Nessus Automation in Python

A comparison between API requests and Selenium.

Ryan Denholm

CMP320: Advanced Ethical Hacking – Scripting Assessment

2023/24

.

Note that Information contained in this document is for educational purposes.

.

Abstract

Information gathering is a key step in the process of conducting a penetration test as well as protecting a network. This step can be extremely time consuming with desirable information such as Host IP's and related vulnerabilities taking multiple scans to obtain and relate this information. Nessus provides an easy-to-use host discovery and vulnerability scanner, hosted on their website. This time commitment can be reduced with the implementation of automation using a python script.

Python's Request library is used for handling the generation and downloading of Nessus Reports as .csv . With the acquired .csv. Panda is utilized to extract information from the .csv to then be sorted by CVSS severity. The automation of scans through API is prohibited for Nessus Essentials, so Selenium was employed to circumvent this issue. Navigating from the login panel to creating a report and running it based on XPath's and HTML element ID's.

A comparison of methods used is provided, with relation to the goal of reducing the time spent by the user conducting information gathering. Each procedure is critically analysed on the basis of time efficiency and the potential to build upon this script is also detailed, with key improvements that will increase the efficiency of this script.

.

Contents

.

1 INTRODUCTION

1.1 BACKGROUND

As industries across the world continuously adopt new technologies, cyber security has become paramount for businesses of all sizes. New threats are developed just as quickly as the last one was patched (Sayegh, 2024), creating a troublesome situation for organizations and their network security. Vulnerability scanning is a vital aspect of any cyber security protocol, whether it be offensive or defensive, and it is an essential step to potentially uncovering threats to an organization's safety.

Introduction of the Technology:

A plethora of tools are available for any cyber security personnel to utilize, this report will focus on Tenable's Nessus Vulnerability Scanner. This software can be run on Linux, macOS and Windows, meaning that it is deployable on any organization's network. Like Nmap, Nessus uses Layer 3 Protocols such as: ICMP, TCP and UDP to extract information from target IP Addresses. Currently, Nmap holds a larger market share in comparison to Nessus (6sense, 2024), but this is mainly due to the cost factor, with Nmap being free and Nessus costing £4,252.75 (Tenable, 2024) per year.

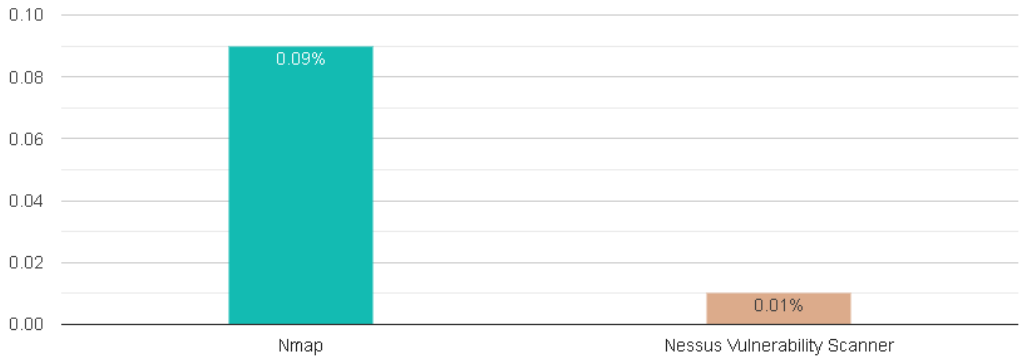


Figure 1: Nmap vs Nessus market share (6sense, 2024)

Despite that cost, Nessus Pro comes some notable benefits:

- Advanced Detection – 65,000 CVEs on their database, allowing for a more accurate scan.
- Customer Support – Helps with implementation and management of Nessus on the network.
- Nessus GUI - allows for automation through their API. Tenable provides an interactive API Guide, this streamlines the API process for paying customers.

Due to the cost of Nessus Pro, it is somewhat unfeasible for a student to obtain that license. Thus, limiting the Nessus' API capabilities to "read-only", provided the individual is using Nessus Essentials. This calls for a different approach for some of the Nessus functionalities that will be discussed later in this report.

Manual vs Automated Scans:

Traditional techniques used for vulnerability scanning are limited in several ways, making them potentially less effective than automated scans. Manual scans are reliant on an individual to create, execute and analyse the scan. Introducing this human element also introduces the risk of a less accurate scan due to human error:

- Missing Nmap Tags.
- Misreading/Misunderstanding Results

Automated scans are becoming increasingly prevalent in the cyber security sphere, this is due to one main benefit, time. Running a script to conduct the vulnerability scan enabling a cyber security professional to spend their time elsewhere. Allowing for a more efficient workflow and potentially a more secure network due to the availability of human resources. Automation of scans is not without its flaws. Due to the nature of the script construction, it can be difficult to implement a dynamic environment for the script to operate. Another issue can be the introduction of false positives or negatives (SecureLayer7, 2023).

1.2 AIM

The Nessus Automation script aims to reduce the time commitment from a user conducting information gathering, this will be achieved with the following implementation:

- Generate and Download Nessus scans as .csv using Nessus' API in conjunction with Python's Requests library.
- Remediate Nessus's scan results to remove some data, and sort by Common Vulnerability Scoring System (CVSS), from most severe to least using Panda.
- Automation of a Nessus scan via the implementation of Selenium to bypass API restrictions.

2 PROCEDURE

2.1 OVERVIEW OF PROCEDURE

The following section includes key steps undertaken to produce the Nessus Automation Script, including initialising the session with the Nessus Interface, requesting, and downloading a desired Nessus Report, both using the Nessus API. Demonstrating the use of Panda to modify the obtained .csv and saving it locally. Finally, showing implementation of Selenium to navigate and interact with the Nessus Interface. This has been used to create and launch an Advanced Network Scan. A flow chart is included to provide clarity in relation to the overall layout of the script, see ***Appendix A***.

