Module I

Introduction to Web Security

- Web Application Technologies
- Web Functionality
- Encoding Schemes
- Mapping the Application
    - Enumerating the Content and Functionality
    - Analyzing the Application
- Bypassing Client-Side Controls
- Transmitting Data via the Client
- Capturing User Data
- Handling Client-Side Data Securely
- Input Validation - Blacklist Validation, Whitelist Validation, The Defense in-Depth Approach
- Attack Surface Reduction, Rules of Thumb
- Classifying and Prioritizing Threats.

Web Application Technologies

The hypertext transfer protocol (HTTP) is the core communications protocol used to access the World Wide Web. It is a simple protocol, originally developed for static webpages and now is used to support the complex applications.

HTTP uses the TCP protocol as its transport mechanism. Each exchange of request and response uses a different TCP connection.

HTTP requests

All HTTP messages (requests and responses) consist of one or more headers, each on a separate line, followed by a mandatory blank line, followed by an optional message body. A typical HTTP request is as follows:

```
GET /books/search.asp?q=wahh HTTP/1.1
Accept: image/gif, image/xxbitmap, image/jpeg, image/pjpeg,
application/xshockwaveflash, application/vnd.msexcel,
application/vnd.mspowerpoint, application/msword, */*
Referer: http://wahh-app.com/books/default.asp
Accept-Language: en-gb,en-us;q=0.5
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)
Host: wahh-app.com
Cookie: lang=en; JSESSIONID=0000tI8rk7joMx44S2Uu85nSWc_:vsnlc502
```

The first line of every HTTP request consists of three items, separated by spaces:

- A verb indicating the HTTP method. The most commonly used method is GET, whose function is to retrieve a resource from the web server. GET requests do not have a message body, so there is no further data following the blank line after the message headers.
- The requested URL. The URL functions as a name for the resource being requested, together with an optional query string containing parameters that the client is passing to that resource. The query string is indicated by the ? character in the URL, and in the example there is a single parameter with the name q and the value wahh.
- HTTP version being used in the request. The only HTTP versions in common use on the Internet are 1.0 and 1.1.

Some other points of interest in the example request are:

- The Referer header is used to indicate the URL from which the request originated (for example, because the user clicked a link on that page).
- The User-Agent header is used to provide information about the browser or other client software that generated the request.
- The Host header is used to specify the hostname that appeared in the full URL being accessed. This is necessary when multiple web sites are hosted on the same server, because the URL sent in the first line of the request does not normally contain a hostname.
- The Cookie header is used to submit additional parameters that the server has issued to the client

HTTP Responses

A typical HTTP response is as follows:

```
HTTP/1.1 200 OK
Date: Sat, 19 May 2007 13:49:37 GMT
Server: IBM_HTTP_SERVER/1.3.26.2  Apache/1.3.26 (Unix)
Set-Cookie: tracking=tI8rk7joMx44S2Uu85nSWc
Pragma: no-cache
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: text/html;charset=ISO-8859-1
Content-Language: en-US
Content-Length: 24246

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html lang="en">
<head>
    <meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
...
```

The first line of every HTTP response consists of three items, separated by spaces:

- The HTTP version being used.
- A numeric status code indicating the result of the request. 200 is the most common status code; it means that the request was successful and the requested resource is being returned.

Some other points of interest in response are:

- The Server header contains a banner indicating the web server software being used, and sometimes other details such as installed modules and the server operating system.
- The Set-Cookie header is issuing the browser a further cookie; this will be submitted back in the Cookie header of subsequent requests to this server.
- The Pragma header is instructing the browser not to store the response in its cache, and the Expires header also indicates that the response content expired in the past and so should not be cached.
- Almost all HTTP responses contain a message body following the blank line after the headers, and the Content-Type header indicates that the body of this message contains an HTML document.
- The Content-Length header indicates the length of the message body in bytes.

Status Codes

Each HTTP response message must contain a three digit status code in its first line, indicating the result of the request. The status codes fall into five groups, according to the first digit of the code:

1xx — Informational.
2xx — The request was successful.
3xx — The client is redirected to a different resource.
4xx — The request contains an error of some kind.
5xx — The server encountered an error fulfilling the request.

URLs

A uniform resource locator (URL) is a unique identifier for a web resource, via which that resource can be retrieved. The format of most URLs is as follows:

protocol://hostname[:port]/[path/]file[?param=value]

Several components in this scheme are optional, and the port number is normally only included if it diverges from the default used by the relevant protocol.

The URL used to generate the HTTP request shown earlier is:

http://wahh-app.comm/books/search.asp?q=wahh

In addition to this absolute form, URLs may be specified relative to a particular host, or relative to a particular path on that host, for example:

/books/search.asp?q=wahh

search.asp?q=wahh

These relative forms are often used in web pages to describe navigation within the web site or application itself.

Cookies

Cookies are a key part of the HTTP protocol, which can frequently be used as a vehicle for exploiting vulnerabilities.

The cookie mechanism enables the server to send items of data to the client, which the client stores and resubmits back to the server. Unlike the other types of request parameters (those within the URL query string or the message body), cookies continue to be resubmitted in each subsequent request without any particular action required by the application or the user.

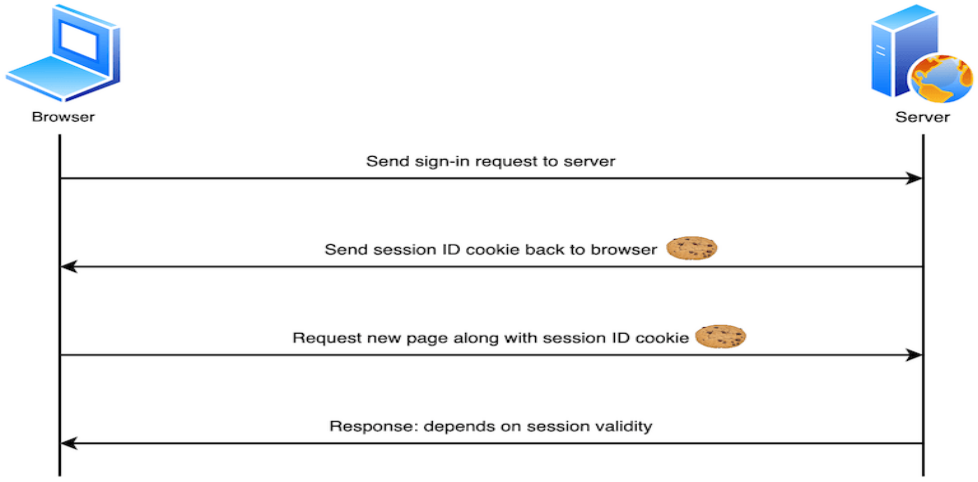
A server issues a cookie using the Set-Cookie response header:

Set-Cookie: tracking=tI8rk7joMx44S2Uu85nSWc

The user's browser will then automatically add the following header to subsequent requests back to the same server:

Cookie: tracking=tI8rk7joMx44S2Uu85nSWc

Cookies normally consist of a name/value pair, as shown, but may consist of any string that does not contain a space.



Multiple cookies can be issued by using multiple Set-Cookie headers in the server's response, and are all submitted back to the server in the same Cookie header, with a semicolon separating different individual cookies.

