Web Application - User’s View
Web Application – Hacker’s View

- Cross-site scripting
- Session Hijacking
- Parameter Manipulation
- CSRF
- Password Guessing
- Account Enumeration
- SQL Injection
- Denial of Service
Which web programming languages and industries are most secure?

Data collected from January 1, 2006 to March 25, 2010
WhiteHat Security

- 300+ enterprise customers
  - Start-ups to Fortune 500
- Flagship offering “WhiteHat Sentinel Service”
  - Thousands of assessments performed annually
- Recognized leader in website security
  - Jeremiah Grossman CTO
  - Quoted thousands of times by the mainstream press
WhiteHat Sentinel

Complete Website Vulnerability Management
*Customer Controlled & Expert Managed*

- **Unique SaaS-based solution** – Highly scalable delivery of service at a fixed cost
- **Production Safe** – No Performance Impact
- **Full Coverage** – On-going testing for business logic flaws and technical vulnerabilities – uses WASC 24 classes of attacks as reference point
- **Unlimited Assessments** – Anytime websites change
- **Eliminates False Positives** – Security Operations Team verifies all vulnerabilities
- **Continuous Improvement & Refinement** – Ongoing updates and enhancements to underlying technology and processes
# Website Classes of Attacks

## Technical: Automation Can Identify

### Command Execution
- Buffer Overflow
- Format String Attack
- LDAP Injection
- OS Commanding
- SQL Injection
- SSI Injection
- XPath Injection

### Information Disclosure
- Directory Indexing
- Information Leakage
- Path Traversal
- Predictable Resource Location

### Client-Side
- Content Spoofing
- Cross-site Scripting
- HTTP Response Splitting*

## Business Logic: Humans Required

### Authentication
- Brute Force
- Insufficient Authentication
- Weak Password Recovery Validation
- CSRF*

### Authorization
- Credential/Session Prediction
- Insufficient Authorization
- Insufficient Session Expiration
- Session Fixation

### Logical Attacks
- Abuse of Functionality
- Denial of Service
- Insufficient Anti-automation
- Insufficient Process Validation
Attacker Profile/Targeting

Random Opportunistic
- Fully automated scripts
- Unauthenticated scans
- Targets chosen indiscriminately

Directed Opportunistic
- Commercial and Open Source Tools
- Authentication scans
- Multi-step processes (forms)

Fully Targeted (APT?)
- Customize their own tools
- Focused on business logic
- Profit or goal driven ($$$)
What’s a website?

Websites, which may be a collection of multiple web servers and hostnames, often utilize more than one programming language or framework. As such, a single website may contain vulnerabilities with multiple different extensions.
Data Overview

- **1,659 total websites**
- **24,286 verified custom web application vulnerabilities**
- Data collected from January 1, 2006 to March 25, 2010
- Vast majority of websites assessed for vulnerabilities weekly
- Vulnerabilities classified according to WASC Threat Classification, the most comprehensive listing of Web application vulnerabilities
- Vulnerability severity naming convention aligns with PCI-DSS
- Contrasted and compared ASP Classic, .NET, Cold Fusion, Struts, Java Server Pages, PHP, and Perl.

<table>
<thead>
<tr>
<th></th>
<th>ASP</th>
<th>ASPX</th>
<th>CFM</th>
<th>DO</th>
<th>JSP</th>
<th>PHP</th>
<th>PL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average # of inputs (attack surface) per website</td>
<td>470</td>
<td>484</td>
<td>457</td>
<td>569</td>
<td>919</td>
<td>352</td>
<td>588</td>
</tr>
<tr>
<td>Average ratio of vulnerability count / number of inputs</td>
<td>8.7%</td>
<td>6.2%</td>
<td>8.4%</td>
<td>6.3%</td>
<td>9.8%</td>
<td>8.1%</td>
<td>11.6%</td>
</tr>
</tbody>
</table>
Key Findings

• Languages / frameworks do not have identical security postures when deployed in the field -- they have moderately different vulnerabilities, with different frequency of occurrence, which are fixed in different amounts of time.

• Perl (PL) had the highest average number of vulnerabilities found historically by a wide margin, at 44.8 per website and also the largest number currently at 11.8. Struts (DO) edged out Microsoft’s .NET (ASPX) for the lowest average number of currently open vulnerabilities per website at 5.5 versus 6.2.

• Cold Fusion (CFM) had the second highest average number of vulnerabilities per website historically at 34.4, but the lowest likelihood of having a single serious* unresolved vulnerability if currently managed under WhiteHat Sentinel (54%).

• Perl (PL), Cold Fusion (CFM), JSP, and PHP websites were the most likely to have at least one serious* vulnerability, at roughly 80% of the time.

