Application Security through a Hacker’s Eyes

James Walden
Northern Kentucky University
waldenj@nku.edu
Why Do Hackers Target Web Apps?

<table>
<thead>
<tr>
<th>Attack Goal</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stealing Sensitive Information</td>
<td>42%</td>
</tr>
<tr>
<td>Defacement</td>
<td>23%</td>
</tr>
<tr>
<td>Planting Malware</td>
<td>15%</td>
</tr>
<tr>
<td>Unknown</td>
<td>8%</td>
</tr>
<tr>
<td>Deceit</td>
<td>3%</td>
</tr>
<tr>
<td>Blackmail</td>
<td>3%</td>
</tr>
<tr>
<td>Link Spam</td>
<td>3%</td>
</tr>
<tr>
<td>Worm</td>
<td>1%</td>
</tr>
<tr>
<td>Phishing</td>
<td>1%</td>
</tr>
<tr>
<td>Information Warfare</td>
<td>1%</td>
</tr>
</tbody>
</table>
A system’s *attack surface* consists of all of the ways an adversary can enter the system.
Defender’s View of Attack Surface

- firewall
- VPN
- wireless
- web server
Firewalls don’t protect Web Apps

- telnet
- ftp
- HTTP Traffic
- Port 80
- Web Client
- Web Server
- Application
- Application
- Database Server
SSL won’t stop injection attacks, XSS

- telnet
- ftp
- HTTPS Traffic
- Port 443
- Web Client
- Web Server
- Application
- Database Server
- Firewall
Revised View of Attack Surface

- external web server
- external web apps
- firewall
- VPN
- wireless
- database
Intranet Security Assumptions

Since the firewall protects you

- Patches don’t have to be up to date.
- Passwords don’t have to be strong.
- There’s no need to be careful when you code.
- There’s no need to audit your source code.
- There’s no need to run penetration tests.

But do your users have web browsers?
Javascript Malware controls Clients

Diagram showing various components:
- Web Server (Javascript malware)
- HTTP Traffic
- FTP
- Telnet
- Firewall
- Intranet
- Main Server
- Group Server
- Web Client
- Port 80
- Wiki

October 3, 2008
IMI Security Symposium
Port Scanning with JavaScript
SPI Dynamics.com - Security Brief

This is a proof of concept page for port scanning arbitrary IP addresses from JavaScript. Given a range of IP addresses, the scanner will detect if there is a host running at that IP. It will then look for a web server running on port 80 and try to fingerprint what kind of web server it is. Only fingerprinting of Microsoft IIS and Apache are currently supported. If the scanner cannot fingerprint the server it will report it as "Unknown Webserver." This page will not automatically scan your network, will not attack any hosts it discovers, and will not report any information about your network back to SPI Dynamics.

Known issues with the scanner:

<table>
<thead>
<tr>
<th>IP</th>
<th>Host Exists?</th>
<th>Webserver</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.1.100</td>
<td>false</td>
<td>NA</td>
</tr>
<tr>
<td>192.168.1.101</td>
<td>false</td>
<td>NA</td>
</tr>
<tr>
<td>192.168.1.102</td>
<td>false</td>
<td>NA</td>
</tr>
<tr>
<td>192.168.1.103</td>
<td>true</td>
<td>none</td>
</tr>
<tr>
<td>192.168.1.104</td>
<td>false</td>
<td>NA</td>
</tr>
<tr>
<td>192.168.1.105</td>
<td>false</td>
<td>NA</td>
</tr>
<tr>
<td>192.168.1.106</td>
<td>true</td>
<td>none</td>
</tr>
<tr>
<td>192.168.1.107</td>
<td>false</td>
<td>NA</td>
</tr>
<tr>
<td>192.168.1.108</td>
<td>false</td>
<td>NA</td>
</tr>
<tr>
<td>192.168.1.109</td>
<td>true</td>
<td>none</td>
</tr>
<tr>
<td>192.168.1.110</td>
<td>true</td>
<td>Unknown Webserver</td>
</tr>
</tbody>
</table>

Done

October 3, 2008

IMI Security Symposium
Sources of Javascript Malware

1. Evil web site owner inserts in page.
2. Attacker inserts malware into defaced page.
3. Attacker inserts malware into a public comment or forum post (stored XSS.)
4. Attacker creates link that causes web site to echo malware to user (reflected XSS.)
Re-revised View of Attack Surface

- external web server
- external web apps
- internal web apps
- firewall
- VPN
- wireless
- database
- internal web servers
Web Applications

- firewall
- external web server
- VPN
- wireless
- database
- internal web servers
- external web apps
- internal web apps
Web Application Vulnerabilities

