

# 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...

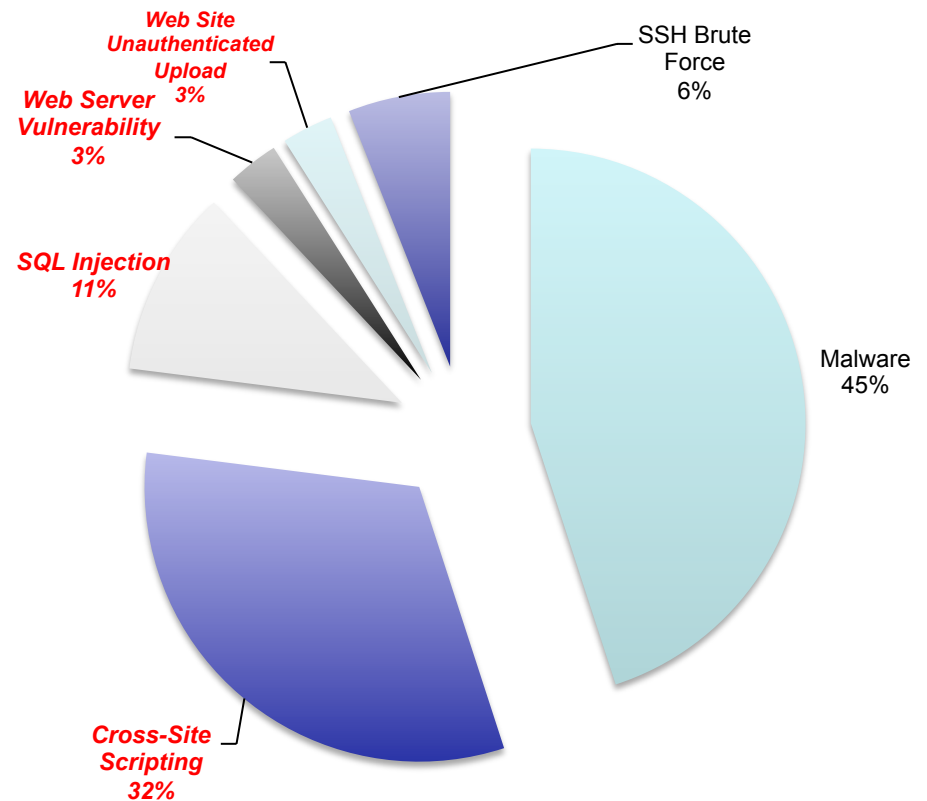
- Application Threats
  - Hackers moved up a level from OS to Application
  - Directed attacks against
    - SSH
    - Apache web servers
    - SQL database servers
- Mitigation
  - SA's got quicker at patch management
  - Vendors started releasing fixes quicker
  - Firewalls had better protection
  - 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

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

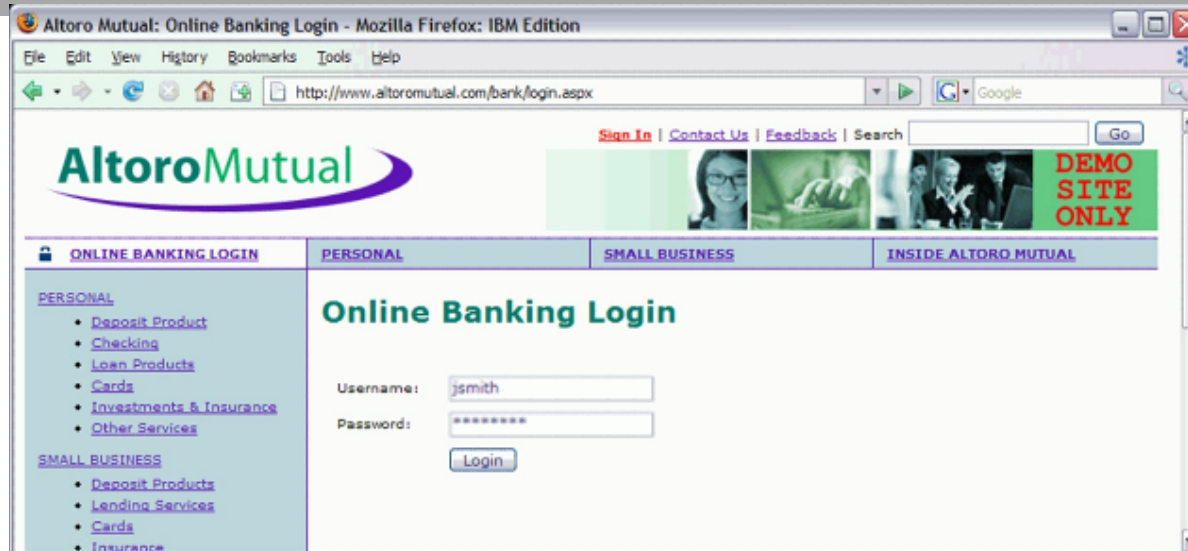
- SELECT user FROM Users where loginName = ' \$User' and LoginPassword = ' \$Password'