In addition to the cookie's actual value, the Set-Cookie header can also include any of the following optional attributes, which can be used to control how the browser handles the cookie:

- **expires —** Used to set a date until which the cookie is valid. This will cause the browser to save the cookie to persistent storage, and it will be reused in subsequent browser sessions until the expiration date is reached. If this attribute is not set, the cookie is used only in the current browser session.
- **domain —** Used to specify the domain for which the cookie is valid. This must be the same or a parent of the domain from which the cookie is received.
- **path —** Used to specify the URL path for which the cookie is valid.
- **secure** – If this attribute is set, then the cookie will only ever be submitted in HTTPS requests.
- **HttpOnly —** If this attribute is set, then the cookie cannot be directly accessed via client-side JavaScript.

Web Functionality

Web applications are developed using numerous different technologies to deliver their functionality.

Server-Side Functionality

The early World Wide Web contained entirely static content. Web sites consisted of various resources such as HTML pages and images, which were simply loaded onto a web server and delivered to any user who requested them. Each time a particular resource was requested, the server responded with the same content.

Now, web applications still typically employ a fair number of static resources. However, a large amount of the content that they present to users is generated dynamically. When a user requests a dynamic resource, the server's response is created on the fly, and each user may receive content that is uniquely customized for them.

Dynamic content is generated by scripts or other code executing on the server. These scripts are akin to computer programs in their own right—they have various inputs, perform processing on these, and return their outputs to the user.

When a user's browser makes a request for a dynamic resource, it does not normally simply ask for a copy of that resource. In general, it will also submit various parameters along with its request. It is these parameters that enable the server-side application to generate content that is tailored to the individual user. There are three main ways in which HTTP requests can be used to send parameters to the application:

- In the URL query string.
- In HTTP cookies.
- In the body of requests using the POST method.

In addition to these primary sources of input, the server-side application may in principle use any part of the HTTP request as an input to its processing.

Web applications employ a wide range of technologies on the server side to deliver their functionality. These include:

- Scripting languages such as PHP, VBScript, and Perl.
- Web application platforms such as **ASP.NET and Java**.
- Web servers such as Apache, IIS, and Netscape Enterprise.
- Databases such as MS-SQL, Oracle, and MySQL.
- Other back-end components such as file systems, SOAP-based web services, and directory services.

The Java Platform

Used since several several years

Developed by Sun Microsystems

Multi-tiered and load-balanced architectures, modular development and code reuse

Platform independent - The Java Platform can be run on several underlying operating systems, including Windows, Linux, and Solaris.

Uses a number of components –

An **Enterprise Java Bean** (EJB) is a relatively heavyweight software component that encapsulates the logic of a specific business function within the application. EJBs are intended to take care of various technical challenges that application developers must address, such as transactional integrity.

A **Plain Old Java Object** (POJO) is an ordinary Java object, as distinct from a special object like an EJB. POJO is normally used to denote objects that are user-defined and much simpler and more lightweight than EJB.

A **Java Servlet** is an object that resides on an application server and receives HTTP requests from clients and returns HTTP responses. There are numerous useful interfaces that Servlet implementations can use to facilitate the development of useful applications.

A Java **web container** is a platform or engine that provides a runtime environment for Java-based web applications.

Many Java web applications employ third-party and open source components, such as -

- Authentication — JAAS, ACEGI
- Presentation layer — SiteMesh, Tapestry
- Database object relational mapping — Hibernate
- Logging — Log4J

ASP.NET

ASP.NET is Microsoft's web application framework and is a direct competitor to the Java Platform. ASP.NET is several years younger than Java Platform.

ASP.NET uses Microsoft's .NET Framework, which provides a virtual machine (the Common Language Runtime) and a set of powerful APIs. ASP.NET applications can be written in any .NET language, such as C# or VB.NET.

PHP

It is a highly powerful and rich framework for developing web applications. It is often used in conjunction with other free technologies in the LAMP stack (comprising Linux, Apache, MySQL, and PHP).

Numerous open source applications and components have been developed using PHP.

for example:

Bulletin boards — PHPBB, PHP-Nuke

Administrative front ends — PHPMyAdmin

Web mail — SquirrelMail, IlohaMail

Photo galleries — Gallery

Shopping carts — osCommerce, ECW-Shop

Wikis — MediaWiki, WakkaWikki

Client-Side Functionality

In order for the server-side application to receive user input and actions, and present the results of these back to the user, it needs to provide a client-side user interface. Because all web applications are accessed via a web browser, these interfaces all share a common core of technologies.

HTML

The core technology used to build web interfaces is the hypertext markup language (HTML). This is a tag-based language that is used to describe the structure of documents that are rendered within the browser. From its simple beginnings as a means of providing basic formatting to text documents, HTML has developed into a rich and powerful language that can be used to create highly complex and functional user interfaces.

Hyperlinks

A large amount of communication from client to server is driven by the user clicking on hyperlinks. In web applications, hyperlinks frequently contain preset request parameters. These are items of data which are never entered by the user but which are submitted because the server placed them into the target URL of the hyperlink on which the user clicks. For example, a web application might present a series of links to news stories, each having the following form:

Sale now on!

When a user clicks on this link, the browser makes the following request:

GET /news/showStory?newsid=19371130&lang=en HTTP/1.1

Host: wahh-app.com

...

The server receives the two parameters in the query string (newsid and lang) and uses their values to determine what content should be presented to the user.

Forms

While hyperlink-based navigation is responsible for the majority of client-toserver communications, in most web applications there is a need for more flexible ways of gathering input and receiving actions from users. HTML forms are the usual mechanism for allowing users to enter arbitrary input via their browser. A typical form is as follows:

<form action=”/secure/login.php?app=quotations” method=”post”>

username: <input type=”text” name=”username”>

password: <input type=”password” name=”password”>

<input type=”hidden” name=”redir” value=”/secure/home.php”>

<input type=”submit” name=”submit” value=”log in”>

</form>

When the user enters values into the form and clicks the submit button, the browser makes a request like the following:

POST /secure/login.php?app=quotations HTTP/1.1

Host: wahh-app.com

Content-Type: application/x-www-form-urlencoded

Content-Length: 39

Cookie: SESS=GTnrpx2ss2tSWSnhXJGyG0LJ47MXRsjcFM6Bd

username=daf&password=foo&redir=/secure/home.php&submit=log+in

In this request, there are several points of interest reflecting how different aspects of the request are used to control server-side processing:

Because the HTML form tag contained an attribute specifying the POST method, the browser uses this method to submit the form, and places the data from the form into the body of the request message.

In addition to the two items of data entered by the user, the form contains a hidden parameter (redir) and a submit parameter (submit). Both of these are submitted in the request and may be used by the server-side application to control its logic.

The target URL for the form submission contains a preset parameter (app), as in the hyperlink example shown previously. This parameter may be used to control the server-side processing.

The request contains a cookie parameter (SESS), which was issued to the browser in an earlier response from the server. This parameter may be used to control the server-side processing.

JavaScript

Hyperlinks and forms can be used to create a rich user interface capable of easily gathering most kinds of input which web applications require. However, most applications employ a more distributed model, in which the client side is used not simply to submit user data and actions but also to perform actual processing of data. This is done for two primary reasons:

- It can **improve the application's performance**, because certain tasks can be carried out entirely on the client component, without needing to make a round trip of request and response to the server.
- It can **enhance usability**, because parts of the user interface can be dynamically updated in response to user actions, without needing to load an entirely new HTML page delivered by the server.