Basic Python programming will not be commented on, interactions pertaining to API usage or Selenium, however, will be commented on. Repetitive code may be shown in multiple subsections but will not be commented on besides the initalisation of that code e.g. **Headers_perm**.

Easy formatting will be used:

- System commands: highlighted, bolden and italics e.g. # ***ifconfig***
- Directions: bold, arrows e.g. **File -> Save**
- Important information directly related to the previously provided code: Bold, Bar, explanation e.g. **print() |** prints data inside the parentheses.
- Notable information regarding the code but not necessarily covered by code snippets: underlined e.g. X-API Keys becomes X-Cookie

2.2 PRE-REQUISITES

Software	Version	Link/Install
Python3	3.12.3	https://www.python.org/downloads/release/python-3123/
Selenium	4.19	pip install Selenium
WebDriver-Manager	12.1.9	Pip install webdriver-manager
Tenable Nessus Essentials	10.7.2 (#29) LINUX (amd-64)	https://www.tenable.com/downloads/nessus?loginAttempted=true

2.3 SETTING UP ENVIRONMENT

This section includes the commands and steps taken to create the necessary environment for this script to work including:

- Starting the Nessus service
- Obtaining Access and Secret keys.

Assuming Kali Linux is on the appropriate network, Nessus should be intialised using the following command:

 # systemctl start nessusd

This will start the service and allow access to the Nessus interface at https://*kali ip address here*:**8834**. Once logged in, create API keys: **home page -> My Account -> API Keys -> Generate.** Store these keys securely, they are used in the next section.

2.4 INITIATING CONNECTION

The following are the necessary steps to establish a session with Nessus with the previously obtained API Keys, then storing the session token for later use. Python's Requests library handles the API interaction for this section, because of its ability to send HTTP GET and POST requests as well as easily configure their headers with the appropriate authentication method.

Start with importing the necessary libraries for this section as follows:

```
import requests
from urllib3.exceptions import InsecureRequestWarning
requests.packages.urllib3.disable_warnings(InsecureRequestWarning)
```

Figure 2: importing important libraries.

This imports Python's Requests library which is vital for API interaction. InsecureRequestWarning is raised when trying to make requests over HTTPS without proper SSL Certification. InsecureRequestWarning into .urllib3.disable_warnings is then passed to disable this obstacle.

The following code creates a session with Nessus.

```
headers_auth = {
    "accept": "application/json",
    "content-type": "application/json",
    "X-ApiKeys":
"accessKey=4a2725412945e523eac9bad1f1d51d1882c48bae83644610ad1b495978ea634c;secre
tKey=2195e79d8ee8060cbac7102736ab3e6bb50682d76855f61ed0c05df5315446f6"
}

# Creating payload with login credentials
payload_auth = {"username": username, "password": password}

# Authenticates the session with login credentials and APIKeys
response_login = requests.post(url+"/session", json=payload_auth,
headers=headers_auth, verify=False)
```

Figure 3: Session code snippet

Headers_auth |contains the HTTP headers that will be contained in the API login request, including:

- **Accept: application/json |** Informs the client to expect JSON responses from Nessus.
- **Content-type: application/json |** Sets the content type as JSON
- **X-ApiKeys: accessKey= *accesskey* , secretKey = secretkey |** Contains the API keys that are needed for authentication on Nessus, these keys need to be refreshed every time the Nessus service is started.

payload_auth | holds both the username and password that will be passed through the HTTP POST request.

response_login | stores the response from the server, regarding the authentication status.

request.post() | initiates the session with Nessus at the "url/session" endpoint. Passing the admin credential variables defined in the payload, and the API keys specified in the headers.

verify=False | argument is used to show that SSL verification is disabled.

For troubleshooting potential issues with establishing a session:

```
print(response_login.text)
```

Figure 4: Troubleshooting method.

The print() statement will return the response from the server, convert from JSON to text with the .text attribute and print the result in the terminal. This is key to troubleshooting any interaction with the API.

Likely error codes and their meaning:

- **Status: 400 |** username format is invalid
- **Status: 401 |** username or password is incorrect

Following from establishing the session, creating a new header variable is recommended for continuous use throughout the API interaction of this script.

```
header_perm = {
    "accept": "application/json",
    "content-type": "application/json",
    "X-Cookie": "token="+ response_login.json()["token"]
}
```

Figure 5: header perm showing different authentication method.

X-ApiKeys becomes X-Cookie because it is more efficient to use a token rather than passing two API Keys for each request. Tokens are also more secure as they are refreshed after each new session created, rather than each time the Nessus service is started.

2.5 DOWNLOADING NESSUS SCANS

This subsection details the steps to request and extract Nessus reports in the file format of .csv .

Starting with obtaining a list of all the current Nessus Scans that exist on the server:

```
response_scan = requests.get(url+"/scans", headers=header_perm, verify=False)
```

Figure 6: response scan syntax.

Note, folder_id can be implemented to specify a desired folder and last_modification_date can be used to limit the results to reports that have only been produced/edited since that date. Both of these parameters can be passed through a payload.

Once the user has determined the scan they want and it is present in the Nessus Server, the details of that scan are obtained, and exporting can commence:

```
response_details = requests.get(url+"/scans"+"/"+str(scan_id),
headers=header_perm, verify=False)
# Export Scan Results
# Basic Settings
export_format = "csv"
template_id = True

    # Payload for export request
    payload_export = {
        "format": export_format,
        "template_id" : template_id,
    }
    url_export = "https://192.168.0.241:8834/scans/"+str(scan_id)+"/export"

    # Send export request
    response_export = requests.post(url_export, json=payload_export,
headers=header_perm, verify=False)
```

Figure 7: exporting snippet.

format and **template_id** are both required parameters that must be passed to successfully initiate exportation of the report. **csv** is the desired file type for this script, but PDF, Nessus and HTML are also possible output files.