- 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

The screenshot shows a web browser window titled "Altoro Mutual: Online Banking Home". The page features the Altoro Mutual logo and a navigation menu with links for "Sign Off", "Contact Us", "Feedback", and a search bar. Below the navigation, there are four tabs: "MY ACCOUNT", "PERSONAL", "SMALL BUSINESS", and "INSIDE ALTORO MUTUAL". The "PERSONAL" tab is active, displaying a personalized greeting: "Hello Admin User" and "Welcome to Altoro Mutual Online." Below this, there is a "View Account Details:" label followed by a dropdown menu and a "GO" button. The footer contains links for "Privacy Policy" and "Security Statement", and a copyright notice for "© 2011 Altoro Mutual, Inc." A red dashed box highlights a disclaimer at the bottom of the page: "The Altoro Mutual website is published by Watchfire, Inc. for the sole purpose of demonstrating the effectiveness of Watchfire products in detecting web application vulnerabilities and website defects. This site is not a real banking site. Similarities, if any, to third party products and/or websites are purely coincidental. This site is provided 'as is' without warranty of any kind, either".

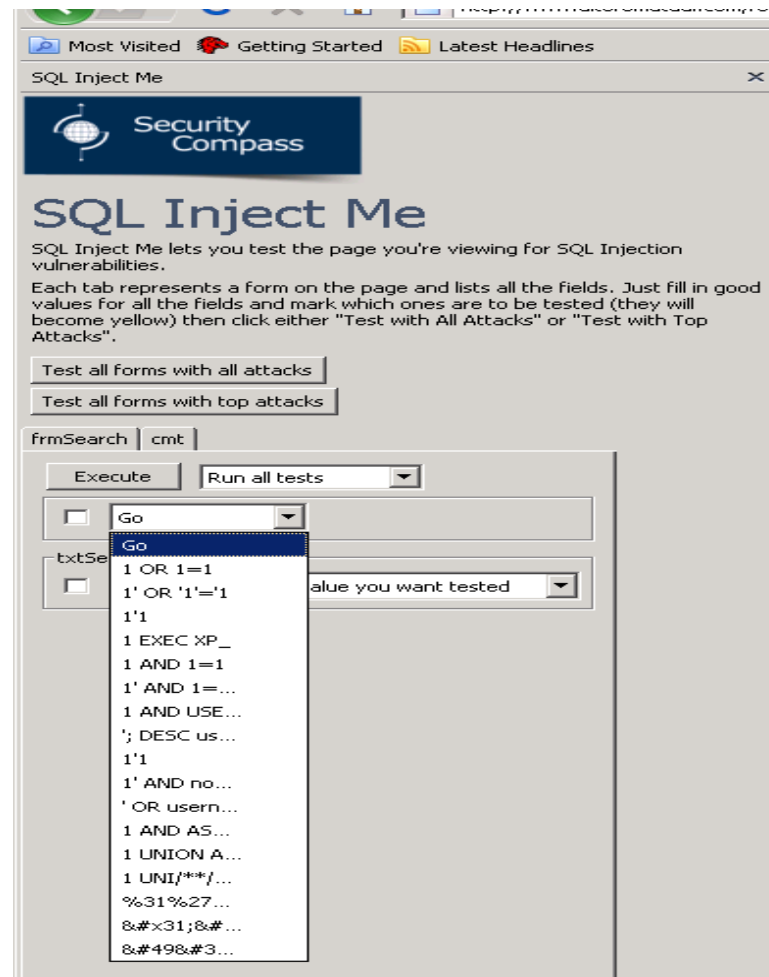
Application vulnerable to SQL injection

# Preventing SQL Injection

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

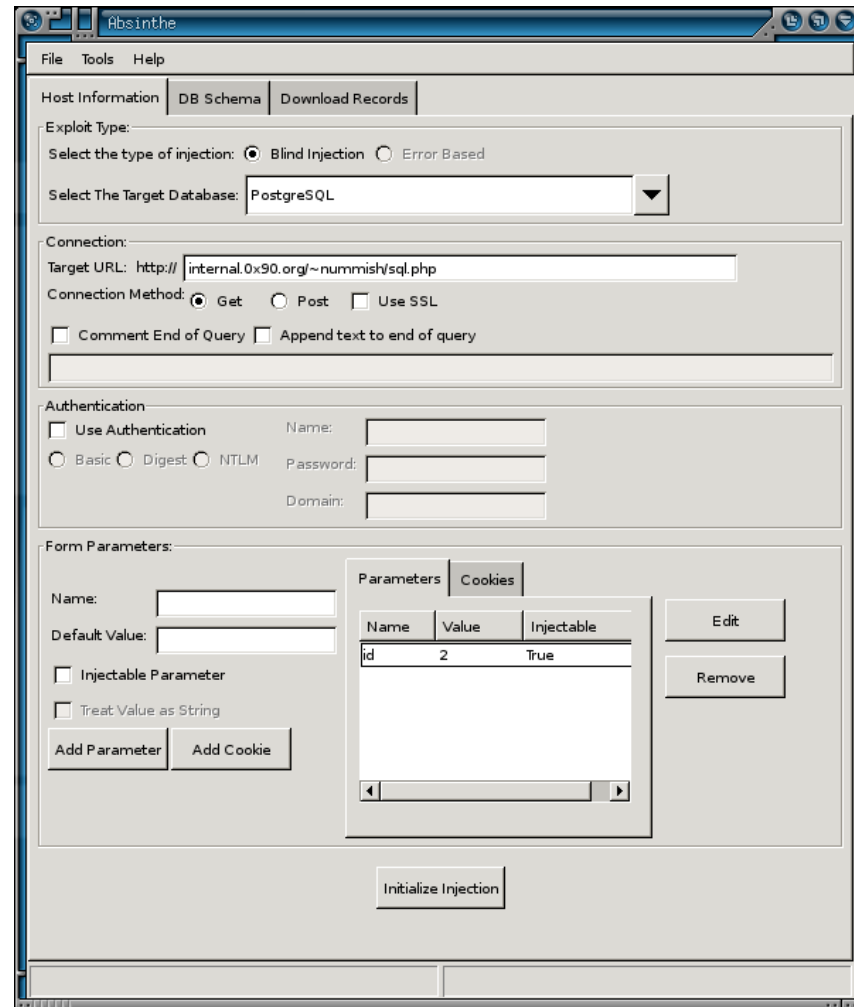
## Testing Tools for SQL Injection

- 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

The screenshot shows a web browser window titled "Altoro Mutual: Search Results". The page features the Altoro Mutual logo, navigation links for "Sign In", "Contact Us", "Feedback", and a search bar. Below the header, there are tabs for "PERSONAL", "SMALL BUSINESS", and "INSIDE ALTORO MUTUAL". The "PERSONAL" tab is selected, and the "Search Results" section displays "No results were found". A JavaScript alert box is overlaid on the page, displaying the message "XSS" with a yellow warning icon and an "OK" button. The alert box title is "The page at http://altoromutual.com says:".

