CODING OUTPUT

2022

UNIVERSITY OF ESSEX ONLINE | SSD\_PCOM7E NOVEMBER 2022

[Coding Output 3](#_Toc122316864)

[READ ME 3](#_Toc122316865)

[Module Assignment 2: Archive CMS 3](#_Toc122316866)

[Summary of the code 3](#_Toc122316867)

[Summary of Limitations 3](#_Toc122316868)

[Differences between the design and the final code 4](#_Toc122316869)

[Usage 5](#_Toc122316870)

[Setup and Install Guide 6](#_Toc122316871)

[Install.sh 8](#_Toc122316872)

[Env.txt 8](#_Toc122316873)

[Login4.py 8](#_Toc122316874)

[Fileupload.py 12](#_Toc122316875)

[DATABASES 14](#_Toc122316876)

[db\_husers.txt 14](#_Toc122316877)

[db\_secret.txt 15](#_Toc122316878)

[SCAN 16](#_Toc122316879)

[Login4\_scan.txt 16](#_Toc122316880)

[Fileupload\_scan.txt 17](#_Toc122316881)

[References 21](#_Toc122316882)

# Coding Output

## READ ME

### Module Assignment 2: Archive CMS

A Content Management Platform aimed to centralize the file management for ‘the Dutch Police Internet Forensics (Government of the Netherlands, n.d.).` and enable them to maintain files in the system as per GDPR requirements.

Note: Due to lack of coding experience and expertise the current implementation only functions on the CLI [command line interface]

#### Summary of the code

* login4.py: Simulates the logon function with sufficient password complexity checks.
* fileupload.py: Simulates the file upload with file upload restrictions to certain allowed file extensions, size, content-type and copies the successfully uploaded file to simulate a file upload.

#### Summary of Limitations

Our biggest limitation was the lack of experience with Django and expertise to code in django where we decided to move back to a CLI functional code base.

* Limitations of login4.py: Because this is a CLI only code. Checks such as csrf-protection, session management, brute-force protection are not deployed.
* Limitations of fileupload.py: Because this is a CLI only code code checks such as csrf-protection and other form-field content checks are not deployed.

#### Differences between the design and the final code

Originally, group 5 was very ambitious with the development of a secure Content Management Platform with ticketing system. In the original submission, the application had those target features:

* Logging
* MFA
* Creation of files
* Upload files
* Download files
* Read files
* Write files
* Delete files
* Upload events
* Ticketing system to assign events
* Ticketing system with a view on event status

The design plan was conceptualised with security in mind, as many approaches were considered to ensure the development of a secure software. Amongst the main security methods selected, there were:

* Authentication method
* Password complexity
* Multi-Factor Authentication (MFA)
* DMZ, application, and data boundaries
* The use of HTTPS and TCP
* Encryption and decryption via hashing and/or salting,
* The use of Django Framework which as many secure development patterns integrate

Although the design concept was complete, the final code does not reflect the attempted desire to reproduce a working application. Inflicted by the absence of a teammate and the lack of experience in coding in 75% of the team has led the final code to be minimalistic. Therefore, the software has two main working functionalities: logon and file upload. Even though it is not a lot of code compared to the original design concept, there have been security methods put in place to ensure secure functionalities, hence possible secure software. To securely protect the login function, a password complexity has been developed, as well as a multi-factor authentication with a Time-based One-Time Password. The upload files functionality also has secure development method to prevent vulnerabilities to be exploited, amongst them, there are file extensions, size, and content-type restrictions.

#### Usage

Since this is a Proof-Of-Concept [POC] code and not a fully functional application;

* Run the login4.py file as `python3 login4.py'
* Supply the user credentials as below [any one]

#| ID | Password || ----------- | ----------- || AAdmin | Staging@2022 || AMgr | Uploader@2022 |

Note password is hashed with sha256 in the database

* If the submitted password is correct, the application will wait at a prompt requesting a 6 Digit OTP code.
* At this point add dummy secret to google authenticator or 1password generated in the `db\_secret.txt` section, using the [+] icon in the app.
* Submit the `OTP` generated on Google Authenticator.
* Press authenticates

|  |  |
| --- | --- |
| ID | Secret |
| AAdmin | PAM7X3XELZJDMB5B7QJZRBZVBXF3JHOS |
| AMgr | BDBWALE3CEK4RL35EOMQ34GHFBJ2IXXS |

