Creating Stronger, Safer, Web Facing Code

JPL IT Security
Mary Rivera
June 17, 2011
Agenda

- Evolving Threats
  - Operating System
  - Application
  - User Generated Content
- JPL’s Application Security Program
- Securing Web applications
  - Common vulnerabilities
  - Prevention techniques
  - Security testing tools
- Summary
10 Years ago…

- **Operating System Attacks**
  - Direct attacks
  - Buffer Overflow
  - Denial of Service

- **Mitigation**
  - System administrators got quicker at patch management
  - Vendors started releasing fixes quicker
  - Firewalls had better protection
3 years ago...

- **Application Threats**
  - Hackers moved up a level from OS to Application
  - Directed attacks against
    - SSH
    - Apache web servers
    - SQL database servers

- **Mitigation**
  - SA’s got quicker at patch management
  - Vendors started releasing fixes quicker
  - Firewalls had better protection
  - IT Sec started scanning applications not just operating systems
Today...

- **User Content Threats**
  - Hackers moved up one more level from application itself to content within the application
  - Attacking User Content
  - User generated code
    - SQL injection, Cross Site Scripting
  - Neither SA’s nor vendors know how to fix user code

- **Mitigation**
  - Help user become security aware
  - Security in the Lifecycle
  - Scan code
Half of the Security Incidents involved Applications

- Problem:
  - In 2008, 49% of the JPL security incidents involved application vulnerabilities (shown in red).
Agenda

- Evolving Threats
  - Operating System
  - Application
  - User Generated Content
- JPL’s Application Security Program <<
- Securing Web Applications
  - Common vulnerabilities
  - Prevention techniques
  - Security testing tools
- Summary
JPL’s Application Security Program

- Security Guidelines
- Training & Awareness
- Security in Lifecycle
- Scanning Tools
- App Security Registry

- Security Guidelines
JPL Application Security Program

- Security Guidelines
  - Programming languages
    • PERL, ColdFusion, Java
  - Security checklists

- Training & Awareness
  - Developer training courses
    • Web Application Security
    • Online AppSec Training tutorials
  - Quarterly Application Security Newsletter
Application Security Program

- Security in Lifecycle
  - IT Security checklist
  - Security process

- Security Scanning tools
  - AppScan
    - Web application testing
    - Static source code analysis
Application Security Program

- Application Security Registry
  - Inventory of applications
  - Technical information about applications for security purposes
  - Identifies responsible personnel for each application in the inventory
Agenda

- Evolving Threats
  - Operating System
  - Application
  - User Generated Content

- JPL’s Application Security Program

- Securing Web Applications <<
  - Common vulnerabilities
  - Prevention techniques
  - Security testing tools

- Summary
Common Web Vulnerabilities

- Open Web Application Security Project (OWASP) Top 10 list
  - Identifies the most common vulnerabilities

- Top Vulnerability categories
  - Injection flaws
  - Cross site scripting flaws
Injection Flaws

- Allows attackers to execute malicious code through a web application or other system
  - Access to OS via shell commands
  - Access to backend Database through SQL
    - SQL Injection
Injection Flaws

- SQL Injection
  - Application receives input from a user
  - Input is sent as part of a database query
  - Allows malicious users to execute commands on the database

- Occurs due to:
  - Improper input validation
  - Over privileged database logins
Potential Effects of SQL Injection

- Complete access to database
- Bypass authentication controls
- Potential command line access from database machine
SQL Injection Example

- **Vulnerable Query:**
  - SELECT user FROM Users where loginName = ' $User' and LoginPassword = ' $Password'

- **Injected Query:**
  - Attacker Input: $Password = ' OR 1 = 1 --
  - SELECT user FROM Users where loginName = ' jsmith' and LoginPassword = 'Demo1234 ' OR 1 = 1 --
SQL Injection Example

SELECT true FROM users
WHERE username = 'jsmith' AND password = 'Demo1234'

Injected Query:
Attacker’s extra input to password: ‘OR 1 = 1

--
SQL Injection Example

Application vulnerable to SQL injection
Preventing SQL Injection

- Use parameterized queries
- Use input validation
- Use low privileged accounts
- Limit error messages
- OWASP SQL Injection Prevention Cheat Sheet
Testing Tools for SQL Injection

- SQL Inject Me
  - Firefox add on

- Other tools
  - Absinthe
  - Paros
Testing tool for SQL Injection

- Absinthe
Cross-site scripting (XSS)

- Tricks the browser into executing code
  - JavaScript, VBScript, ActiveX, HTML, or Flash can be injected into a vulnerable application
- Application receives input from the user
- Input is returned back to the user without being sanitized
Potential Effects of XSS

- Redirection
- Web page contents modified
- Scripting commands
- Cookies compromised
XSS Example

Input String: `<SCRIPT>alert(“XSS”)<SCRIPT>`
Preventing XSS

- Filter meta characters, scripting, object tags
  - `<script>` and `<object>`
- Use encoding
  - HTML encode or URL encode
- Detailed information on XSS prevention
  - OWASP XSS Prevention Cheat Sheet
Testing Tool for XSS

- Paros Proxy
Summary

- Changes in threats require keeping pace with changes
- Secure web applications by
  - Fixing common web vulnerabilities
  - Using prevention techniques
  - Using security testing tools
Resources

- Open Source Web Application Security Project (OWASP)
  - [http://www.owasp.org](http://www.owasp.org)

- SQL Injection Cheat Sheet

- XSS Cheat Sheet
  - [https://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_Cheat_Sheet](https://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_Cheat_Sheet)

- Tools
  - Paros
    - [http://www.parosproxy.org/download.shtml](http://www.parosproxy.org/download.shtml)
  - SQL Injectme
  - Absinthe
    - [http://www.0x90.org/releases/absinthe/](http://www.0x90.org/releases/absinthe/)