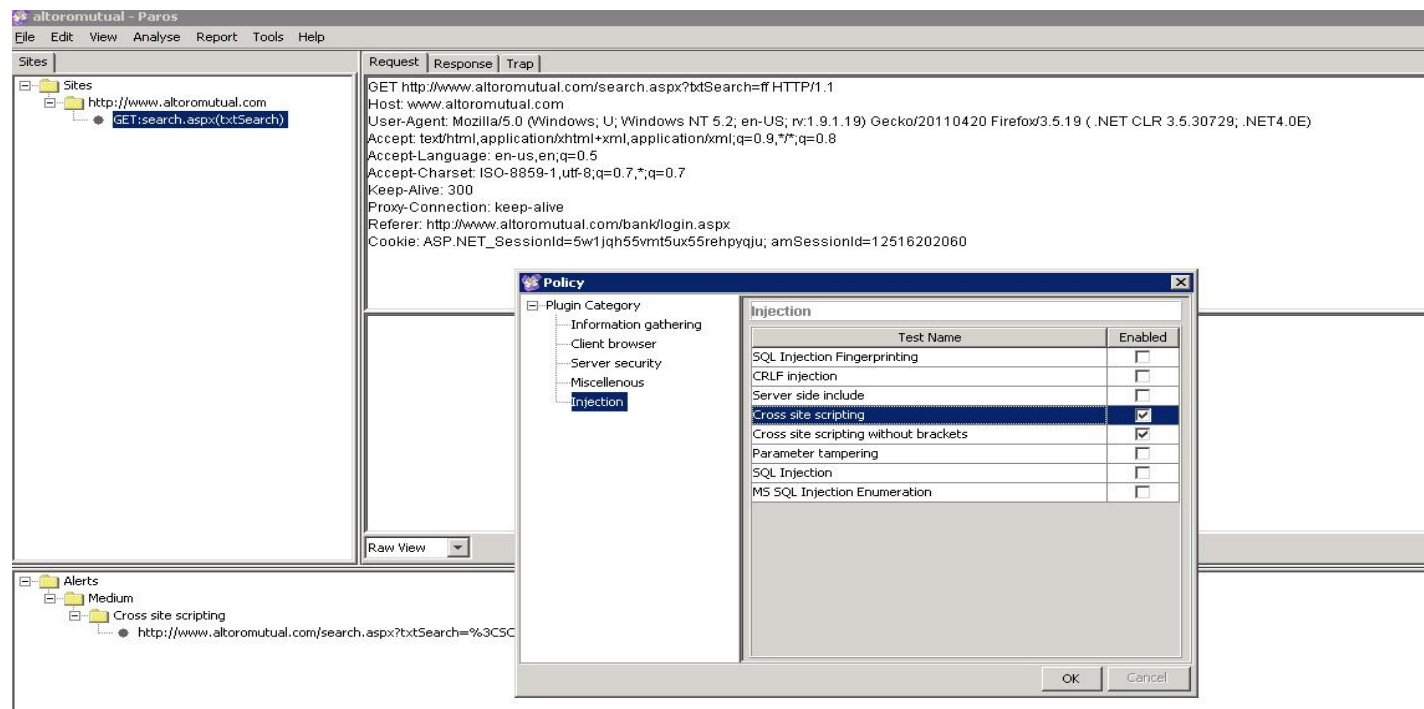
Input String: `<SCRIPT>alert("XSS")<SCRIPT>`

# 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>
- SQL Injection Cheat Sheet
  - [https://www.owasp.org/index.php/SQL\\_Injection\\_Prevention\\_Cheat\\_Sheet](https://www.owasp.org/index.php/SQL_Injection_Prevention_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>
  - SQL Injectme
    - <https://addons.mozilla.org/en-US/firefox/addon/sql-inject-me/>
  - Absinthe
    - <http://www.0x90.org/releases/absinthe/>

# QUESTIONS?