#### Setup and Install Guide

To setup the project please run through the following steps; [You can also copy paste the commands as it is for simplicity]

Copy the Final.zip to a Ubuntu-system for installation:

Unzip the archive using:

*unzip Final.zip*

Move into the newly created Final directory:

*cd Final/*

Install dependancies using requirements.txt:

*sudo pip3 install -r requirements.txt*

Copy Environment variables and export to terminal:

*export DB\_USER=projroot*

*export DB\_PASSWORD=New@2022*

*export DB\_HOST=localhost*

*export DB\_NAME=pycli*

Setup your mysql-server with the root password as you require and run the following

---Create user for the project---

*CREATE USER 'projroot'@'localhost' IDENTIFIED WITH mysql\_native\_password BY 'New@2022';*

---Create database for the project---

*CREATE DATABASE pycli;*

*GRANT ALL ON pycli.\* TO 'projroot'@'localhost';*

*use DATABASE pycli;*

---Create tables for the project---

*CREATE TABLE secret (*

*id INT NOT NULL AUTO\_INCREMENT PRIMARY KEY,*

*username VARCHAR(255) NOT NULL UNIQUE,*

*secret VARCHAR(32) NOT NULL*

*);*

*CREATE TABLE husers (*

*id INT NOT NULL AUTO\_INCREMENT PRIMARY KEY,*

*name VARCHAR(255) NOT NULL,*

*username VARCHAR(255) NOT NULL UNIQUE,*

*password VCHAR(255) NOT NULL*

);

---Create test data for the project---

*INSERT INTO husers (username, password) VALUES*

*('AAdmin', SHA2('Staging@2022', 256)),*

*('AMgr', SHA2('Uploader@2022', 256));*

*INSERT INTO secret (username, secret) VALUES ('AAdmin', 'PAM7X3XELZJDMB5B7QJZRBZVBXF3JHOS');*

*INSERT INTO secret (username, secret) VALUES ('AMgr', 'BDBWALE3CEK4RL35EOMQ34GHFBJ2IXXS');*

Please Note [POC only setup]:

* Database is set with a non-secure simple password as `New@2022`
* The current implementation does not allow creation of users unless added directly to the database
* Secret generated for Authenticator is not converted and served as a QR code.
* Generated secret length is short to keep the manual efforts minimal. [Making it Insecure by nature]

## Install.sh

*apt-get update;*

*apt-get install python3*

*apt-get install python3-pip*

*apt-get install mysql-server*

## Env.txt

ENV to export into the system.

This can be hardcoded by setting it in the /etc/environment file

*export DB\_USER=projroot*

*export DB\_PASSWORD=New@2022*

*export DB\_HOST=localhost*

*export DB\_NAME=pycli*

## Login4.py

*import mysql.connector*

*import re*

*import os*

*import pyotp*

*import hashlib*

*import sys*

Read the command line arguments

This input simluates the login fields of a form

*username = input("Enter Your Username: ")*

*password = input("Enter Your Password: ")*

Check if credentials are valid

This prevents null passwords and IDs to be submitted into the code

*if username == '' or password == '':*

*print("Please Enter Valid Credentials!!")*

*exit()*

Check the complexity of passwords

Function to check if password meets the required criteria

This prevents user to choose a ‘weak password’ and forces constraints on the chosen passwords

*def check\_password\_criteria(password):*

Check if password has 1 uppercase, 1 lowercase, 1 special character, and minimum 8 characters

*if not any(char.isdigit() for char in password):*

*return False*

*if not any(char.isupper() for char in password):*

*return False*

*if not any(char.islower() for char in password):*

*return False*

*if not any(char in set(r'[~!@#$%^&\*()\_+{}":;\']+$') for char in password):*

*return False*

*if len(password) < 8:*

*return False*

*return True*

Check if the password respects the criteria

The above function gets called here; verifies even the submitted password follows password complexity requirements.

*if check\_password\_criteria(password) == False:*

*print("Password Entered does not match the required critera!!")*

*exit()*

Connect to the database using the environment variables

This prevents hard coding credentials into the file allowing anyone to read it if the file is left accessible.