JavaScript is a relatively simple but powerful programming language that can be easily used to extend web interfaces in ways that are not possible using HTML alone. It is commonly used to perform the following tasks:

- Validating user-entered data before this is submitted to the server, to avoid unnecessary requests if the data contains errors.
- Dynamically modifying the user interface in response to user actions; for example, to implement drop-down menus and other controls familiar from non-web interfaces.
- Querying and updating the document object model (DOM) within the browser to control the browser's behavior.

A significant development in the use of JavaScript has been the appearance of AJAX(Asynchronous JavaScript and XML) techniques for creating a smoother user experience which is closer to that provided by traditional desktop applications.

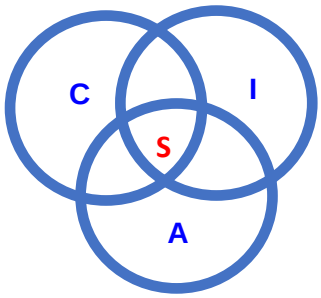
Thick Client Components

Going beyond the capabilities of JavaScript, some web applications employ thicker client technologies that use custom binary code to extend the browser's built-in capabilities in arbitrary ways. The thick-client technologies used in web applications are:

■■ Java applets

■■ ActiveX controls

■■ Shockwave Flash objects



S = Secure

Basic Components of Security

The basic components of security include confidentiality, integrity, availability, authentication, and non-repudiation. The first three are the main components of security.

1. Confidentiality : Data is accessible by authorized users.
2. Integrity : Data sent is ensured to be correct and not edited.
3. Availabilty : Data is available whenever required.
4. **Authentication**: Verifying the identity of a user, system, or entity to ensure they are who they claim to be.
5. **Authorization**: Determining and granting specific permissions or access levels to authenticated users based on their identity.
6. **Non-repudiation**: Ensuring that actions or transactions cannot be denied by the entity that performed them, typically using digital signatures or audit logs.

Example Scenario: **Online Shopping Platform (e.g., Amazon)**

1. Confidentiality:

When a user enters their credit card details during checkout, the platform uses SSL/TLS encryption to ensure sensitive data remains private and secure during transmission.

2. Integrity:

The platform verifies the integrity of orders by using checksum mechanisms to ensure that the total amount and order details are not altered during processing or transmission.

3. Availability:

To handle millions of simultaneous users, the platform uses cloud-based infrastructure with failover mechanisms and distributed denial-of-service (DDoS) protection to ensure the website is always accessible.

4. Authentication:

Customers must log in using their email and password, or two-factor authentication (2FA), to access their accounts and make purchases.

5. Authorization:

A logged-in user is authorized to perform actions such as adding items to the cart, placing an order, or viewing their personal purchase history, while restricted from accessing admin functionalities.

6. Non-repudiation:

After placing an order, the platform generates a digital invoice with a unique order ID, timestamp, and delivery details, ensuring the customer cannot deny making the purchase.

Encoding Schemes

Web applications employ several different encoding schemes for its data. Both the HTTP protocol and the HTML language are text-based, and different encoding schemes have been devised to ensure that unusual characters and binary data can be safely handled by these different encoding mechanisms.

URL Encoding

URLs are permitted to contain only the printable characters in the US-ASCII character set(eg. Null Ack,del,etc are not included). The URL encoding scheme encodes any problematic characters within the extended ASCII character set so that they can be safely transported over HTTP. The URL-encoded form of any character is the % prefix followed by the two-digit hexadecimal.

Some examples of characters that are commonly URL-encoded are shown here:

%20 space

%0a new line

A further encoding to be aware of is the + character, which represents a URLencoded space.

Unicode Encoding

Unicode is a character encoding standard that is designed to support all of the writing systems(languages) of the world. It employs various encoding schemes, like UTF-16,UTF-8, UTF-32.

UTF-16 encoding works in a similar way to URL-encoding. For transmission over HTTP, the 16-bit Unicode-encoded form of a character is the %u prefix followed by the character's Unicode code point expressed in hexadecimal.

UTF-8 is a variable-length encoding standard that employs one or more bytes to express each character.

HTML Encoding

HTML encoding is a scheme used to represent problematic characters so that they can be safely incorporated into an HTML document. Various characters have special meaning within HTML and are used to define the structure of a document rather than its content. To use these characters safely as part of the document's content, it is necessary to HTML-encode them.

HTML encoding defines numerous HTML entities to represent specific literal characters, for example:

&quot;	“
&apos;	‘
&amp;	&
&lt;	<
&gt;	>

Base64 Encoding

Base64 encoding allows any binary data to be safely represented using only printable ASCII characters. It is commonly used for encoding email attachments for safe transmission over SMTP, and is also used to encode user credentials in basic HTTP authentication.

Base64 encoding processes input data in blocks of three bytes. Each of these blocks is divided into four chunks of six bits each. Six bits(2^6) of data allow for 64 different possible combinations, and so each chunk can be represented using a set of 64 characters.

Base64 encoding employs the following character set, which contains only printable ASCII characters:

ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/

(26+26+10+2 = 64 characters) are only represented in Base64 encoding. [by using 6 bits]

If the final block of input data results in less than three chunks of output data, then the output is padded with one or two = characters.

For example, the Base64-encoded form of ‘Man’ is -

- ASCII Binary: M = 01001101, a = 01100001, n = 01101110
- Combine: 01001101 01100001 01101110
- Split into 6-bit groups: 010011 010110 000101 101110
- Map to Base64: TWFu
- Output: TWFu

2) *The Web Application Hacker’s Handbook* is:

VGhlIFdlYiBBcHBsaWNhdGlvbiBIYWNrZXIncyBIYW5kYm9vaw==

Many web applications make use of Base64 encoding for transmitting binary data within cookies and other parameters.

Hex Encoding

Many applications use straightforward hexadecimal encoding when transmitting binary data, using ASCII characters to represent the hexadecimal block.

For example, hex-encoding the username "daf" within a cookie would result in:

646166 [d (100)->01100100, a (97)-> 01000001….]

Comparison of **URL encoding**, **Unicode encoding**, **HTML encoding**, **Base64 encoding**, and **Hex encoding** in table format:

Aspect	**URL Encoding**	**Unicode Encoding**	**HTML Encoding**	**Base64 Encoding**	**Hex Encoding**
Purpose	Encode special characters in URLs for safe transmission.	Encode characters as Unicode code points.	Represent reserved or special characters in HTML.	Encode binary data into text for safe transmission/storage.	Represent binary data in a human-readable hexadecimal format.
Characters Encoded	Reserved characters (e.g., &, ?, /) and unsafe characters.	All characters represented as Unicode.	Reserved (<, >, &) and non-ASCII characters.	Binary data, including all ASCII and non-ASCII.	Binary data, including all ASCII and non-ASCII.
Encoding Method	Replaces characters with % followed by two hexadecimal digits (ASCII values).	Encodes characters as numeric code points in decimal or hexadecimal.	Uses entities (&name;) or numeric references (&#xNNNN;).	Converts 3 bytes of binary data into 4 ASCII characters.	Converts each byte of binary data into 2 hexadecimal digits.
Example Input	Hello World!	-		Binary file, image, or text	Binary file, image, or text
Example Output	Hello%20World%21	U+1F60A (hex) or &#12852	<strong>	SGVsbG8gd29ybGQh	48656C6C6F20576F726C6421