Possible Error responses and their meaning:

- **Status: 400 |** A required parameter is missing.
- **Status: 404 |** Designated scan does not exist.
- **Status: 409 |** Designated scan is currently running or is creating a report already.

Once the export request has been successfully submitted, create the download request:

```
# Create URL for download
```

```
url_download = "https://192.168.0.241:8834/scans/" + str(scan_id) + "/export/" +
str(file_url) + "/download"

# Initiate download
response_download = requests.get(url_download, headers=header_perm, verify=False)

# Check if download request was successful
if response_download.status_code == 200:
      # Get the file content
      file_content = response_download.content

      # Define the file path where you want to save the downloaded file
      file_path = r"C:\Users\ryant\Documents\Scripting\Report
"+scan_name+"."+export_format  # You can change the file name and path as needed

      # Write the content to a local file
      with open(file_path, "wb") as file:
          file.write(file_content)
```

Figure 8: creation of download request.

After creating the download url, initiating the request and validating the success. The file is stored locally for remediation in the next subsection.

Potential error code and its meaning:

- **Status: 404 |** File does not exist.

2.6 Remediation of Nessus Scans

This subsection demonstrates the implementation of Panda in the script.

```
    # Read the downloaded CSV file
    df = pd.read_csv(file_path)

    # Select only the required columns
    selected_columns = ["CVSS v2.0 Base Score", "Host", "Protocol", "Port",
"Name"]
    df_selected = df[selected_columns]
```

```
    # Sort the data by "CVSS v2.0" column from High to Low
    df_sorted = df_selected.sort_values(by="CVSS v2.0 Base Score",
ascending=False)

    # Define the file path for the sorted CSV file
    sorted_file_path = r"C:\Users\ryant\Documents\Scripting\Sorted_Report_" +
scan_name + ".csv"

    # Save the sorted data to a new CSV file
    df_sorted.to_csv(sorted_file_path, index=False)

    print("Sorted CSV file saved successfully at:", sorted_file_path)
```

Figure 9: Loading, Modifying and saving .csv using Panda.

The above code reads in the previously obtained .csv from Nessus, extracts the desired columns and then sorts them in descending order by CVSS. Once sorted, the file is then stored locally in the same directory as the script.

*Please see **Appendix B**, containing sample output data.*

2.7 Automating Advanced Network Scan

In this subsection, Selenium will be used to automatically conduct an Advanced Network Scan through the Nessus Interface. Selenium was chosen because of robust capabilities when interacting with different web browsers and their elements. This allows for the script to be used for different browsers, as long as the webdriver for that browser is correctly installed.

```
from selenium import webdriver
import selenium.webdriver.support.ui as ui
from selenium.webdriver.chrome.service import Service
from webdriver_manager.chrome import ChromeDriverManager
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.desired_capabilities import DesiredCapabilities
import os
```

Figure 10: importing essential modules for following code.

All of these imports play an important role in the proceeding section of the script, they are explained as follows:

From selenium import webdriver | Imports the module responsible for the automation of web browser interactions.

Import selenium.webdriver.support.ui as ui |adds utility functions.

From selenium.webdriver.chrome.service import Service | Imports the service class and allows for starting and stopping the ChromeDriver service.

From webdriver_manager.chrome import ChromeWebManager |This class automatically downloads and installs the correct version of ChromeDriver.

From selenium.webdriver.support.ui import WebDriverWait | Enables the script to wait for a certain condition to be met before proceeding.

From selenium.webdriver.common.by import By | Enables the ability to locate web page elements.

From selenium.webdriver.support import expected_conditions as EC | Provides predefined conditions to wait for.

From selenium.webdriver.common.desired_capabilities import DesiredCapabilities | This class mimics capabilities of a browser.

Import os | Provides operating system functionality, like reading and writing.

```
# Specify the path to the Chrome driver executable
chrome_driver_path = r"C:\Users\ryant\Documents\Scripting\chromedriver.exe"

# Initialize Chrome WebDriver with the specified driver path
service = Service(chrome_driver_path)
service.start()

# Allows for chrome webdriver options
chrome_options = webdriver.ChromeOptions()

# Specify the path to the Chrome browser executable
chrome_binary_path = r"C:\Program Files\Google\Chrome\Application\chrome.exe"

# Adds Chrome binary path to chrome options
chrome_options.binary_location = chrome_binary_path

# Ignores cert, allows for automation past the chrome warning
chrome_options.add_argument('--ignore-certificate-errors')
```

Figure 11: Configuration of ChromeDriver.

It is important to set the file path for the ChromeDriver.exe as well as the original Chrome.exe, as well as initiating the driver to run. From there, **"—ignore-certificate-errors"** is used, this is similar to the verify=False from the previous subsections and is used to bypass SSL warnings.

Once the variables are configured correctly, Selenium's WebDriver is used to connect to the Nessus Interface IP Address at **https:/192.168.0.241:8834/**

```
    # Pass the Chrome options to the Chrome driver
    driver = webdriver.Chrome(service=service, options=chrome_options)

    # Connects to Nessus URL
    driver.get("https://192.168.0.241:8834/")
    wait = ui.WebDriverWait(driver, 30)
```

Figure 12: Starts ChromeDriver instantce.

To ensure that connection is being established, print out the source code of the Nessus Interface using:

```
print(driver.page_source)
```

Figure 13: Troubleshooting method in Selenium.