*cnx = mysql.connector.connect(user=os.environ['DB\_USER'], password=os.environ['DB\_PASSWORD'],*

*host=os.environ['DB\_HOST'],*

*database=os.environ['DB\_NAME'])*

Check if the credentials are valid

This query in the database fetches the password against a valid username

*query = "SELECT password FROM husers WHERE username = %s"*

*cursor = cnx.cursor()*

*cursor.execute(query, (username,))*

*hashed\_password = cursor.fetchone()[0]*

This query in the database verifies password against a valid username and if the submitted password is valid. The hashlib of sha256 is used as our stored password is hashed with the same. The library hashes the submitted input with sha256 and attempts to match it to the 'hashed\_password' fetched earlier.

This protects data at rest and prevents the conversion of hash back into the original key.

*if hashlib.sha256(password.encode()).hexdigest() == hashed\_password:*

*print("Login On Verified, Success!")*

If the credentials are correct up to this point a new 6-digit code is prompted by the system.

This prevents some forms of identity theft by making it impossible to capture credentials and use it a second time

*code = input("Please Enter your 6 Digit OTP Code: ")*

This query fetches the user's secret key from the database

*query = "SELECT secret FROM secret WHERE username = %s"*

*cursor.execute(query, (username,))*

*secret\_key = cursor.fetchone()[0]*

Check if the totp is valid

The fetched secret key is passed into pyOTP's totp function to verify the code submitted by the user by regenerating the same value.

*totp = pyotp.TOTP(secret\_key)*

*if totp.verify(code):*

*print("Login Successful; Welcome to Archive CMS Platform!")*

*else:*

*print("Failure! Invalid Code Please Try Again!")*

*else:*

*print("Failure Invalid Logon Credentials, Please Try Again!!")*

Close the connection to the database

*cnx.close()*

## Fileupload.py

*import sys*

*import os*

*import magic*

*import shutil*

Get the file path from the command line argument

[ This simulates the file upload as it would be on from a frontend form ]

*file\_path = input("Enter the file path: ")*

Check if the file has a valid extension

Validates if the uploaded file has the valid extensions against an allowed-list of pdf and docx.

This prevents other content extensions to be passed into the system and creating malformed files

*valid\_extensions = ['.pdf', '.docx']*

*\_, file\_extension = os.path.splitext(file\_path)*

*if file\_extension not in valid\_extensions:*

*print("Invalid file! The application only accepts '.pdf' or '.docx' extensions")*

*sys.exit(1)*

Check if the file has the correct content type

Validates if the uploaded file has the required content-type against an allowed-list of pdf and docx.

This prevents other content types to be passed into the system and creating malformed files

*mime = magic.Magic(mime=True)*

*content\_type = mime.from\_file(file\_path)*

*if file\_extension == '.pdf' and content\_type != 'application/pdf':*

*print("Invalid content type for PDF file!")*

*sys.exit(1)*

*elif file\_extension == '.docx' and content\_type != 'application/vnd.openxmlformats-officedocument.wordprocessingml.document':*

*print("Invalid content type for DOCX file!")*

*sys.exit(1)*

Check if the file is larger than 10 MB

Validates if the uploaded file has is within the file limit.

This prevents huge files to enter the systems [such as zip bombs] and overload and potentially DOS the system

*file\_size = os.path.getsize(file\_path)*

*if file\_size > 10 \* 1024 \* 1024:*

*print("File is too large! Reduce the file size to under 10 Megabytes")*

*sys.exit(1)*

Check if the file has multiple extensions

Validates if the file only is allowing a single extension. multiple extensions could potentially allow files to bypass the allow list

This prevents .pdf.php or .docx.py or similar malformed files to enter the system.

*if '.' in file\_path.split(os.path.extsep, 1)[1]:*

*print("Multiple extensions are prohibited!")*

*sys.exit(1)*

This portion of the app simulates that the file is uploaded

Given this is a CLI only function code /tmp/UploadedFiles directory simulates the secured storage of the system.

This prevents unauthorized file access and would be locked down for a specific user.