Aspect	URL Encoding	Unicode Encoding	HTML Encoding	Base64 Encoding	Hex Encoding
		2; (HTML entity)			
Use Case	Safe transmission of URLs and query parameters.	Standard for text encoding across languages.	Displaying reserved characters in HTML pages.	Transmitting binary data as text over protocols like email.	Debugging, binary data representation, and cryptography.
Readability	Partially human-readable (%20 → Space).	Not human-readable unless familiar with Unicode.	Partially human-readable (< → <).	Not human-readable; appears as a string of characters.	Partially human-readable; appears as a string of hex digits.
Length Increase	Slight increase (depends on number of special characters).	None (depends on encoding type).	Slight increase for special characters.	Increases output by ~33% (4 characters for every 3 bytes).	Doubles the size (1 byte → 2 hex digits).
Example Use Cases	Embedding special characters in URLs, forms.	Internatio nal text display and storage.	Escaping reserved HTML symbols in web content.	Embedding images or files in email, JSON, or XML.	Representing binary data in debugging and cryptography.
Tools to Decode	Browser, decodeURICompo nent() in JS.	Unicode parsers (e.g., UTF-8 decoder).	Web browsers or HTML parsers.	Decoders in programming languages (e.g., Python, JavaScript).	Hex converters or programming languages.

- **URL Encoding**: For URLs, replaces reserved characters with %-encoded values.
- **Unicode Encoding**: Encodes text as standardized code points for multiple languages.
- **HTML Encoding**: Escapes special characters for safe use in HTML.

- **Base64 Encoding**: Encodes binary data as ASCII text for transmission.
- **Hex Encoding**: Represents data as a string of hexadecimal values for readability and debugging.

Enumerating Content and Functionality

In any attack, the first task is to map the target application's content and functionality, to establish how it functions, how it attempts to defend itself, and what technologies it uses.

[[Enumerating Content and Functionality means to systematically check all the individual components, features, and actions that are available within a system/ application, so that the attacker understands the system of what it can do and what information it contains.]]

In a typical application, the majority of the content and functionality can be identified via manual browsing. The basic approach is to walk through the application starting from the home page, following every link and navigating through all multistage functions (such as user registration or password resetting). If the application contains a "**site map**," this can provide a useful starting point for enumerating content.

However, to perform a rigorous inspection of the enumerated content, and to obtain a comprehensive record of everything identified, it is necessary to employ some more advanced techniques than simple browsing.

[Enumeration meaning -> the action of establishing the number of things one by one.]

Web Spidering

Various tools exist which perform automated spidering of web sites. These tools work by requesting a web page, parsing it for links to other content, and then requesting these, continuing recursively until no new content is discovered.

[parsing - analyzing and processing the structure]

Web application spiders attempt to achieve a higher level of coverage by also parsing **HTML forms** and submitting these back to the application using various preset or random values. This can enable them to walk through multistage functionality, and to follow forms-based navigation (e.g., where drop-down lists are used as content menus). Some tools also perform some parsing of **client-side JavaScript** to extract URLs pointing to further content.

Some of enumerating tools are –

- Paros
- Burp Spider (part of Burp Suite)
- WebScarab

Limitations of fully automated content enumeration:

Unusual navigation mechanisms (such as menus dynamically created using complicated JavaScript code) are often not handled properly by these tools, and so they may miss whole areas of an application.

Input data taken by automated tools **through forms** may not be of proper format and fails validation, so the spider fails to discover the further content or functions. (For example, a user registration form may contain fields for name, email address, telephone number, and ZIP code. An automated application spider may enter wrong number of numbers in telephone number and an error message saying that one or more of the items submitted were invalid.)

Automated spiders typically use URLs as identifiers of unique content. To avoid continuing indefinitely, they recognize when linked content is requested again. However, many applications use forms-based navigation in which the same URL may return very different content and functions.

User-Directed Spidering

The user walks through the application in the normal way using a standard browser, attempting to navigate through all of the application's functionality. As he does so, the resulting traffic is passed through a tool combining an intercepting proxy and spider, which monitors all requests and responses. The tool builds up a map of the application, incorporating all of the URLs visited by the browser, and also parses all of the application's responses in the same way as a normal application-aware spider and updates the site map with the content and functionality it discovers.

Eg - Burp Suite and WebScarab

Benefits compared to basic spidering approach -

The unusual or **complex mechanisms for navigation** are followed by the user in a normal way, the functions or content accessed by the user will be processed(mapped) by the proxy/spider tool.

The user controls all data submitted to the application and can ensure that **data validation** requirements are met.

The user can log in to the application in the usual way, by **entering authenticated details** and remain active throughout the mapping process. If any action performed results in session termination, the user can log in again and continue browsing.

Discovering Hidden Content

There can be content and functionality which is not directly reachable from the main visible content (home page of application).

Example –

Functionality that has been implemented for testing or debugging purposes and has never been removed.

Different functionalities are designed to different categories of users (for example, anonymous users, authenticated regular users, and administrators). Users at one privilege level who perform exhaustive spidering of the application may miss functionality that is visible to users at other levels.

Guessing different parameter values as a request to server.

An attacker who discovers the functionality may be able to use it to elevate the privileges within the application. (admin – change the password and use others privileges).

Vulnerabilities can be identified at -

Backup copies of live files (created dynamically) – this could be changed. So that it can then be identify all content and functionality within application.

Old versions of files that have not been removed from the server – may be open to attacks.

Configuration and include files containing sensitive data such as database credentials.

Log files that may contain sensitive information such as valid usernames, session tokens, URLs visited, actions performed, and so on.

Automated and manual techniques + degree of luck = Effective discovery of hidden content.

Analyzing the Application

Enumerating (tracing) as much of the application's content as possible is only one element of the mapping process. Equally important is the task of analyzing **the application's functionality, behavior, and technologies employed**, in order to identify the key attack surfaces that it exposes.

Regular security assessments help mitigate(reduce) risks and protect applications from cyber threats.

Some key areas to investigate are:

The core functionality of the application —the actions that it can be leveraged to perform when used as intended. (Eg – Banking software – to check the account details, transfers).

Other more peripheral behavior of the application, including off-site links (phishing), error messages, administrative and logging functions, use of redirects, and so on.

Different locations at which user-supplies input to server — every URL, query string parameter, item of POST data, cookie, etc.

The analysis of application is done by -

- **Identifying Entry Points for User Input**
- **Identifying Server-Side Technologies**
- **Identifying Server – Side Functionality**

Identifying Entry Points for User Input

The user's input for server side processing is obtained by reviewing the HTTP requests that are generated as the user walk through the application's functionality. The key locations to retrieve user inputs are:

- Every parameter submitted within the URL query string.
- Every parameter submitted within the body of a POST request.
- Every cookie.
- Every other HTTP header that in rare cases may be processed by the application, in particular the User-Agent, Referer, Accept, Accept- Language, and Host headers.

Identifying Server-Side Technologies

It is normally possible to fingerprint(chalk-out) the technologies employed on the server via various clues and indicators.

