CIS 610: Advanced Topics in Systems Security
Web Security

Prof. Kevin Butler

Fall 2011
Projects

• Presentation Thursday
  ‣ Have slides prepared
  ‣ Give background, problem, what you’ve done, ways forward
  ‣ Plan on 10-15 minute presentation plus questions

• Project paper
  ‣ Due December 8
  ‣ Should be the length of a workshop paper
  ‣ Write in LaTeX, either one or two-column article format, should look like the papers we’ve been reading
  ‣ `\usepackage{geometry}`
Network vs. Web Security
What is the web?

- A collection of application-layer services used to distribute content
  - Web content (HTML)
  - Multimedia
  - Email
  - Instant messaging
- Many applications
  - News outlets, entertainment, education, research and technology, …
  - Commercial, consumer and B2B
Web security: the high bits

• The largest distributed system in existence
  ‣ threats are as diverse as applications and users
  ‣ But need to be thought out carefully …

• The stakeholders are …
  ‣ Consumers (users, businesses, agents, …)
  ‣ Providers (web-servers, IM services, …)

• Another way of seeing web security is
  ‣ Securing the web infrastructure such that the integrity, confidentiality, and availability of content and user information is maintained
What the Web Means

• **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

• **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)
The new web page

- Rendered elements from many sources containing *scripts*, *images*, and stylized by *cascading style sheets* (CSS)

- A browser may be compromised by any of these elements
Complexity Means...
Web-server APIs

- Web-servers often provide application extension APIs to which developers can build ...
  - ISSAPI
  - Apache API

- Act as kinds of “kernel modules” for web-server
  - Web-server processes received inputs (URL, fields, etc.)
  - Passes result to custom code (typically, C code)
Application Frameworks

- Application frameworks are software stacks that implement web application
  - Programmer adds domain-specific programming
  - Handle request handling and rendering
  - Quickly implement web apps without dealing the the nasty details of HTTP/HTML

- For example, the Zend framework implements a web application by processing incoming URLs
  - E.g., http://base/module/function
  - Zend accepts returned framework objects and renders them via internal API
  - Modify documents on the fly using AJAX scripts such as JavaScript
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 it 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

- Can we solve all problems with a reference monitor?
  - What is missing here?
- But, we need to enforce an MPS
  - What is missing with regards to that?
- Other problems?
  - Hint: Not exactly satisfying Biba integrity
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

Diagram showing the architecture of a web page instance connected to storage, network, and operating system components such as access control, audit log, user interface, browser kernel, display, and 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

![Diagram of Web Page Subsystem]

- Plugin
- Plugin parsing and rendering
- JavaScript script 1
- JavaScript script 2
- Xvnc
- To UI

Browser Kernel
Web Page Subsystem

- **HTML parsing and rendering Engine**
  - KHTML
- **JavaScript Interpreter**
  - Rhino (Java based)
    - provides strong isolation
- **Plugins are separate OS level processes**
  - provide better isolation
- **Xvnc server renders visual content**
# 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
Browser as OS

• Apply OS principles for security
  ‣ Process abstraction to provide protection and isolation
  ‣ Process management
    • How long to cache the web page instance?
  ‣ Messages to subsystems analogous to syscalls
    • Access to protected resources via the kernel
    • Browser kernel intercepts all messages

• Plugins are like device drivers
  ‣ Hard to secure
  ‣ OP provides protection against buggy plugins
    • Not against malicious ones
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.
Chrome Design

• Different processes for different tabs
  ‣ separate address space
  ‣ No fragmentation of memory unlike legacy browsers
  ‣ isolation offers protection

• Separate thread for JavaScript
Chrome vs OP

• Similarities with OP
  ‣ Browser split into components
    • Each within its own protection domain
  ‣ Both have a main process
    • Browser Kernel, Chrome Process
  ‣ Plugins are executed as separate processes
  ‣ No details about extension handling

• Differences with OP
  ‣ Separate thread for JavaScript VM
  ‣ Separation of browser and tabs
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
Protecting Web Apps

• OP and Chrome can protect the browser from other malicious pages that are loaded

• What about web applications themselves?
  ‣ Still susceptible to clickjacking etc
  ‣ New security features deployed:
    • HTTPOnly cookies: can’t be accessed by JavaScript
    • X-Frame-Options: Can’t be placed in an IFRAME
    • JSON.parse(): allow deserialization of text without execution
• Protections are there but sites aren’t using them
• ZAN: interposes on data in browser and uses context from there to automatically set protections
  ‣ all client-side, must maintain compatibility with web apps
Inferring Protections

- Source code patterns and object characteristics
- Train based on top websites and create classifiers
  - Analysis of phrases related to authentication and use of Gaussian distribution classifier
  - Frame-busting detection code to prevent frames on data through Javascript inspection
  - Javascript string inspection on eval() functions to retrofit function with parse
Introspection

• What are the advantages of this approach?
• What are the potential problems?
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