Once connected to the website, the script will enter the username and password provided through the same variables set at the very start of the script.

```
    # Waits until input 1 field appears.
    username_input = wait.until(EC.visibility_of_element_located((By.XPATH,
"/html/body/div/form/div[1]/input")))
    # sends username variable to username_input
    username_input.send_keys(username)

    # Waits until input field 2 appears.
    password_input = wait.until(EC.visibility_of_element_located((By.XPATH,
"/html/body/div/form/div[2]/input")))
    # sends password variable to password_input
    password_input.send_keys(password)
```

Figure 14: script waiting for elements to load and then passes credentials.

~_input = wait.until(EC.visibility_of_element_located((By.XPATH, “”) | wait.until is provided by WebDriverWait and will not pass information to the input variable until the form with username and password are generated. By.XPATH is used to locate the element by its location within the source code.

~_input.send_keys | Passes variables “username” and “password” into their respective fields.

Proceeding through the “login” button, uses the same **wait.until** command but rather than using **.send_keys, log_in.click()** is used, to interact with the button element.

Due to the diversity of “**By.”**, a simpler method can be used to find the “new scan” tab on the website:

```
    new_scan = driver.find_element(By.ID, "new-scan")
    new_scan.click()
```

Figure 15: Example of using By.ID to search for “new-scan” element

This will display the HTML source code in the terminal for the user to inspect.

```
<html lang="en" style="--sidenav-toggle-bottom: 0rem; --pendo-resource-center-container: 4rem
; --layout-height: 867px;"><head>
        <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">
        <meta http-equiv="Content-Security-Policy" content="upgrade-insecure-requests; block-
all-mixed-content; form-action 'self'; frame-src https://store.tenable.com; default-src 'self
'; connect-src 'self' www.tenable.com; script-src 'self' www.tenable.com; img-src 'self' data
:; style-src 'self' www.tenable.com; object-src 'none'; base-uri 'self';">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <meta charset="utf-8">
        <title>Nessus</title>
        <link rel="stylesheet" href="nessus6.css?v=1711560262479" id="theme-link">
        <link rel="stylesheet" href="tenable_links.css?v=ac05d80f1e3731b79d12103cdf9367fc">
        <link rel="stylesheet" href="wizard_templates.css?v=11939be86ca24a4dbbe8f9b85f95e140"
>
        <!--[if lt IE 11]>
            <script>
                window.location = '/unsupported6.html';
            </script>
        <![endif]-->
        <script src="nessus6.js?v=1711560262479"></script>
        <script src="pendo-client.js"></script>
        <!--Resource-Script-->
    </head>
    <body>
```

Figure 16: Expected output of print(driver.page.source)

The same syntax is then used to interact with the “Advanced Scan”, Advanced Scan does not have an ID so XPATH was used yet again, to interact with the element directly. Please ensure that the XPATH is in reference to the clickable portion of the HTML **<a href>** and not <i class = “glyphicons template advanced”> or <h5 class=”title> Advanced Scan</h5>. The <i class> and <h5> are both used for styling adding an icon and header respectively, whereas <a href> provides a path to the next page responsible for configuring the Advanced Scan.

```
time_asmb = time.asctime(time.localtime(time.time())).split()
# Create scan name based on date and time of accessing "advanced scan"
scan_name = time_asmb[2]+'_'+time_asmb[1]+'_'+time_asmb[4]+'_'+time_asmb[3]
set_scan_name = wait.until(EC.visibility_of_element_located((By.XPATH,
"/html/body/section[3]/section[3]/section/form/div[1]/div/div/div[1]/section/div[
1]/div[1]/div[1]/div[1]/div/input")))

set_scan_name.send_keys(scan_name)
```

```
time.sleep(1)


upload_descp = driver.find_element(By.XPATH,
"/html/body/section[3]/section[3]/section/form/div[1]/div/div/div[1]/section/div[
1]/div[1]/div[1]/div[2]/div/textarea")
description = "This is a scan made by Nessus Automated Script"
upload_descp.send_keys(description)

time.sleep(1)

# Set targets, in this case 192.168.0.1/24
set_targets = driver.find_element(By.XPATH,
"/html/body/section[3]/section[3]/section/form/div[1]/div/div/div[1]/section/div[
1]/div[1]/div[1]/div[5]/div/textarea")
target = "192.168.0.1/24"
set_targets.send_keys(target)

time.sleep(2)

# Click the "save" button
save_settings = wait.until(EC.visibility_of_element_located((By.XPATH,
"/html/body/section[3]/section[3]/section/form/div[2]/span"))).click()

time.sleep(5)
```

Figure 17: usage of OS module, creating unique scan name and other parameters.

time_asmb | time.asctime returns a string containing the local time based on the OS time, time.time creates the current date, .split is then used to separate each data point into a list of strings.

scan_name | Takes time_asmb strings and formats them into: Day, Month, Year, Hour, Minutes, Seconds based on the local time of the machine.

set_scan_name | Interacts directly with the “scan name” field and inputs the previously created scan_name.

upload_desc | Inputs a desired description into the “description field” of the form.

set_targets | This script uses a predefined range of IP addresses and is hard coded for that, a .text file can be uploaded and iterated through for more complex networks.

time.sleep is used to see the iteration of each step, otherwise the inputs would be allocated too quickly to see where the code is potentially failing. Once success is ensured, these can be removed to speed up the script.