Some of the techniques are - Banner Grabbing, HTTP Fingerprinting, File Extensions, Directory Names, Session ID etc

Banner Grabbing - servers disclose some information about the web server software and its other components like - HTTP Server header discloses a huge amount of detail about some installations.

HTTP Fingerprinting- tools used to identify web technologies based on responses and behavior.

File Extensions - File extensions used within URLs discloses the platform or programming language used to implement the relevant functionality.

Eg -

asp—Microsoft Active Server Pages
aspx—Microsoft ASP.NET
jsp—Java Server Pages
php—the PHP language

pl—the Perl language
py—the Python language
dll—usually compiled native code (C or C++)

Directory Names - subdirectory names in the path, usually indicates the presence of an associated technology.

For example:

servlet—Java servlets

pls—Oracle Application Server PL/SQL gateway

WebObjects or {function}.woa—Apple WebObjects

rails—Ruby on Rails

Session Tokens - Many web servers and web application generates session tokens by default with names that provide information about the technology in use.

For example:

JSESSIONID—The Java Platform

ASPSESSIONID—Microsoft IIS server

PHPSESSID—PHP

Identifying Server-Side Functionality

Dissecting Requests – analysing the URL request to understand many concepts like the platform used, other technologies

https://wahh-app.com/calendar.jsp?name=new%20applicants&isExpired=0&startDate=22%2F09%2F2006&endDate=22%2F03%2F2007&OrderBy=name

As we have seen, the .jsp file extension indicates that Java Server Pages are in use. A database is used to retrieve its information as there is OrderBy parameter. This suggests that a back-end database is being used, and that the value you submit may be used as the ORDER BY clause of a SQL query. This parameter may well be vulnerable to SQL injection, as may any of the other parameters if they are used in database queries.

Extrapolating Application Behavior

Extrapolating application behavior is a **technique used during web application analysis** to predict how an application functions in different scenarios, based on observable patterns or behavior.

In this approach extra parameters are guessed based on existing parameters. This approach helps attackers discover hidden features, input parameters, and security flaws by analyzing **responses, workflows, and data handling mechanisms**.

Attacker makes educated guesses about its **internal logic, hidden features, and untested functionalities**.

Example:

- If a URL looks like /product?id=123, you can try modifying the id to /product?id=admin or /product?debug=true to check if hidden behaviors exist.
- If an application shows different error messages for valid vs. invalid input, it may be vulnerable to **username enumeration**.

Techniques for Extrapolating Behavior

1. URL Pattern Analysis

Applications often have **predictable URL structures**. By analyzing known URLs, you can extrapolate others.

- **Example:**
  If /user/profile/123 shows a user profile, then /user/profile/124 might be another user's profile.

2. Parameter Discovery

If certain parameters control application behavior, attackers may try guessing related parameters.

- **Example:**
  If you discover &debug=true, you might also try &test=true, &admin=true, or &verbose=true to see if additional functionality exists.

3. Error Message Analysis

Different error messages can reveal application logic and security flaws.

- **Example:**
  A login form that says User not found vs. Invalid password allows **username enumeration**.
- **Example:**
  SQL errors like Unknown column 'username' reveal backend database structure.

Tools to Help with Extrapolation

- **Burp Suite:** For intercepting requests, manipulating parameters, and automating discovery.

- **Dirbuster:** For brute-forcing hidden directories and files.
- **Nmap:** For discovering open ports and services that might expose functionality.
- **WhatWeb/Wappalyzer:** For identifying technologies and potential vulnerabilities.

Mapping the Attack Surface

The final stage of the mapping process is to identify the various attack surfaces exposed by the application, and the potential vulnerabilities that are commonly associated with each one.

Example -

- Client-side validation —Checks may not be replicated on the server.
- Database interaction —SQL injection.
- File uploading and downloading—Path traversal vulnerabilities.
- Display of user-supplied data—Cross-site scripting.
- Dynamic redirects—Redirection and header injection attacks.
- Session state—Predictable tokens, insecure handling of tokens.
- Access controls—Horizontal and vertical privilege escalation.
- Email interaction—Email and/or command injection.

Bypassing Client-Side Controls

- Transmitting Data via the Client
- Capturing User Data
- Handling Client-Side Data Securely

Transmitting Data via the Client

Application may pass data from the client to servers in a form that is not directly visible or modifiable by the end user. The unseen data transmitted from the client can be modified by the interceptor, and leaves the application vulnerable to one or more attacks.

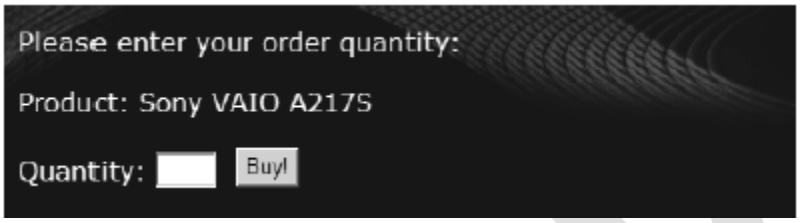
The unseen data is in various ways –

- Hidden Form Fields
- HTTP Cookies
- Referrer Header
- URL Parameters
- Opaque data

Hidden Form Fields-

Hidden HTML form fields are a common mechanism for transmitting data via the client in a superficially unmodifiable way. If a field is flagged as hidden, it is not displayed on-screen. However, the field's name and value are stored within the form and sent back to the application when the user submits the form.

The classic example of this security flaw is a retailing application that stores the prices of products within hidden form fields.



The code behind this form is as follows:

<form action="order.asp" method="post">
Product: Sony VAIO A217S
Quantity: <input size="2" name="quantity">
<input name="price" type="hidden" value="1224.95">
<input type="submit" value="Buy!">
</form>
Notice the form field called price, which is flagged as hidden. This field will be sent to the server when the user submits the form:

POST /order.asp HTTP/1.1
Host: wahh-app.com
Content-Length: 23
quantity=1&price=1224.95

As the price field is not displayed on-screen, it is not editable by the user, this restriction can be overcome in different ways. So that user can edit the price.

One way to achieve this is to save the source code for the HTML page, edit the value of the field, reload the source into a browser, and click the Buy button.

Another method is to use an intercepting proxy to modify the desired data on the fly. An intercepting proxy is tremendously useful when attacking a web application.

There are numerous such tools available. Example - Burp Proxy (part of Burp Suite), WebScarab, Paros etc.

The proxy sits between your web browser and the target application. It intercepts every request issued to the application, and every response received back. It can trap and intercept any message for inspection or modification.

HTTP Cookies

Another common mechanism for transmitting data via the client is HTTP cookies. These are not normally displayed on-screen or directly modifiable by the user. They can be modified using an intercepting proxy, either by changing the server response that sets them, or subsequent client requests that issue them.

Suppose, a customer receives the following response from the server, when a user logs in to the application -

HTTP/1.1 302 Found

Location: /home.asp

Set-Cookie: SessId=191041-1042

Set-Cookie: UID=1042

Set-Cookie: DiscountAgreed=25

This response sets three cookies, the first appears to be a session token, which may be vulnerable to sequencing attacks. The second appears to be a user identifier, which can potentially be leveraged to exploit access control weaknesses. The third appears to represent a discount rate that the customer will receive on purchases.