• 37% of Cold Fusion (CFM) websites had SQL Injection vulnerabilities, the highest of all measured, while Struts (DO) and JSP had the lowest with 14% and 15%.

• PHP and Perl websites were among the worst in average vulnerability counts, but had fastest average Cross-Site Scripting remediation times -- 52 and 53 days respectively. At same time Microsoft’s .NET (ASPX) performed among the best in vulnerability count averages, but placed dead last for remediation times at 87 days.
## Key Findings - 1,659

<table>
<thead>
<tr>
<th></th>
<th>ASP</th>
<th>ASPX</th>
<th>CFM</th>
<th>DO</th>
<th>JSP</th>
<th>PHP</th>
<th>PL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Websites having had at least one serious* vulnerability</td>
<td>74%</td>
<td>73%</td>
<td>86%</td>
<td>77%</td>
<td>80%</td>
<td>80%</td>
<td>88%</td>
</tr>
<tr>
<td>Websites currently with at least one serious* vulnerability</td>
<td>57%</td>
<td>58%</td>
<td>54%</td>
<td>56%</td>
<td>59%</td>
<td>63%</td>
<td>75%</td>
</tr>
<tr>
<td>Avg. # of serious* vulnerabilities per website during the WhiteHat Sentinel assessment lifetime</td>
<td>25</td>
<td>18.7</td>
<td>34.3</td>
<td>19.9</td>
<td>25.8</td>
<td>26.6</td>
<td>44.8</td>
</tr>
<tr>
<td>Avg. # of serious* severity unresolved vulnerabilities per website</td>
<td>8.9</td>
<td>6.2</td>
<td>8.6</td>
<td>5.5</td>
<td>9.6</td>
<td>8.3</td>
<td>11.8</td>
</tr>
</tbody>
</table>
Top Ten Classes of Attack

Percentage likelihood of a website having a vulnerability by class

- Cross-Site Scripting
- Information Leakage
- Content Spoofing
- Insufficient Authorization
- SQL Injection
- Predictable Resource Location
- Cross-Site Request Forgery
- Session Fixation
- HTTP Response Splitting
- Abuse of Functionality
- Insufficient Authentication
- Directory Traversal
- Directory Indexing

ASP, ASPX, CFM, DO, JSP, PHP, PL
<table>
<thead>
<tr>
<th>Technology</th>
<th>Cross-Site Scripting</th>
<th>Information Leakage</th>
<th>Content Spoofing</th>
<th>Insufficient Authorization</th>
<th>SQL Injection</th>
<th>Predictable Resource Location</th>
<th>Cross-Site Request Forgery</th>
<th>Session Fixation</th>
<th>HTTP Response Splitting</th>
<th>Abuse of Functionality</th>
<th>Insufficient Authentication</th>
<th>Directory Traversal</th>
<th>Directory Indexing</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASPX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CFM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JSP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PHP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AVG.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Resolution Rates by Severity