Input-based Security Problems
- Injection Flaws
- Insecure Remote File Inclusion
- Unvalidated Input

Authentication and Authorization
- Authentication
- Access Control
- Cross-Site Attacks

Other Bugs
- Error Handling and Information Leakage
- Insecure Storage
- Insecure Communications
SQL Injection

1. App sends form to user.
2. Attacker submits form with SQL exploit data.
3. Application builds string with exploit data.
4. Application sends SQL query to DB.
5. DB executes query, including exploit, sends data back to application.
6. Application returns data to user.
Cross-Site Scripting

1. Login
2. Cookie
3. XSS Attack
4. User clicks on XSS link.
5. XSS URL
6. Page with injected code.
7. Browser runs injected code.
8. Attacker hijacks user session.

Evil site saves ID.
Web Application Attack Surface

- form inputs
- HTTP headers
- cookies
- URLs
Traditional Web Applications

User waits

HTTP Request (form submission)

HTTP Response (new web page)

User waits

HTTP Request (form submission)

HTTP Response (new web page)

Server processing

User interaction
AJAX

Asynchronous Javascript and XML

- User interacts with client-side Javascript.
- Javascript makes asynchronous requests to server for data.
- Continues to allow user to interact with application.
- Updates when receives encoded data from server.
AJAX Applications

User interaction → partial update → HTTP request (asynchronous) → HTTP Response (data) → partial update → HTTP request (asynchronous) → HTTP Response (data) → partial update → HTTP request (asynchronous) → HTTP Response (data) → partial update → HTTP request (asynchronous) → HTTP Response (data) → partial update

Client-side Code

User interaction

User interaction

User interaction

Server processing

Server processing

Server processing

October 3, 2008

IMI Security Symposium

NKU NORTHERN KENTUCKY UNIVERSITY
Example Client-side Code

```javascript
var auth = checkPassword(user, pass);
if (auth == false) {
    alert('Authentication failed.');
    return;
}
var itemPrice = getPrice(itemID);
debitAccount(user, itemPrice);
downloadItem(itemID);
```
AJAX Application Attack Surface

- Form inputs
- Cookies
- Client-side code
- Client-side state
- HTTP headers
- URLs
- Server API
- Client-side data transforms
Drilling Down: Mapping the Application

1. Visible Content
   - Spider the site.
   - Browse site while using intercepting proxy.

2. Hidden Content
   1. Unlinked sections of site.
   2. Backup copies of live files.
   3. Configuration and include files.
   4. Source code.
   5. Log files.
Entry Points

For each resource found, identify inputs:

- Additional path parameters
- Query string
- POST parameters
- Cookies
- HTTP headers
Application Feature Vulnerability Map

Database interaction → SQL injection.
Displays user-supplied data → Cross-site scripting.
Error messages → Information leakage.
File upload/download → Path traversal.
Login → Authentication, session management, access control flaws.
Code Auditing

Why?
- Find vulnerabilities faster than testing.
- Find different vulnerabilities than testing.

What?
- Identify modules of high business risk.
- Use static analysis to find common vulnerabilities.
- Manually review code + static analysis results.

Who?
- Developers, security team, outside auditors.

When?
- On a regular basis, at least once before release.
Static Analysis

Automated assistance for code auditing

Speed: review code faster than humans can

Accuracy: hundreds of secure coding rules

Tools

- Coverity
- FindBugs
- Fortify
- Klocwork
- Ounce Labs
Fuzz testing consists of

- Sending unexpected input.
- Monitoring for exceptions.
Monitoring for Exceptions

Application mapping

- Response code
- Response size
- Presence of string “not authorized”

Password guessing

- Response code
- Response size
- Presence of string “login incorrect”
Prevention Guidelines

1. Use a standard, secure authentication scheme.
2. Check access control on every transaction.
3. Avoid using interpreters where possible.
4. Don’t leak sensitive information in error pages.
5. Encrypt sensitive data in transit and on disk.
6. Encode user data in output.
7. Don’t trust any data from the client.
8. Validate all input.
Input Validation

**Blacklist:** reject known bad input
- Reject input matching list of bad strings/patterns.
- Accept all other input.
- Vulnerable to encoding attacks.

**Whitelist:** accept known good input
- Accept input matching list of good strings/patterns.
- Reject all other input.
- Highly effective, but not always feasible.
1. Visible Content
   • Linked URLs.
   • Authenticated URLs.
2. Hidden Content
   1. Unlinked sections of site.
   2. Backup copies of live files.
   3. Configuration/include files.
   4. Source code.
   5. Log files.

A site’s attack surface is nearly fractal.