*upload\_dir = '/tmp/UploadedFiles'*

*if not os.path.exists(upload\_dir):*

*os.makedirs(upload\_dir)*

*shutil.copy(file\_path, upload\_dir)*

*print("Success! The file has been uploaded to our system")*

## DATABASES

## db\_husers.txt

SQL Schema to create a table called 'husers' in the database [currently set to pycli]

*CREATE TABLE husers (*

*id INT AUTO\_INCREMENT PRIMARY KEY,*

*username VARCHAR(255) NOT NULL,*

*password VARCHAR(255) NOT NULL*

*);*

SQL Queries to insert test data for username and password (hashed with SHA256) into the table called 'husers' in the database [currently set to pycli]

*INSERT INTO husers (username, password) VALUES*

*('AAdmin', SHA2('Staging@2022', 256)),*

*('AMgr', SHA2('Uploader@2022', 256));*

## db\_secret.txt

SQL Schema to create a table called 'secret' in the database [currently set to pycli]

*CREATE TABLE secret (*

*id INT NOT NULL AUTO\_INCREMENT PRIMARY KEY,*

*username VARCHAR(255) NOT NULL UNIQUE,*

*secret VARCHAR(32) NOT NULL*

*);*

SQL Queries to insert test data for the test users in the database [currently set to pycli]

*INSERT INTO secret (username, secret) VALUES ('AAdmin', 'PAM7X3XELZJDMB5B7QJZRBZVBXF3JHOS');*

*INSERT INTO secret (username, secret) VALUES ('AMgr', 'BDBWALE3CEK4RL35EOMQ34GHFBJ2IXXS');*

## SCAN

## Login4\_scan.txt

Run started:2022-12-19 08:29:28.144326

Test results:

>> Issue: [B322:blacklist] The input method in Python 2 will read from standard input, evaluate and run the resulting string as python source code. This is similar, though in many ways worse, then using eval. On Python 2, use raw\_input instead, input is safe in Python 3.

Severity: High Confidence: High

Location: login4.py:9

More Info: https://bandit.readthedocs.io/en/latest/blacklists/blacklist\_calls.html#b322-input

8 # Read the command line arguments

9 username = input("Enter Your Username: ")

10 password = input("Enter Your Password: ")

--------------------------------------------------

>> Issue: [B322:blacklist] The input method in Python 2 will read from standard input, evaluate and run the resulting string as python source code. This is similar, though in many ways worse, then using eval. On Python 2, use raw\_input instead, input is safe in Python 3.

Severity: High Confidence: High

Location: login4.py:10

More Info: https://bandit.readthedocs.io/en/latest/blacklists/blacklist\_calls.html#b322-input

9 username = input("Enter Your Username: ")

10 password = input("Enter Your Password: ")

11

--------------------------------------------------

>> Issue: [B322:blacklist] The input method in Python 2 will read from standard input, evaluate and run the resulting string as python source code. This is similar, though in many ways worse, then using eval. On Python 2, use raw\_input instead, input is safe in Python 3.

Severity: High Confidence: High

Location: login4.py:47

More Info: https://bandit.readthedocs.io/en/latest/blacklists/blacklist\_calls.html#b322-input

46 print("Login On Verified, Success!")

47 code = input("Please Enter your 6 Digit OTP Code: ")

48 # Get the user's secret key from the database

--------------------------------------------------

Code scanned:

Total lines of code: 47

Total lines skipped (#nosec): 0

Run metrics:

Total issues (by severity):

Undefined: 0.0

Low: 0.0

Medium: 0.0

High: 3.0

Total issues (by confidence):

Undefined: 0.0

Low: 0.0

Medium: 0.0

High: 3.0

Files skipped (0):

## Fileupload\_scan.txt

Run started:2022-12-19 08:29:53.620747

Test results:

>> Issue: [B322:blacklist] The input method in Python 2 will read from standard input, evaluate and run the resulting string as python source code. This is similar, though in many ways worse, then using eval. On Python 2, use raw\_input instead, input is safe in Python 3.