<table>
<thead>
<tr>
<th>Class of Attack</th>
<th>Severity</th>
<th>ASP</th>
<th>ASPX</th>
<th>CFM</th>
<th>DO</th>
<th>JSP</th>
<th>PHP</th>
<th>PL</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL Injection</td>
<td>Urgent</td>
<td>70%</td>
<td>72%</td>
<td>66%</td>
<td>79%</td>
<td>58%</td>
<td>70%</td>
<td>71%</td>
</tr>
<tr>
<td>Insufficient Authorization</td>
<td>Urgent</td>
<td>21%</td>
<td>45%</td>
<td>46%</td>
<td>20%</td>
<td>25%</td>
<td>18%</td>
<td>10%</td>
</tr>
<tr>
<td>Directory Traversal</td>
<td>Urgent</td>
<td>43%</td>
<td>20%</td>
<td>67%</td>
<td>0%</td>
<td>33%</td>
<td>32%</td>
<td>16%</td>
</tr>
<tr>
<td>Cross Site Scripting</td>
<td>Urgent</td>
<td>100%</td>
<td>0%</td>
<td>100%</td>
<td>0%</td>
<td>0%</td>
<td>50%</td>
<td>0%</td>
</tr>
<tr>
<td>Cross-Site Scripting</td>
<td>Critical</td>
<td>51%</td>
<td>57%</td>
<td>50%</td>
<td>51%</td>
<td>52%</td>
<td>66%</td>
<td>54%</td>
</tr>
<tr>
<td>Cross-Site Request Forgery</td>
<td>Critical</td>
<td>18%</td>
<td>34%</td>
<td>17%</td>
<td>27%</td>
<td>39%</td>
<td>57%</td>
<td>27%</td>
</tr>
<tr>
<td>Session Fixation</td>
<td>Critical</td>
<td>19%</td>
<td>18%</td>
<td>0%</td>
<td>36%</td>
<td>50%</td>
<td>50%</td>
<td>100%</td>
</tr>
<tr>
<td>Abuse of Functionality</td>
<td>Critical</td>
<td>76%</td>
<td>23%</td>
<td>82%</td>
<td>38%</td>
<td>57%</td>
<td>59%</td>
<td>97%</td>
</tr>
<tr>
<td>Insufficient Authentication</td>
<td>Critical</td>
<td>55%</td>
<td>37%</td>
<td>0%</td>
<td>33%</td>
<td>71%</td>
<td>0%</td>
<td>100%</td>
</tr>
<tr>
<td>Information Leakage</td>
<td>High</td>
<td>32%</td>
<td>34%</td>
<td>57%</td>
<td>49%</td>
<td>45%</td>
<td>39%</td>
<td>29%</td>
</tr>
<tr>
<td>Content Spoofing</td>
<td>High</td>
<td>31%</td>
<td>30%</td>
<td>43%</td>
<td>37%</td>
<td>44%</td>
<td>46%</td>
<td>69%</td>
</tr>
<tr>
<td>Predictable Resource Loc.</td>
<td>High</td>
<td>29%</td>
<td>64%</td>
<td>85%</td>
<td>64%</td>
<td>53%</td>
<td>56%</td>
<td>29%</td>
</tr>
<tr>
<td>HTTP Response Splitting</td>
<td>High</td>
<td>28%</td>
<td>24%</td>
<td>33%</td>
<td>10%</td>
<td>36%</td>
<td>42%</td>
<td>35%</td>
</tr>
<tr>
<td>Directory Indexing</td>
<td>High</td>
<td>33%</td>
<td>56%</td>
<td>40%</td>
<td>25%</td>
<td>27%</td>
<td>33%</td>
<td>18%</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td></td>
<td>65%</td>
<td>67%</td>
<td>75%</td>
<td>72%</td>
<td>63%</td>
<td>69%</td>
<td>74%</td>
</tr>
</tbody>
</table>
Technology in Use

Financial Services
- ASP: 27%
- ASPX: 4%
- CFM: 17%
- DO: 16%
- JSP: 19%
- PHP: 17%
- PL: 4%

Retail
- ASP: 24%
- ASPX: 7%
- CFM: 33%
- DO: 8%
- JSP: 28%
- PHP: 14%
- PL: 2%

IT
- ASP: 20%
- ASPX: 10%
- CFM: 14%
- DO: 27%
- JSP: 26%
- PHP: 16%
- PL: 1%

Healthcare
- ASP: 14%
- ASPX: 1%
- CFM: 15%
- DO: 16%
- JSP: 26%
- PHP: 22%
- PL: 3%

Insurance
- ASP: 35%
- ASPX: 2%
- CFM: 53%
- DO: 50%
- JSP: 22%
- PHP: 16%
- PL: 2%

Pharma
- ASP: 32%
- ASPX: 6%
- CFM: 40%
- DO: 14%
- JSP: 32%
- PHP: 16%
- PL: 11%

Social Networking
- ASP: 13%
- ASPX: 3%
- CFM: 18%
- DO: 17%
- JSP: 32%
- PHP: 9%
- PL: 3%

Telecom
- ASP: 14%
- ASPX: 3%
- CFM: 24%
- DO: 24%
- JSP: 17%
- PHP: 11%
- PL: 7%

Education
- ASP: 11%
- ASPX: 2%
- CFM: 9%
- DO: 24%
- JSP: 17%
- PHP: 11%
- PL: 4%
Top Five by Technology in Use & Industry
Final Thoughts

- **Technically speaking, one Web programming language / development framework can be made basically just as secure (or not) as any other.**

- **You can't secure what you don't know you own** – Inventory your Web applications to gain visibility into what data is at risk and where attackers can exploit the money or data transacted.

- **Assign a champion** – Designate someone who can own and drive data security and is strongly empowered to direct numerous teams for support. Without accountability, security, and compliance, will suffer.

- **Don't wait for developers to take charge of security** – Deploy shielding technologies to mitigate the risk of vulnerable Web applications.

- **Shift budget from infrastructure to Web application security** – With the proper resource allocation, corporate risk can be dramatically reduced.

- **Community** - get personally involved in supporting OWASP, leverage the projects that can help you **Open Software Assurance Maturity Model (Open SAMM**
Thank You

Tom Brennan
tom.brennan@whitehatsec.com

Full Report Available

https://www.whitehatsec.com/home/assets/WPstats_spring10_9thFR.pdf