: More principled input validation

• **Lessons**: Overall approach and techniques
A Saner Approach

- Model the way that the application processes input values
- Check that model (statically) to identify likely errors in sanitization
- Use a runtime technique to verify legitimate errors in sanitization
Approaches

- Static taint checking

- **Approach**: Identify tainted sources and see whether they flow to sinks that require them to be untainted

- Identify tainted sinks
  - Label a program variable or expression (e.g., with a type)

- Identify untainted sinks
  - Label a program variable, expression, or function arg (e.g., with a type incompatible with tainted data)

- Type checking detects incompatibility
Problems

• Static taint checking problems

• Sinks have different requirements for sanitization
  ‣ Just changing from tainted to untainted is not expressive enough
  ‣ E.g., Protecting data structure integrity does not prevent SQL injection

• Identifying operations that do sanitization in code
  ‣ Some are well-known
  ‣ But, some are custom

• *Check sinks to identify whether malicious inputs reach*
Their Approach: Static

- Use automata to represent an arbitrary set of strings
  - Represent value and which parts are tainted
  - Find dependence among variables

![Figure 2. Automata for the string “Hello”, and for an unknown, tainted string.](image)

![Figure 3. Example dependence graph.](image)
Their Approach: Static

- Postorder traversal of the dependence tree
- Combine string automata based on descendants
  - Known function behaviors are modeled as precisely as possible to generate new automata
  - Unknown function behaviors generate "any string" of either type tainted or untainted
    - Former if any input is tainted
- Tricky to deal with cycles
Their Approach: Dynamic

- Static analysis may generate false positives
  - Is conservative, so some executions may be assumed that are not really possible
- Use runtime analysis to validate statically found positives
  - Which part to run? In what initial state?
- Generate runtime inputs and see if they cause problems
  - How to generate the runtime inputs?
Their Approach: Dynamic

- Build the sanitization graph
  - Interprocedural data flow graph
    - Build summary functions of each procedure’s data flow and connect
  - Keep only nodes relevant to sanitization
    - String operations
    - Sanitization functions
- Build a test suite of strings
  - For SQL injection and XSS attacks
Take Away

• Web systems are just another operating environment for applications
  ‣ Except they enable anyone in the world to participate
• Claim: They need enforcement to satisfy reference monitor concept with an MPS
  ‣ They don’t satisfy that
• Major problem: the requirement for interaction with untrusted parties
  ‣ Sanitization of inputs in ad hoc
  ‣ Saner provides a testing mechanism for that
Browsing Problems

• Problems in browsers
  ‣ Script Attacks
    • XSS, XSRF, Clickjacking, …
  ‣ Plugins and extensions

• Symantec Report (2007)
  ‣ Internet Explorer: 93 security vulnerabilities; Mozilla: 74, Safari: 29; and Opera: 9
  ‣ 301 reported vulnerabilities in browser plugins
## Plugins and Extensions

![Apple Software Update](image)

### New software is available from Apple

Select the items you want, then click Install.

<table>
<thead>
<tr>
<th>Updates</th>
<th>Name</th>
<th>Version</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>QuickTime</td>
<td>7.5</td>
<td>28.47 MB</td>
</tr>
</tbody>
</table>

**New Software**

<table>
<thead>
<tr>
<th>Name</th>
<th>Version</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Safari</td>
<td>3.1.1</td>
<td>22.56 MB</td>
</tr>
<tr>
<td>iTunes + QuickTime</td>
<td>7.6.2</td>
<td>66.25 MB</td>
</tr>
</tbody>
</table>

QuickTime 7.5 improves application compatibility and addresses security issues.

This release is recommended for all QuickTime 7 users.

For detailed information on the security content of this update, please visit this website: [http://www.info.apple.com/kbnum/n61798](http://www.info.apple.com/kbnum/n61798).

Important Notice to QuickTime 6 Pro Users

Installing QuickTime 7 or later will disable the QuickTime Pro functionality in prior versions of QuickTime, such as QuickTime 6. If you are a QuickTime 6 Pro user and you proceed with this installation, you will need to...