Optionally, it may be desired to separate manual and automated scans, thus the folder option can be used through interacting with that drop down menu.

```
launch_xpath =
'/html/body/section[3]/section[3]/section/div[2]/table/tbody/tr[@data-
name={}]/td[9]'.format(scan_name)
print(launch_xpath)
launch_scan = wait.until(EC.visibility_of_element_located((By.XPATH,
launch_xpath)))
time.sleep(2)
launch_scan.click()
```

Figure 18: usage of OS module, creating unique scan name and other parameters.

launch_xpath | was created in a separate variable so that the correct XPath can be obtained, relative to the most recent scan created. Taking the scan_name and passing it into [@data-name={"scan name"}]. With manual scans also being available, using the table hierarchy to determine where the element is inconsistent as there is no guarantee that the correct scan will be in that corresponding row.

print(launch_xpath) | is used to ensure that the XPath is generated correctly.

wait.until | Is vital to ensure that there are no errors with clicking the button, the scan launch attempt will not initiate until the element has been loaded on to the website.

*Please see **Appendix C**, containing the Nessus Report related to the automatically generated scan.*

3 DISCUSSION

3.1 GENERAL DISCUSSION

Efficiency and accuracy are essential in the Cyber Security sector, automation is emerging as a great tool to achieve both aspects (Vazquez, 2023). The provided procedure details the steps to streamline the vulnerability scanning process using Nessus' Vulnerability Scanner.

Handling authentication and implementation of tokens, rather than API Keys for maintaining a secure connection (Procedure 2.4). Tokens have a shorter lifespan and are considered more secure as they are easier to revoke. If a Token is used maliciously, a threat actor can only access the same information as the user that requested that Token. This was the deciding factor in using Tokens instead of API Keys for everything except initialising the session (Helton, 2023).

Requesting and Downloading Nessus Reports as .CSV using Nessus' API (Procedure, 2.5) proved to provide a negligible improvement in time efficiency, as user-input was required to download specific scans. Due to this factor, it is recommended that this task is either done automatically with Selenium after each new report or remains manual.

Remediation of .CSVs to extract desired data using Panda (Procedure, 2.6) yielded better results than the previously mentioned aim but still left much to be desired. The script generates a new .CSV containing more concise information about the target(s) and sorts them in descending order based on the CSVV 2.0 score. Improvements can be made to this portion of the script and will be discussed in Future Work.

Creating and executing an Advanced Network Scan was achieved by employing the use of Selenium (Procedure, 2.7). Selenium proved to be superior to Nessus API requests due to its flexibility and bypassing the restraints placed on Nessus Essentials. Although a scan was created and ran automatically, there are more utilities possible that are not covered by this script (See Future work). This portion of the script is considerably more time efficient than interacting with the Nessus API due to the lack of user-interaction.

Overall, this procedure provides an overview of the necessary steps to accomplishing automation with Nessus, from initial setup of the environment to remediating results. The provided troubleshooting techniques and error codes should help to ensure easy implementation on a network, coupled with the detailed insight into optional parameters that allow for a variety of customisation for certain requests.

3.2 FUTURE WORK

The modularity of the requests should be developed upon. The current iteration of the script only allows for the "Advanced Network Scan" to be initiated and adding more scans to further develop the utility of the script would be desirable. As mentioned, but not implemented, using Selenium to create a folder for automated scans based on the day they were run would further increase the user-friendliness of the script.

Furthermore, better error handling would be added. Instead of exiting the script once a response that is anything other than successful, the script should print the error code and give advice on what the issue would be in regard to user-input. If the request is misconfigured, the error codes should be provided with insight to potential causes of these issues before exiting the script.

Example for an error code produced due to misconfiguration of a launch request:

 # Error Status 404: "Returned if Tenable Vulnerability Management cannot find the specified scan" – Ensure you are passing the scan_id correctly by using print(scan_id) and check the URL passed to ensure that is also correct.

Better editing of .CSVs to enhance the reports generated by the script, including charts, better formatting and potentially converting this information to .docx and passing through important/highlighted information to further automate the report writing process.

Nessus API interaction via Python's Requests libraries would be removed. Instead, opting to use Selenium to automate the downloading of generated reports.

The Nessus Automation Script would better achieve the goal of reducing time spent conducting scans by simply allowing for the scans to be scheduled. This can be achieved with Selenium and would reduce the need for user interaction with Nessus. The current functionality of the script is still relevant but would need to be developed further to give the option to schedule the scan for a later date/time or run a scan at that current time.

4 REFERENCES

6sense, 2024. *Nmap Vs Nessus Vulnerability Scanner : In-Depth Comparison.* [Online]
Available at: https://6sense.com/tech/network-security/nmap-vs-nessusvulnerabilityscanner
[Accessed 24 April 2024].

Helton, A., 2023. *API keys vs tokens - what's the difference?.* [Online]
Available at: https://www.gomomento.com/blog/api-keys-vs-tokens-whats-the-difference
[Accessed 24 April 2024].

Sayegh, E., 2024. *The Imperative Of Patching: A Resolution For Cybersecurity In 2024.* [Online]
Available at: https://www.forbes.com/sites/emilsayegh/2024/01/11/the-imperative-of-patching-a-resolution-for-cybersecurity-in-2024/
[Accessed 24 April 2024].

SecureLayer7, 2023. *The Pros And Cons Of Vulnerability Scanning Tools.* [Online]
Available at: https://blog.securelayer7.net/vulnerability-scanning-tools-pros-and-cons/
[Accessed 24 April 2024].

