Attack Surface of Web Applications

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>

Pie chart showing the distribution of attack goals:
- Stealing Sensitive Information (42%)
- Defacement (23%)
- Planting Malware (15%)
- Unknown (8%)
- Deceit (3%)
- Blackmail (3%)
- Link Spam (3%)
- Other (11%)
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
## History of Web Security

<table>
<thead>
<tr>
<th>Year</th>
<th>Technology</th>
<th>Security</th>
</tr>
</thead>
<tbody>
<tr>
<td>1993</td>
<td>CGI</td>
<td>Firewalls, SSL</td>
</tr>
<tr>
<td>1995</td>
<td>PHP, Javascript</td>
<td>Firewalls, SSL</td>
</tr>
<tr>
<td>1997</td>
<td>ASP, JSP</td>
<td>Firewalls, SSL</td>
</tr>
<tr>
<td>2000</td>
<td>REST, SOA</td>
<td>Firewalls, SSL</td>
</tr>
<tr>
<td>2006</td>
<td>AJAX</td>
<td>Firewalls, SSL</td>
</tr>
</tbody>
</table>
Firewalls don’t protect Web Apps

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

- telnet
- ftp
- Web Server (Javascript malware)
- HTTP Traffic
- Firewall
- Main Server
- Group Server
- Intranet
- Port 80
- Web Client
- Wiki
- Telnet
- FTP
- HTTP

November 13, 2008
Ohio Information Security Forum
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 To Start:</th>
<th>IP To End:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<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>
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

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

external web server
firewall
VPN
wireless
internal web servers
database
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.

Web Server

User ‘ or 1=1--

Pass

Attacker

DB Server

Firewall
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.

Attacker

User

Web Server

Evil site saves ID.
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.
Web Application Attack Surface

- Form inputs
- HTTP headers
- Cookies
- URLs
Traditional Web Applications

HTTP Request (form submission)

HTTP Response (new web page)

Server processing

User waits

User interaction

User waits

HTTP Request (form submission)

HTTP Response (new web page)

Server processing
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

Client-side Code

Server processing
Architecture Differences

**Traditional**

- Application on server.
- Entire form sent to server.
  - User fills in input items.
  - Clicks on submit.
- Server returns new page.
  - Presentation + Data.

**AJAX**

- App on client and server.
- JavaScript receives user input, issues function calls to server when needed.
  - Get map tile.
  - Save location data.
- Server returns individual data items.
- JavaScript incorporates data items into existing page.
Example Client-side Code

var auth = checkPassword(user, pass);
if (auth == false) {
    alert('Authentication failed. ');
    return;
}
var itemPrice = getPrice(itemID);
debitAccount(user, itemPrice);
downloadItem(itemID);
var json = getItem()
// json = "[ ‘Toshiba’, 499, ‘LCD TV’]"

var item = eval(json)
// item[0] = ‘Toshiba’
// item[1] = 499
// item[2] = ‘LCD TV’
JSON Injection

Evil input: ‘];alert(‘XSS’);//

var json = getItem()
// json = “[ ‘Toshiba’, 499, ‘’ ];alert(‘XSS’);//”

var item = eval(json)
// Alert box with ‘XSS’ appears.
// Use json2.js validation library to prevent.
Client-Side State

Storage Technologies

- Cookies
- DOM Storage (HTML5)
- Flash LSOs
- UserData (IE)

Client-Side Storage Issues

- User can always modify client-side data.
- Cross-Domain Attacks (between subdomains).
- Cross-directory Attacks.
- Cross-port Attacks.
AJAX Application Attack Surface

- form inputs
- cookies
- client-side code
- client-side state
- HTTP headers
- URLs
- server API
- client-side data transforms
A site’s attack surface is nearly fractal.