Severity: High Confidence: High

Location: fileupload.py:7

More Info: https://bandit.readthedocs.io/en/latest/blacklists/blacklist\_calls.html#b322-input

6 # Get the file path from the command line argument

7 file\_path = input("Enter the file path: ")

8

--------------------------------------------------

Code scanned:

Total lines of code: 30

Total lines skipped (#nosec): 0

Run metrics:

Total issues (by severity):

Undefined: 0.0

Low: 0.0

Medium: 0.0

High: 1.0

Total issues (by confidence):

Undefined: 0.0

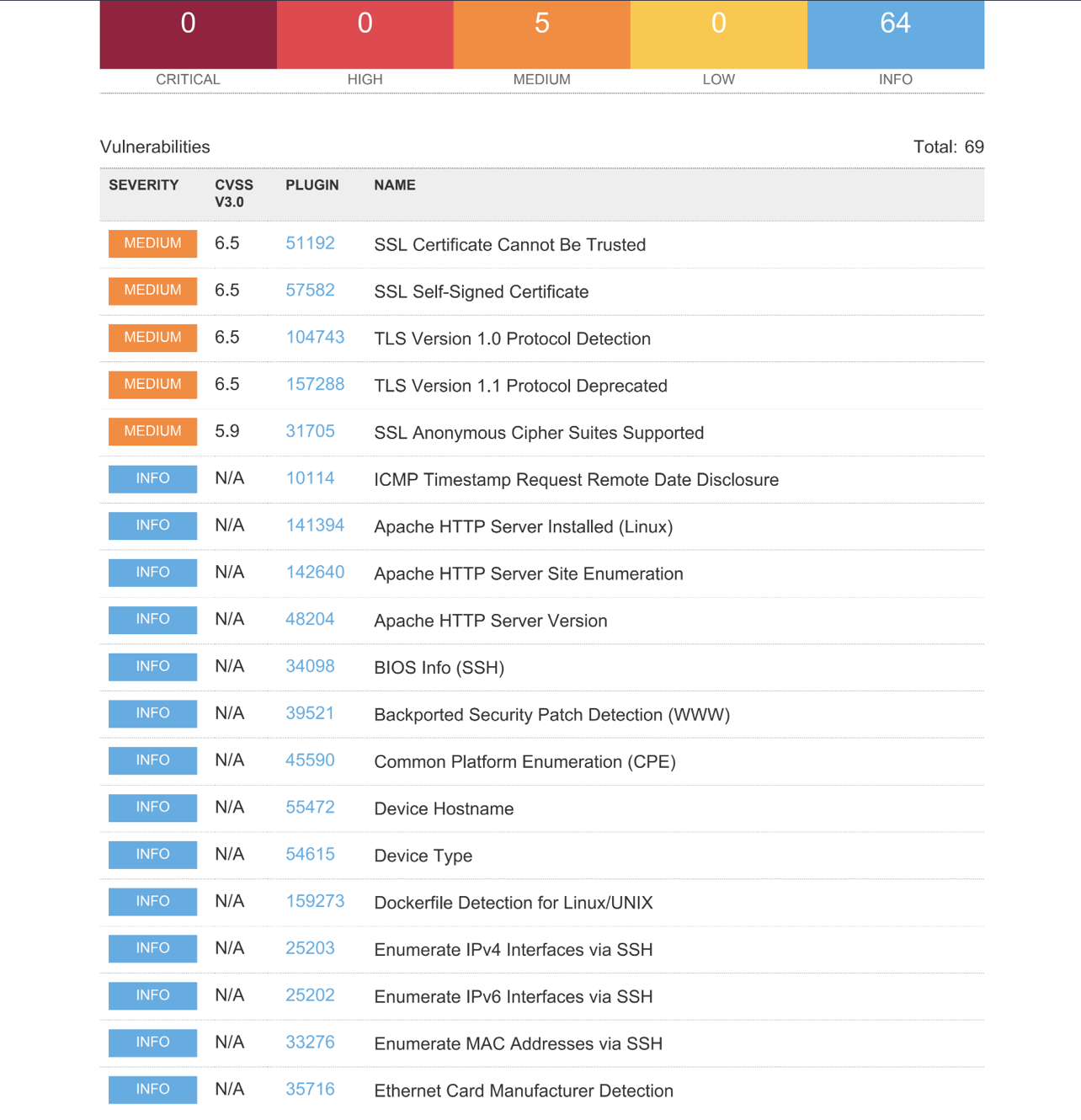
Low: 0.0

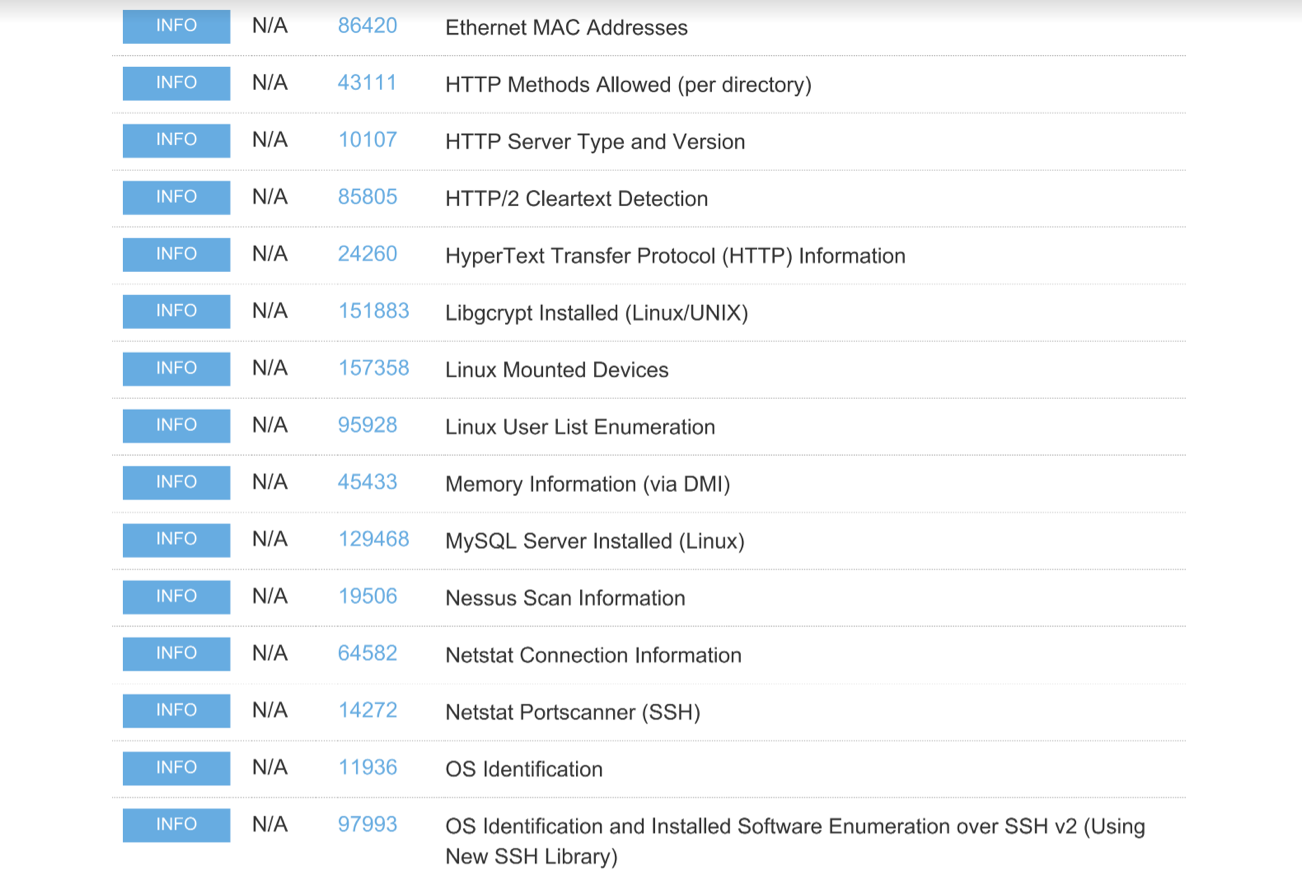
Medium: 0.0

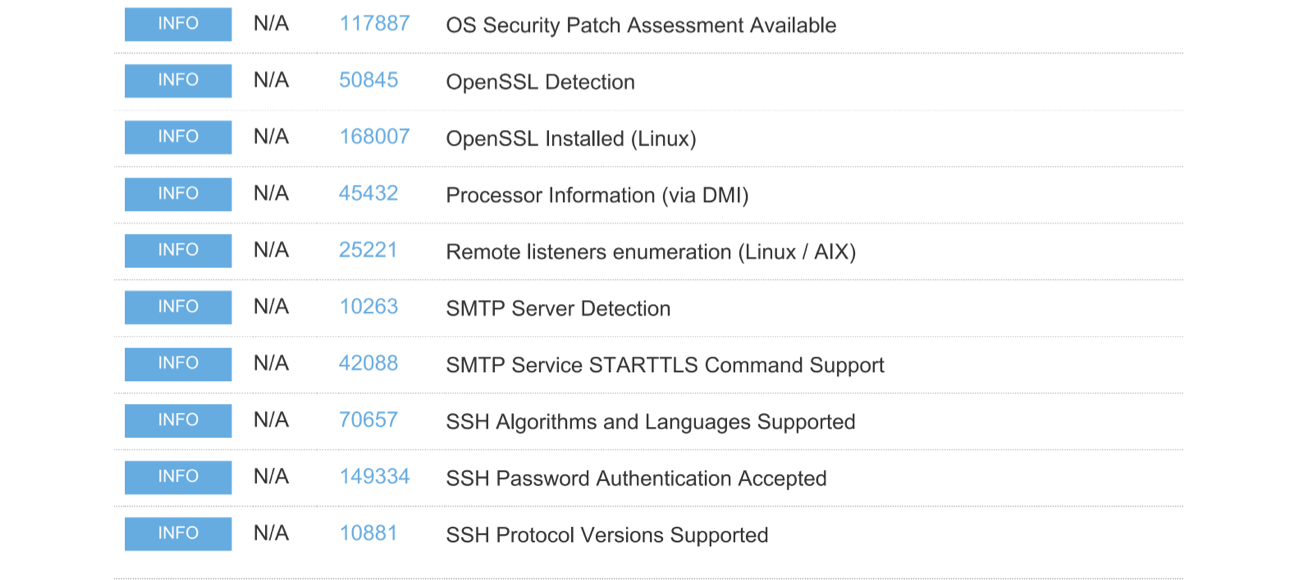
High: 1.0

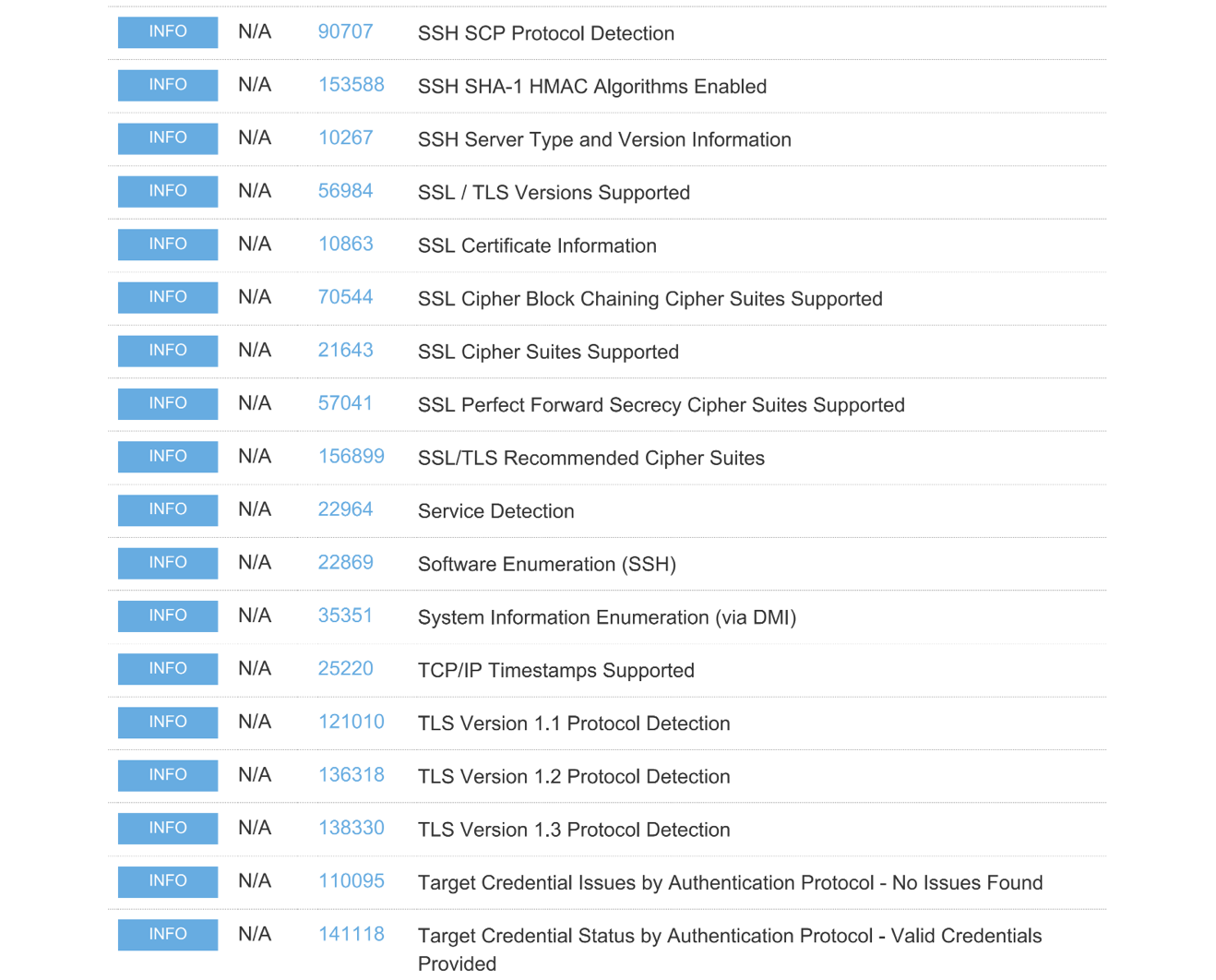
Files skipped (0):