If the application trusts the value of third cookie – DiscountAgreed when it is submitted back to the server, then customers can obtain arbitrary discounts by modifying its value.

For example:

POST /order.asp HTTP/1.1
Host: wahh-app.com
Cookie: SessId=191041-1042; UID=1042; DiscountAgreed=99
Content-Length: 23
quantity=1&price=1224.95

URL Parameters

Applications frequently transmit data via the client using preset URL parameters. For example, when a user browses the product catalogue, the application may provide them with hyperlinks to URLs like the following:

https://wahh-app.com/browse.asp?product=VAIOA217S&price=1224.95

The values of any URL parameters can be modified by the user without the use of tools or by using an intercepting proxy (if parameters are hidden).

The Referer Header

Browsers include the Referer header within most HTTP requests. This is used to indicate the URL of the page from which the current request originated. The request may be originated either because the user clicked a hyperlink or submitted a form.

For example, consider a mechanism that enables users to reset their password if they have forgotten it. The application requires users to proceed through several steps in a defined sequence, before they actually reset their password's value with the following request:

```
POST /customer/ResetForgotPassword.asp HTTP/1.1
Referer: http://wahh-app.com/customer/ForgotPassword.asp
Host: wahh-app.com
Content-Length: 44
uname=manicsprout&pass=secret&confirm=secret
```

The application may use the Referer header to verify that this request originated from the correct stage (ForgotPassword.asp), and if so allow the user to reset their password.

However, using an intercepting proxy the Referer header value may be reset by the attacker as required.

Opaque Data

Sometimes, data transmitted via the client is not transparent, because it has been encrypted in some way. For example, instead of seeing a product's price stored in a hidden field, you may see some cryptic value being transmitted:

```
<form action="order.asp" method="post">
<p>Product: Sony VAIO A217S</p>
<p>Quantity: <input size="2" name="quantity">
<input name="enc" type="hidden" value="262a4844206559224f456864206668643
2657720313839326544 48a3524846346672336832773 84f22455565333272336664552 25
242452a526674696f6471">
<input type="submit" value="Buy!"></p>
</form>
```

When the form is submitted, the server-side application will decrypt the opaque string and perform some processing on its plaintext value. This further processing may be vulnerable to any kind of bug.

Capturing User Data

HTML forms are the simplest and most common mechanism for capturing input from the user and submitting it to the server. In the most basic uses of this method, users type data into named text fields, which are submitted to the server as name/value pairs. However, forms can be used in other ways, which are designed to impose restrictions or perform validation checks on the user-supplied data.

These data collected from client may be vulnerable to attack.

Length Limits

Suppose the length of a form field is fixed, a user cannot enter more than that max_length, but an interceptor can modify the length and change the parameter value.

```
<form action=”order.asp” method=”post”>
<p>Product: Sony VAIO A217S</p>
<p>Quantity: <input size=”2” maxlength=”3” name=”quantity”>
<input name=”price” type=”hidden” value=”1224.95”>
<input type=”submit” value=”Buy!”></p>
</form>
```

The quantity value can be increased by the hacker, by changing the maxlength value.

Script based validation

A user registration form might contain fields for name, email address, telephone number, and ZIP code, all of which expect different types of input. To validate all fields at client-side, scripts can be used.

Consider the following example:

```
<script>
function ValidateForm(theForm)
{
        var isInteger = /^\d+$/
        if(!isInteger.test(theForm.quantity.value))
        {
                alert(“Please enter a valid quantity”);
                return false;
        }
return true;
}
</script>

<form action=”order.asp” method=”post” onsubmit=”return
ValidateForm(this)“>
```

```
<p>Product: Sony VAIO A217S</p>
<p>Quantity: <input size=”2” name=”quantity”>
<input name=”price” type=”hidden” value=”1224.95”>
<input type=”submit” name=”buy” value=”Buy!”></p>
</form>
```

The onsubmit attribute of the form tag instructs the browser to execute the ValidateForm function when the user clicks the submit button and to submit the form only if this function returns true. This mechanism enables the client-side validation of the inputs by user.

Here, the validation checks whether the data entered in the amount field is an integer.

Disable elements

If an element on an HTML form is flagged as disabled, it appears on-screen but is usually grayed out and is not editable or usable in the way an ordinary control is used. Also, it is not sent to the server when the form is submitted. For example,

consider the following form:

```
<form action=”order.asp” method=”post”>
<p>Product: <input disabled=”true” name=”product” value=”Sony VAIOA217S”></p>
<p>Quantity: <input size=”2” name=”quantity”>
<input name=”price” type=”hidden” value=”1224.95”>
<input type=”submit” value=”Buy!”></p>
</form>
```

This includes the name of the product as a disabled text field and appears onscreen. An attacker can modify this text, but modifying the name of the product may not appear to be as promising an attack as modifying its price.

However, if this parameter was a normal text, then it may be vulnerable to many kinds of bugs such as SQL injection or cross-site scripting.

Capturing User Data: Thick-Client Components

Besides HTML forms, the other main method for capturing, validating, and submitting user data is to use technologies like Java applets, ActiveX controls, and Shockwave Flash objects.

Data is captured in various different ways, by input forms or by interacting with the client operating system’s file system or registry(using different technologies).

Java Applets

Java applets are small Java programs that run inside a web browser using the Java Virtual Machine (JVM). They were commonly used for creating **interactive web applications**, such as games, charts, and real-time data visualization.

Since Java applets run on the client-side, they can **perform complex operations**, such as processing user input, encrypting data, or interacting with local system files.

Attackers can exploit **insecure Java applets** to bypass client-side validation and send **malicious data** to the server.

Many web applications **rely on Java applets** to validate user input, assuming that their logic cannot be manipulated, which can lead to security vulnerabilities. By reverse-engineering Java applets using **decompilers**, hackers can analyze their internal logic and identify weaknesses. Attackers can **modify applet code** or intercept its communication with the server to inject harmful payloads, such as SQL injection or cross-site scripting (XSS).

ActiveX controls

ActiveX controls are small software components developed by Microsoft that run within Internet Explorer to enhance web functionality. They allow web applications to interact with the Windows operating system, enabling tasks like file manipulation, system configuration, and automation.

Many web applications rely on ActiveX controls for data validation, assuming that the logic cannot be bypassed, which creates potential security weaknesses.

Hackers use ActiveX decompilers to reverse-engineer these components and uncover vulnerabilities such as buffer overflows or insecure method calls. Unsigned or poorly coded ActiveX controls can be hijacked by malware, making them a common vector for drive-by downloads and phishing attacks.

Handling Client-Side Data Securely

As you have seen, the core security problem with web applications arises because client-side components and user input are outside of the server's direct control. The client, and all of the data received from it, is inherently untrustworthy.

Transmitting Data via the Client

Securely transmitting data via the client is crucial to protect sensitive information from interception and unauthorized access during transmission. There are various techniques and best practices to ensure secure data transmission, including:

- Importance of Transport Layer Security (TLS) in data transmission: **TLS encrypts** all communications between the client and the server, preventing **MITM attacks**. **HTTPS ensures** that all requests are securely transmitted and not readable by attackers.

- Securing HTTP connections with HTTPS: Implementing **HTTPS** ensures that all data transmitted between the client and server is **encrypted using TLS**, preventing unauthorized interception or modification. This protects against **man-in-the-middle (MITM) attacks**, ensuring that sensitive information like passwords and payment details remain secure during transmission.