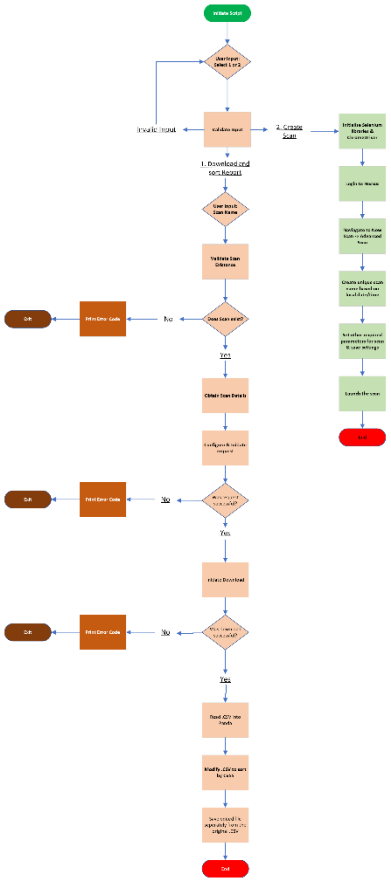
Tenable, 2024. *Purchase Tenable Solutions.* [Online]
Available at: https://www.tenable.com/buy
[Accessed 24 April 2024].

Vazquez, I., 2023. *Economic impact of automation and artificial intelligence.* [Online]
Available at: https://www.watchguard.com/wgrd-news/blog/economic-impact-automation-and-artificial-intelligence#:~:text=According%20to%20the%20same%20IMB,experienced%20significantly%20lower%20incident%20costs.
[Accessed 24 April 2024].

APPENDICES

APPENDIX A

The Flowchart for the script:



APPENDIX B

The following is the Nessus Report as .csv and then the modified .csv created with Panda.

Original Nessus Report:

Plugin ID	CVE	CVSS v2.0 Base Soc	Risk	Host	Protocol	Port	Name	Synopsis	Description	Solution	See Also	Plugin Output
10107			None	192.168.0.1	tcp	80	HTTP Server Type and Version	A web server is running on the remote host.	This plugin attempts to determine the type and the version of the	n/a		The
10287			None	192.168.0.1	udp	0	Traceroute Information	It was possible to obtain traceroute information.	Makes a traceroute to the remote host.	n/a		For your
10663		3.3	Low	192.168.0.1	udp	67	DHCP Server Detection	The remote DHCP server may expose information about the associated	This script contacts the remote DHCP server (if any) and attempts to	Apply filtering to keep this		
11002			None	192.168.0.1	tcp	53	DNS Server Detection	A DNS server is listening on the remote host.	The remote service is a Domain Name System (DNS) server, which	Disable this service if it is not	https://en.wikipedia.org/wiki/Domain_Name_System	
11002			None	192.168.0.1	udp	53	DNS Server Detection	A DNS server is listening on the remote host.	The remote service is a Domain Name System (DNS) server, which	Disable this service if it is not	https://en.wikipedia.org/wiki/Domain_Name_System	
11219			None	192.168.0.1	tcp	53	Nessus SYN scanner	It is possible to determine which TCP ports are open.	This plugin is a SYN 'half-open' port scanner. It shall be reasonably	Protect your target with an IP filter.		Port 53/tcp was found to be open
11219			None	192.168.0.1	tcp	80	Nessus SYN scanner	It is possible to determine which TCP ports are open.	This plugin is a SYN 'half-open' port scanner. It shall be reasonably	Protect your target with an IP filter.		Port 80/tcp was found to be open
11936			None	192.168.0.1	tcp	0	OS Identification	It is possible to guess the remote operating system.	Using a combination of remote probes (e.g., TCP/IP, SMB, HTTP, NTP,	n/a		
19506			None	192.168.0.1	tcp	0	Nessus Scan Information	This plugin displays information about the Nessus scan.	This plugin displays, for each tested host, information about the	n/a		Informati
22964			None	192.168.0.1	tcp	80	Service Detection	The remote service could be identified.	Nessus was able to identify the remote service by its banner or by	n/a		A web server is running on this port.
24260			None	192.168.0.1	tcp	80	HyperText Transfer Protocol (HTTP) Information	Some information about the remote HTTP configuration can be extracte	This test gives some information about the remote HTTP protocol - the	n/a		
24260			None	192.168.0.1	tcp	49152	HyperText Transfer Protocol (HTTP) Information	Some information about the remote HTTP configuration can be extracte	This test gives some information about the remote HTTP protocol - the	n/a		
24260			None	192.168.0.1	tcp	49153	HyperText Transfer Protocol (HTTP) Information	Some information about the remote HTTP configuration can be extracte	This test gives some information about the remote HTTP protocol - the	n/a		
25220			None	192.168.0.1	tcp	0	TCP/IP Timestamps Supported	The remote service implements TCP timestamps.	The remote host implements TCP timestamps, as defined by RFC1323. A	n/a	http://www.ietf.org/rfc/rfc1323.txt	
35711			None	192.168.0.1	udp	1900	Universal Plug and Play (UPnP) Protocol Detection	The remote device supports UPnP.	The remote device answered an SSDP M-SEARCH request. Therefore, it	Filter access to this port if de	https://e	The
35712			None	192.168.0.1	tcp	49152	Web Server UPnP Detection	The remote web server provides UPnP information.	Nessus was able to extract some information about the UPnP-enabled	Filter incoming traffic to this p	https://er	
35712			None	192.168.0.1	tcp	49153	Web Server UPnP Detection	The remote web server provides UPnP information.	Nessus was able to extract some information about the UPnP-enabled	Filter incoming traffic to this p	https://er	
35716			None	192.168.0.1	tcp	0	Ethernet Card Manufacturer Detection	The manufacturer can be identified from the Ethernet OUI.	Each ethernet MAC address starts with a 24-bit Organizationally	n/a	https://st	
43111			None	192.168.0.1	tcp	80	HTTP Methods Allowed (per directory)	This plugin determines which HTTP methods are allowed on various CGI	By calling the OPTIONS method, it is possible to determine which HTTP	n/a	http://w	Based
45590			None	192.168.0.1	tcp	0	Common Platform Enumeration (CPE)	It was possible to enumerate CPE names that matched on the remote	By using information obtained from a Nessus scan, this plugin reports	n/a	http://cp	
50686	CVE-1999-051	5.8	Medium	192.168.0.1	tcp	0	IP Forwarding Enabled	The remote host has IP forwarding enabled.	The remote host has IP forwarding enabled. An attacker can exploit	On Linux, you can disable		IP
54615			None	192.168.0.1	tcp	0	Device Type	It is possible to guess the remote device type.	Based on the remote operating system, it is possible to determine	n/a		Remote
85805			None	192.168.0.1	tcp	80	HTTP/2 Cleartext Detection	An HTTP/2 server is listening on the remote host.	The remote host is running an HTTP server that supports HTTP/2 running	Limit incoming traffic to this p	https://h	
86420			None	192.168.0.1	tcp	0	Ethernet MAC Addresses	This plugin gathers MAC addresses from various sources and	This plugin gathers MAC addresses discovered from both remote probing	n/a		The
106628			None	192.168.0.1	tcp	80	lighttpd HTTP Server Detection	The lighttpd HTTP server was detected on the remote host.	Nessus was able to detect the lighttpd HTTP server by looking at	n/a	https://w	
106658			None	192.168.0.1	tcp	80	JQuery Detection	The web server on the remote host uses JQuery.	Nessus was able to detect JQuery on the remote host.	n/a	https://jq	
156439			None	192.168.0.1	tcp	80	jQuery UI Detection	The web server on the remote host uses jQuery UI.	The web server on the remote host uses jQuery UI.	n/a	https://re	
10287			None	192.168.0.33	udp	0	Traceroute Information	It was possible to obtain traceroute information.	Makes a traceroute to the remote host.	n/a		For your
11219			None	192.168.0.33	tcp	62078	Nessus SYN scanner	It is possible to determine which TCP ports are open.	This plugin is a SYN 'half-open' port scanner. It shall be reasonably	Protect your target with an IP filter.		Port 62078/tcp was found to be open
11936			None	192.168.0.33	tcp	0	OS Identification	It is possible to guess the remote operating system.	Using a combination of remote probes (e.g., TCP/IP, SMB, HTTP, NTP,	n/a		
19506			None	192.168.0.33	tcp	0	Nessus Scan Information	This plugin displays information about the Nessus scan.	This plugin displays, for each tested host, information about the	n/a		Informati
22964			None	192.168.0.33	tcp	62078	Service Detection	The remote service could be identified.	Nessus was able to identify the remote service by its banner or by	n/a		The
25220			None	192.168.0.33	tcp	0	TCP/IP Timestamps Supported	The remote service implements TCP timestamps.	The remote host implements TCP timestamps, as defined by RFC1323. A	n/a	http://www.ietf.org/rfc/rfc1323.txt	
54615			None	192.168.0.33	tcp	0	Device Type	It is possible to guess the remote device type.	Based on the remote operating system, it is possible to determine	n/a		Remote
86420			None	192.168.0.33	tcp	0	Ethernet MAC Addresses	This plugin gathers MAC addresses from various sources and	This plugin gathers MAC addresses discovered from both remote probing	n/a		The
10287			None	192.168.0.54	udp	0	Traceroute Information	It was possible to obtain traceroute information.	Makes a traceroute to the remote host.	n/a		For your
11219			None	192.168.0.54	tcp	903	Nessus SYN scanner	It is possible to determine which TCP ports are open.	This plugin is a SYN 'half-open' port scanner. It shall be reasonably	Protect your target with an IP filter.		Port 903/tcp was found to be open
11219			None	192.168.0.54	tcp	913	Nessus SYN scanner	It is possible to determine which TCP ports are open.	This plugin is a SYN 'half-open' port scanner. It shall be reasonably	Protect your target with an IP filter.		Port 913/tcp was found to be open
11936			None	192.168.0.54	tcp	0	OS Identification	It is possible to guess the remote operating system.	Using a combination of remote probes (e.g., TCP/IP, SMB, HTTP, NTP,	n/a		
19506			None	192.168.0.54	tcp	0	Nessus Scan Information	This plugin displays information about the Nessus scan.	This plugin displays, for each tested host, information about the	n/a		Informati
20301			None	192.168.0.54	tcp	903	VMware ESX/GSX Server detection	The remote host appears to be running VMware Server, ESX Server, or	According to its banner, the remote host appears to be running a	n/a	https://www.vmware.com/	
20301			None	192.168.0.54	tcp	913	VMware ESX/GSX Server detection	The remote host appears to be running VMware Server, ESX Server, or	According to its banner, the remote host appears to be running a	n/a	https://www.vmware.com/	
22964			None	192.168.0.54	tcp	903	Service Detection	The remote service could be identified.	Nessus was able to identify the remote service by its banner or by	n/a		A VMware authentication daemon is running o
22964			None	192.168.0.54	tcp	913	Service Detection	The remote service could be identified.	Nessus was able to identify the remote service by its banner or by	n/a		A VMware authentication daemon is running o
35716			None	192.168.0.54	tcp	0	Ethernet Card Manufacturer Detection	The manufacturer can be identified from the Ethernet OUI.	Each ethernet MAC address starts with a 24-bit Organizationally	n/a	https://st	
45590			None	192.168.0.54	tcp	0	Common Platform Enumeration (CPE)	It was possible to enumerate CPE names that matched on the remote	By using information obtained from a Nessus scan, this plugin reports	n/a	http://cp	
53513			None	192.168.0.54	udp	5355	Link-Local Multicast Name Resolution (LLMNR) De	The remote device supports LLMNR.	The remote device answered to a Link-local Multicast Name Resolution	Make sure that use of this	http://w	
54615			None	192.168.0.54	tcp	0	Device Type	It is possible to guess the remote device type.	Based on the remote operating system, it is possible to determine	n/a		Remote
66717			None	192.168.0.54	udp	5353	mDNS Detection (Local Network)	It is possible to obtain information about the remote host.	The remote service understands the Bonjour (also known as ZeroConf or	Filter incoming traffic to UDP port 5353		Nessus
86420			None	192.168.0.54	tcp	0	Ethernet MAC Addresses	This plugin gathers MAC addresses from various sources and	This plugin gathers MAC addresses discovered from both remote probing	n/a		The
10114	CVE-1999-052	0	None	192.168.0.83	icmp	0	ICMP Timestamp Request Remote Date Disclosure	It is possible to determine the exact time set on the remote host.	The remote host answers to an ICMP timestamp request. This allows an	Filter out the ICMP		The