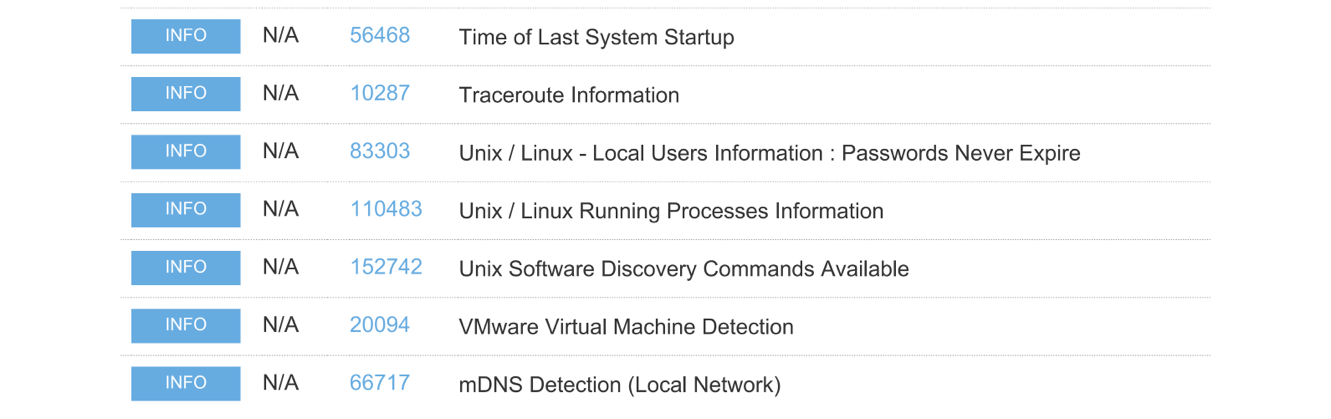
Nessus











## References

Able. (n.d.). How to Set and Get Environment Variables in Python. Available from: https://able.bio/rhett/how-to-set-and-get-environment-variables-in-python--274rgt5 [Accessed 18 Dec. 2022].

Chapple, M; Steward, J. M.; and Gibson, D (2021). CISSP Certified Information Systems Security Professional – Official Study Guide – 9th Edition, *John Wiley & Sons, Inc,* 1250 pages.

Prakash, Y. (2021). The Quick Guide To Using Environment Variables in Python. Medium. Available from: https://towardsdatascience.com/the-quick-guide-to-using-environment-variables-in-python-d4ec9291619e [Accessed 18 Dec. 2022].

Stack Overflow. (n.d.). Simple login function in Python. Available from: https://stackoverflow.com/questions/28483113/simple-login-function-in-python [Accessed 18 Dec. 2022].

STechies. (n.d.). How to Upload Files in Python? Available from: https://www.stechies.com/upload-files-python/ [Accessed 18 Dec. 2022].