- Utilizing Secure Sockets Layer (SSL) SSL certificates **authenticate the identity of a website** and establish an encrypted connection between the client and server, ensuring data privacy. By enabling **HTTPS**, SSL certificates prevent attackers from intercepting or tampering with sensitive information during transmission.

- Implementing HTTP security headers for enhanced protection: Utilizing HTTP security headers like Content-Security-Policy (CSP) to bolster security.

Validating Client-Generated Data/ User-Generated data:

Client-side data validation is essential to ensure that the data submitted by users is safe and free from malicious content. The importance of validating client-generated data and the techniques used to achieve this goal:

- Understanding the risks associated with client-side data manipulation: Trusting data solely on the **client-side** is risky because attackers can **modify form fields, cookies, or API requests** using browser tools or scripts. This can lead to **security vulnerabilities** such as unauthorized access, price manipulation in e-commerce sites, or injection attacks.

- Importance of server-side validation as the first line of defence: Server-side validation is **crucial** because it ensures that all user input is thoroughly checked before processing, preventing **data tampering and malicious attacks** like SQL Injection and Cross-Site Scripting (XSS). Unlike client-side validation, which can be bypassed, server-side validation acts as the **first line of defence** by enforcing strict security rules.

- Implementing client-side validation for improved user experience: Client-side validation enhances **user experience** by providing instant feedback on input errors, but it must be complemented by **server-side validation** to ensure security and prevent malicious data manipulation.

- Utilizing regular expressions and input validation techniques: Regular expressions and input validation techniques **help enforce data formatting rules** on the client-side, ensuring that user input meets expected criteria (e.g., email formats, numeric values) before submission while improving usability and reducing server load.

- Preventing common attacks like Cross-Site Scripting (XSS) and SQL Injection Proper data validation **prevents attacks like Cross-Site Scripting (XSS) and SQL Injection** by sanitizing user input, restricting unexpected characters, and ensuring that only safe and properly formatted data is processed by the application.

Logging and Alerting:

Logging and alerting mechanisms play a critical role in detecting and responding to potential threats and attacks. The importance of logging and real-time alerting:

The significance of robust logging in detecting suspicious activities: Robust logging helps detect suspicious activities by tracking user actions, identifying security threats, and enabling quick responses to potential attacks.

Types of logs: access logs, error logs, and security logs:

Access Logs – Track login attempts, user sessions, and IP addresses.

Error Logs – Record failed authentication, API errors, and invalid requests.

Security Logs – Log SQL injection attempts, XSS payloads, and privilege escalation attempts**.**

Example: A financial app should log all login attempts and send an alert if multiple failed attempts occur from an unknown IP address.

Input Validation

Input validation is the process of ensuring that all data entered by users into a web application is **correct, properly formatted, and secure** before processing it. This step is crucial in preventing **malicious inputs.**

NEVER TRUST THE USER

Input Validation Techniques

- Blacklisting (Less Secure)
- Whitelisting (Preferred Method)
- Defense-in-Depth (Combined approach)

Blacklist validation –

Blacklist validation (also called **negative validation**) is a security technique where **specific disallowed characters, words, or patterns are blocked** in user input. Instead of defining what is **allowed** (whitelisting), blacklist validation attempts to filter out known **malicious inputs**.

- Reject known bad inputs.

Disadvantage: Attackers can bypass blacklists by encoding, obfuscating, or using alternative characters.

Alternative: Whitelisting (positive validation) is preferred because it explicitly defines allowed input formats.

Example – For username and password – don't allow characters like ', \", ;, --, OR , AND.

This might miss some characters.

Whitelist validation –

Whitelist validation (also called positive validation) is a security technique where only pre-approved characters, patterns, or formats are allowed in user input. Instead of trying to block bad input (blacklisting), whitelisting ensures that only expected input is accepted, making it much more secure.

- Accept known good inputs

Advantages -
Stronger security (prevents unexpected input)
Harder for attackers to bypass
Reduces risks of SQL injection, XSS, and other attacks

Disadvantage:
May reject legitimate input if not carefully defined
Needs frequent updates if new valid formats are required

Example – For username and password –allow only characters like a-z,A-Z,0-9,+,$.

Validation Type	Approach	Security Strength	Example
Whitelist (Recommended)	Allows **only pre-approved input**	Stronger security	Allow **only letters & numbers** in usernames
Blacklist (Risky)	Blocks **known bad inputs**	Weak (can be bypassed)	Block SELECT, DROP, OR 1=1 in SQL

Defense in Depth (DiD)

Defense- in-Depth is a multi-layered security strategy where multiple security measures are implemented at different levels to protect systems from cyber threats. The goal is to ensure that even if one layer fails, other layers still provide security. This approach minimizes vulnerabilities and reduces the chances of a successful attack.

- Prevents single points of failure
- Slows down attackers, increasing detection chances
- Provides redundancy in security controls

Attack Surface Reduction

Attack Surface Reduction (ASR) is a security strategy aimed at minimizing the number of ways an attacker can exploit a system. The attack surface refers to all the entry points where an attacker could attempt to gain unauthorized access, manipulate data, or execute malicious activities.

Importance of Attack Surface Reduction -

- Reduces vulnerabilities that hackers can exploit
- Limits system complexity, making security management easier
- Improves overall security posture by eliminating unnecessary risks

Rules of Thumb for Attack Surface Reduction

Security experts follow certain best practices (rules of thumb) to effectively reduce the attack surface.

1) **Disable Unnecessary Features & Services**

   More features = More vulnerabilities

   Practices followed -
   - Remove unused plugins, APIs, or ports
   - Disable old/insecure protocols (e.g., SMBv1, Telnet)

2) **Implement the Principle of Least Privilege (PoLP)**

   Overprivileged accounts can be exploited
   Practices followed –
   - Grant only the **minimum permissions** users or services need
   - Restrict **admin rights** to essential personnel

3) **Reduce Code Complexity & Attackable Codebase**

   Large, complex applications contain more vulnerabilities
   Practices followed –

   - Remove **unused code** or features
   - Keep **third-party dependencies up to date**

4) **Minimize Entry Points & Public Exposure**

   More exposed endpoints = More attack opportunities
   Practices followed –
   - Restrict **public access** to admin panels and APIs
   - Use **firewalls and allowlisting** to limit access.

5) **Regularly Patch & Update Software**

Outdated software contains **known vulnerabilities**
Practices followed –

- Apply **security patches** as soon as they're available
- Use **automated updates** when possible

Classifying and Prioritizing Threats

All security vulnerabilities are equally serious. If there's even a slightest chance of attack, every single possible vulnerability has to be eliminated from the code. Some popular ways of categorizing threats –

STRIDE

IIMF

CIA

CWE

DREAD

CVSS

STRIDE

STRIDE is a threat classification system originally designed by Microsoft security engineers. STRIDE does not attempt to rank or prioritize vulnerabilities; it is just a system to classify vulnerabilities according to their potential effects.