Modified Nessus Report:

CVSS v2.0 Base	Host	Protocol	Port	Name
6.4	192.168.0.241	tcp	8834	SSL Certificate Cannot Be Trusted
6.4	192.168.0.118	tcp	10001	SSL Certificate Cannot Be Trusted
6.4	192.168.0.118	tcp	10001	SSL Self-Signed Certificate
6.1	192.168.0.118	tcp	10001	TLS Version 1.1 Protocol Deprecated
6.1	192.168.0.118	tcp	10001	TLS Version 1.0 Protocol Detection
5.8	192.168.0.1	tcp	0	IP Forwarding Enabled
5	192.168.0.115	tcp	445	SMB Signing not required
4.3	192.168.0.118	tcp	10001	SSL RC4 Cipher Suites Supported (Bar Mitzvah)
4.3	192.168.0.118	tcp	10001	SSL RC4 Cipher Suites Supported (Bar Mitzvah)
3.3	192.168.0.166	icmp	0	Multiple Ethernet Driver Frame Padding Information Disclosure (Etherleak)
3.3	192.168.0.1	udp	67	DHCP Server Detection
2.6	192.168.0.241	tcp	0	OpenJDK 8 <= 8u402 / 11.0.0 <= 11.0.22 / 17.0.0 <= 17.0.10 / 21.0.0 <= 21.0.2 / 22.0.0 <= 22.0.0 Multiple Vulnerabilities (2024-04-16
2.6	192.168.0.241	tcp	0	OpenJDK 8 <= 8u402 / 11.0.0 <= 11.0.22 / 17.0.0 <= 17.0.10 / 21.0.0 <= 21.0.2 / 22.0.0 <= 22.0.0 Multiple Vulnerabilities (2024-04-16
2.6	192.168.0.241	tcp	0	OpenJDK 8 <= 8u402 / 11.0.0 <= 11.0.22 / 17.0.0 <= 17.0.10 / 21.0.0 <= 21.0.2 / 22.0.0 <= 22.0.0 Multiple Vulnerabilities (2024-04-16
2.6	192.168.0.241	tcp	0	OpenJDK 8 <= 8u402 / 11.0.0 <= 11.0.22 / 17.0.0 <= 17.0.10 / 21.0.0 <= 21.0.2 / 22.0.0 <= 22.0.0 Multiple Vulnerabilities (2024-04-16
2.6	192.168.0.241	tcp	0	OpenJDK 8 <= 8u402 / 11.0.0 <= 11.0.22 / 17.0.0 <= 17.0.10 / 21.0.0 <= 21.0.2 / 22.0.0 <= 22.0.0 Multiple Vulnerabilities (2024-04-16
0	192.168.0.162	icmp	0	ICMP Timestamp Request Remote Date Disclosure
0	192.168.0.101	icmp	0	ICMP Timestamp Request Remote Date Disclosure
0	192.168.0.232	icmp	0	ICMP Timestamp Request Remote Date Disclosure
0	192.168.0.83	icmp	0	ICMP Timestamp Request Remote Date Disclosure
	192.168.0.1	tcp	80	HTTP Server Type and Version
	192.168.0.1	udp	0	Traceroute Information
	192.168.0.1	tcp	53	DNS Server Detection
	192.168.0.1	udp	53	DNS Server Detection
	192.168.0.1	tcp	53	Nessus SYN scanner
	192.168.0.1	tcp	80	Nessus SYN scanner
	192.168.0.1	tcp	0	OS Identification
	192.168.0.1	tcp	0	Nessus Scan Information
	192.168.0.1	tcp	80	Service Detection
	192.168.0.1	tcp	80	HyperText Transfer Protocol (HTTP) Information
	192.168.0.1	tcp	49152	HyperText Transfer Protocol (HTTP) Information
	192.168.0.1	tcp	49153	HyperText Transfer Protocol (HTTP) Information
	192.168.0.1	tcp	0	TCP/IP Timestamps Supported
	192.168.0.1	udp	1900	Universal Plug and Play (UPnP) Protocol Detection
	192.168.0.1	tcp	49152	Web Server UPnP Detection
	192.168.0.1	tcp	49153	Web Server UPnP Detection
	192.168.0.1	tcp	0	Ethernet Card Manufacturer Detection
	192.168.0.1	tcp	80	HTTP Methods Allowed (per directory)
	192.168.0.1	tcp	0	Common Platform Enumeration (CPE)
	192.168.0.1	tcp	0	Device Type
	192.168.0.1	tcp	80	HTTP/2 Cleartext Detection
	192.168.0.1	tcp	0	Ethernet MAC Addresses

APPENDIX C

The following is the Nessus Report from the automated scan "22_Apr_2024_20:35:31"