Note: Use of this software is subject to the original Software License Agreement(s) that accompanied the software being updated. A list of Apple SLAs may be found here: [http://www.apple.com/legal/sla/](http://www.apple.com/legal/sla/).
Browser Plugins

- Allow browsers to view additional mime types
  - Loaded as a DLLs or shared objects
  - Communicate with the browser via a well-defined API
- Run in the same address space as the browser
  - Complicates security as they are given unchecked access to browser internals
  - Successful attack on a plugin can lead to a full system compromise
Modify Existing Browsers

- **Script Accent**
  - Domain specific “accents” for scripts and HTML object names
    - “An Analysis of Browser Domain-Isolation Bugs and A Light-Weight Transparent Defence Mechanism” - Shuo Chen et al. - CCS 07

- **Dynamic Tainting**
  - Track flow of sensitive information inside the web browser
    - “Cross Site Scripting Prevention with Dynamic Tainting and Static Analysis” - Phillip Vogt et al. – NDSS 07

- **Different isolation policy**
  - Enforces access using server’s X.509 certificates and keys
    - Locked Same-origin Policies - Chris Karlof et al. - CCS 07
    - Same-origin Mutual Authorization – Oda et al. – CCS 08
OP Principles

- Simple and explicit communication between components
  - Explicit interfaces reduce number of control flow paths
- Strong isolation between browser components
  - Prevents unanticipated and unaudited interactions
- Monitor components to ensure faithful execution
  - Delegate some security logic
- Compatible with existing technologies
  - Goal is to secure web browsing
- **Result**: Reference Monitoring for a Browser
OP Architecture

- Access control and audit log
- User Interface
  - Web Page Instance
  - Web Page Instance
  - Web Page Instance
- Browser Kernel
- Operating System
  - display
- Network
  - network
- Storage
  - file system
Browser Kernel

• Manage subsystems
  ‣ Creates and deletes all processes and subsystems
    • Creates web-page instances on demand
  ‣ Manages communication between subsystems
    • interposes on all message communication (via pipes)

• Manages access control
  ‣ Browser processes tagged with a security context
  ‣ Tag depends on initial messages intercepted

• Maintains detailed security audit log
Web Page Subsystem
## Subsystem Privileges

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>File system access</th>
<th>Network access</th>
</tr>
</thead>
<tbody>
<tr>
<td>UI</td>
<td>allowed</td>
<td>denied</td>
</tr>
<tr>
<td>Web page</td>
<td>denied</td>
<td>denied</td>
</tr>
<tr>
<td>Storage</td>
<td>limited</td>
<td>denied</td>
</tr>
<tr>
<td>Network</td>
<td>denied</td>
<td>allowed</td>
</tr>
<tr>
<td>Browser kernel</td>
<td>allowed</td>
<td>allowed</td>
</tr>
</tbody>
</table>
OP and Plugin Security

• Plugin Security Policies
  ‣ Provider domain policy
    • allows a plugin in a page to use the permissions of the plugin content source
  ‣ Plugin freedom policy
    • allows additional outgoing network accesses for plugins to support peer-to-peer applications
    • can only talk to network and storage subsystems
Formal Verification

- OP browser modeled using Maude (reasoning engine)
  - Both interpreter and language
- Formally specify browser components in terms of invariants over finite space
  - Program invariants include access control policies
  - Visual invariants include UI tampering
    - Address bar spoofing
- Maude checks for states that violate invariants
In terms of permissions on the system, Google Chrome's renderer may run at very low privileges, but there are plugins that run at the same level or even higher than the browser.
VM-based Browser Arch

• Also, may use virtual machines as protection boundaries

• Tahoma
  ‣ Each VM has a browser that may access URLs specified by the server for which the initial page was downloaded

• Flowwolf
  ‣ Each VM has a browser that may access URLs based on the label of the VM
    • URLs are labeled like files by a “URL mapper”
  ‣ Also, the browser tabs implement DLM