STRIDE is an acronym, standing for:

• Spoofing

• Tampering

• Repudiation

• Information Disclosure

• Denial of Service

• Elevation of Privilege

Spoofing vulnerabilities allow an attacker to claim to be someone they're not, or to assume another user's identity. Spoofing attacks are done by impersonating users by obtaining authentication tokens, often through cross-site scripting (XSS) or sniffing unencrypted Wi-Fi traffic. For example, if an attacker steals your bank login credentials, they can pretend to be you. This can lead to identity theft, fraud, or unauthorized transactions. Spoofing is a major concern in web security, often mitigated by using encryption, secure authentication, and input validation.

Tampering is unauthorized modification of data to alter its integrity.

- Example:
  - An attacker exploits SQL injection to modify product prices in an e-commerce database.
  - Malware modifies system files to introduce backdoors.

Mitigation: Use cryptographic hashing, digital signatures, and input validation.

Repudiation is the ability of a user to deny an action without evidence to prove otherwise.

- Example:
  - A customer denies making a fraudulent online purchase, and there's no audit log to verify it.
  - A hacker erases log files after gaining unauthorized access.

Mitigation: Use secure logging, digital signatures, and non-repudiation mechanisms (e.g., blockchain, signed receipts).

Information Disclosure is Unauthorized access to confidential or sensitive information.

- Example:
  - A website exposes credit card numbers due to improper access control.
  - An attacker sniffs unencrypted Wi-Fi traffic to steal login credentials.

Mitigation: Encrypt data (TLS, AES), implement proper access controls, and avoid verbose error messages.

Denial of Service is Disrupting system availability by overwhelming resources.

- Example:
  - A DDoS attack floods a server with traffic, making a website inaccessible.
  - An attacker locks user accounts by repeatedly entering incorrect passwords.

Mitigation: Use rate limiting, firewalls, CAPTCHAs, and load balancers.

Elevation of Privilege is Gaining unauthorized access to higher-level privileges.

- Example:
  - A standard user exploits a buffer overflow vulnerability to gain admin rights.
  - A web application allows unvalidated privilege changes, letting an attacker become an administrator.

Mitigation: Implement least privilege access, enforce strong authentication, and use secure coding practices.

The **IIMF model** is a simplified alternative to **STRIDE**, categorizing vulnerabilities into **Interception, Interruption, Modification, and Fabrication**.

- **Interception** = Information disclosure (unauthorized data access).
- **Interruption** = Denial-of-service (preventing system access).
- **Modification & Fabrication** = Tampering (changing or creating data).

Spoofing & Elevation of Privilege are types of privilege escalation, and Repudiation falls under Modification (altering logs). Both IIMF and STRIDE effectively classify threats.

CIA

The CIA triad—Confidentiality, Integrity, and Availability—defines the core principles of cybersecurity. While IIMF describes how attackers compromise security, CIA represents the system attributes we aim to protect.

Confidentiality ensures sensitive data is accessible only to authorized users. For example, your home address should be protected from unauthorized access, like hackers or unnecessary third parties. Attackers use interception to break confidentiality.

Integrity prevents unauthorized modification or fabrication of data. This means attackers cannot alter a product's price in an e-commerce site or create fraudulent records. Tampering and fabrication are the primary threats to integrity.

Availability ensures systems remain operational and responsive. Denial-of-service (DoS) attacks disrupt availability, affecting businesses relying on cloud services or SaaS applications. Companies must trust external service providers, similar to trusting an airline over personal driving.

Additional security concepts include:

- Authenticity – Verifying users and processes to prevent identity fraud.
- Nonrepudiation – Preventing users from denying their actions, ensuring accountability in transactions.

The CIA model is fundamental for designing secure systems, protecting against threats that compromise data security and operational reliability.

Common Weakness Enumeration (CWE)

The Common Weakness Enumeration (CWE) is a categorized list of software vulnerabilities maintained by MITRE Corporation. Examples include:

- SQL Injection (CWE-89)
- Buffer Overflow (CWE-120)
- Missing encryption of sensitive data (CWE-311)
- Cross-Site Request Forgery (CWE-352)
- Weak Cryptographic Algorithms (CWE-327)
- Integer overflow(CWE – 190)

CWE is more specific than security models like STRIDE or IIMF and is similar to the OWASP Top Ten. MITRE, along with the SANS Institute, releases an annual Top 25 Most Dangerous CWE list.

A key benefit of CWE is its vendor-neutral classification system, allowing security professionals to communicate vulnerabilities effectively. For example, CWE-759 signifies a cryptographic hash issue lacking a proper salt. The CWE website (cwe.mitre.org) provides guidance on identifying and mitigating these weaknesses.

CWE differs from CVE (Common Vulnerabilities and Exposures), which focuses on specific security flaws in particular products. For instance, CVE-2011-2883 describes a vulnerability in Citrix Access Gateway's ActiveX control, enabling man-in-the-middle attacks. CWE, in contrast, provides a broader taxonomy of security weaknesses across various technologies.

DREAD

DREAD is a threat-ranking system developed by Microsoft to assess security risks. Unlike STRIDE, which classifies threats, DREAD assigns risk scores based on five factors:

1. Damage Potential – Measures how severe an attack's impact would be (e.g., minor slowdown vs. complete data breach).
2. Reproducibility – Evaluates how consistently an attack can be repeated (e.g., always successful vs. rarely works).
3. Exploitability – Assesses how easy an attack is to execute (e.g., requires basic hacking tools vs. advanced social engineering).
4. Affected Users – Determines the number of impacted users (e.g., an entire site vs. only logged-in users).

5. Discoverability – Estimates how easily an attacker can find the vulnerability (e.g., credentials in HTML vs. obscure flaws).

Each factor is scored from 1 to 10, then averaged to determine the overall DREAD score.

Criticism of DREAD

- Equal weight issue – A low-damage but highly exploitable attack can score as high as a devastating but hard-to-find attack.
- Discoverability flaw – Unlike other factors that assume the worst, discoverability assumes uncertainty, reducing reliability.

Due to these limitations, many security teams no longer use DREAD for risk assessment, preferring models like CVSS (Common Vulnerability Scoring System).

Common Vulnerability Scoring System (CVSS)

A more commonly used metric for rating vulnerabilities is the Common Vulnerability Scoring System, or CVSS. CVSS is an open standard, originally created by a consortium of software vendors and nonprofit security organizations, including:
• Carnegie Mellon University's Computer Emergency Response Team Coordination Center (CERT/CC)
• Cisco
• U.S. Department of Homeland Security (DHS)/MITRE
• eBay
• IBM Internet Security Systems
• Microsoft
• Qualys
• Symantec

It provides an objective and standardized method to assess vulnerabilities, with scores ranging from 0 to 10, where 10 represents the most severe threats.

CVSS Components

A CVSS score is based on three main parts:

1. Base Score (Primary metric) – Evaluates the inherent characteristics of a vulnerability, considering:
    - Attack vector (local vs. network-based).
    - Authentication requirements.
    - Impact on confidentiality, integrity, and availability (CIA).

2. Temporal Score (Changes over time) – Adjusts based on:
    - Availability of exploits.
    - Vendor fixes or workarounds.
    - Credibility of the vulnerability report.
3. Environmental Score (Organization-specific) – Modifies risk based on:
    - Financial, physical, or safety impact.
    - How many systems are affected.

Unlike DREAD, CVSS scores are more structured and weighted, ensuring consistent assessments across different analysts. Many organizations rely on the Base Score, but Temporal and Environmental Scores help refine prioritization.