Abertay University

Developing an Automated Network Penetration Testing Script Written in Python

How a scripted tool can generate and support opportunities for learning and demonstrate the need for established methodologies, automatically creating reports during the testing process.

[redacted]

CMP509: Ethical Hacking Masters

2023/24

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

.

Abstract

It has been argued that the challenges facing the cyber security industry around the recruitment of sufficiently educated and experienced professionals whilst ensuring market demands are met are not being sufficiently addressed by the training resources currently available. The concerns around the existing offerings are centered around the inability for students to apply what they learn in these solutions to real-world situations. The development of a tool for automating a penetration test that also produces reports that would support a learner's understanding could help address the imbalance by providing the opportunity for customisation to the needs of individual organisations.

A script, written in the Python programming language, was designed and developed to provide a proof-of-concept of an automated penetration test that could also produce reports with content intended for differing audiences. This script was used against a test server to conduct a network penetration test, for which minimal user input was required. The machine used to conduct the penetration test against was hosted as a virtual machine. The script incorporated established methodologies to assist learners in the comprehension of the need for a structured approach when conducting a penetration test. The finalised proof-of-concept included Python instructions from inbuilt and imported libraries to achieve the desired functionality and used a combination of standard programming techniques, such as loops, functions and variables.

The developing script demonstrated promise as a learning tool for use within a corporate network when combined with a test environment to conduct penetration test exercises against. The majority of the tooling used by the script provided output that could be effectively manipulated to extract relevant information though there was scope for the development of custom tooling to achieve increased levels of automation. The reports generated included comments to the learner, customised to the results of the penetration test, that would provide learners with opportunities to identify areas where their knowledge of tooling required improvement whilst also offering the possibility for a user to customise the script to their own needs. Both the Learner and Management Reports generated could benefit from some refinement in their aesthetic to improve a learner's understanding of commercial expectations when producing reports for the customer. The use of script enhancements also assisted in increasing the appeal of the program.

.

Contents

.

.

1 INTRODUCTION

1.1 BACKGROUND

A recent report from the Department of Science, Innovation and Technology (DSTI) identified that 78% of cybersecurity course graduates find employment within two years of graduation, a third of which are in cybersecurity-related professions (Coutinho, et al., 2023). Despite these findings, a separate study recognised that training providers and employers felt that university courses did not adequately prepare graduates for cyber roles and did not teach students how to apply the knowledge gained to a work environment (Williams, et al., 2021).

Since 2020, the market for penetration testing has consistently increased year on year, and is expected to grow exponentially into 2030 (Polaris Market Research, 2022), demonstrated in Figure 1.1:

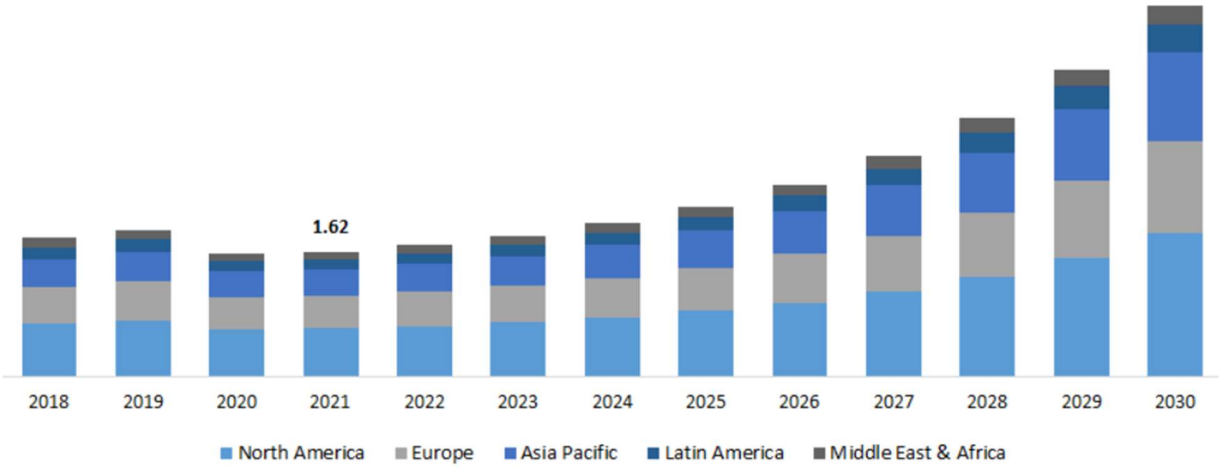

Figure 1.1: Penetration testing market size, by region, 2018 - 2030 *(Polaris Market Research, 2022)*.

These statistics represent a challenge to both employers and graduate/entry-level cybersecurity professional regarding the availability of training resources available that encourage the acquisition of appropriate and relevant skills whilst being able to meet the demands within the marketplace. Established security practitioners are aware of the lack of appropriate training resources (Weidman, 2014) and there is a concerted effort being made to offer more relevant training to aspiring penetration testers (Li, 2015), though delivery methods for these continue to appear focused around educational courses and competitions (Schmeelk & Dragos, 2023), rather than in practical workplace-based engagements.

HackTheBox is a platform that prides itself on providing a penetration testing certification path that differentiates itself from the standard market offering, and emphasises both practical skills and fundamental knowledge using lab environments that resemble corporate environments (mrb3n, 2023), which requires the production of a finalised report by the learner as part of the assessment process.

However, the exam and related course content is hosted entirely within the HackTheBox Academy platform and cannot be customised to an individual organisation's needs.

This report follows the development of an automated Python script capable of conducting an automated network penetration test with a view to producing documentation that would be beneficial to a learner or entry-level employee to assist in their comprehension of the tools and techniques used in a penetration test. It is envisaged that the script would typically form part of a set of tools available to a graduate or junior penetration tester to be used alongside an enclosed test environment within the employing organisation's infrastructure. The script will also serve to educate the user in the use of a methodology to ensure the production of a clear and structured report that could be presented to a customer. For the purposes of this report, the server targeted during the network penetration test is a virtual machine and all data held on it is entirely fictional.

This paper's main content lies in the Program Development section, introduced by the overarching procedure that the author (hereafter referred to as "the developer") chose to implement. The first section, Pre-requisites, does not form a part of the script development exercise itself but instead offers a detailed account of the installation of any additional tools that were required prior to the commencement of the script development process. Specific commands and download locations (accurate at the time of writing) are provided here. Each function's procedural section is followed by a functionality validation section, which outlines the script's output for that particular function. The paper is concluded with a Discussion section, where the script's effectiveness in conducting an automated network penetration test and in producing sufficient learning opportunities for the user are discussed. Concepts for further work that could be done to further develop the script are also included in the final Discussion section.

The scope of the script development is limited to the testing of network security and does not involve any processes for the testing of web applications. Web application penetration testing is considered separate to network security testing and requires in-depth testing in its own right (Halock, 2020). The scope is further limited to the testing of services actively available for enumeration on the test server, and to those phases of the penetration test that occur after a reconnaissance phase (demonstrated in Figure 1.2) and as such should be considered a proof of concept in development.

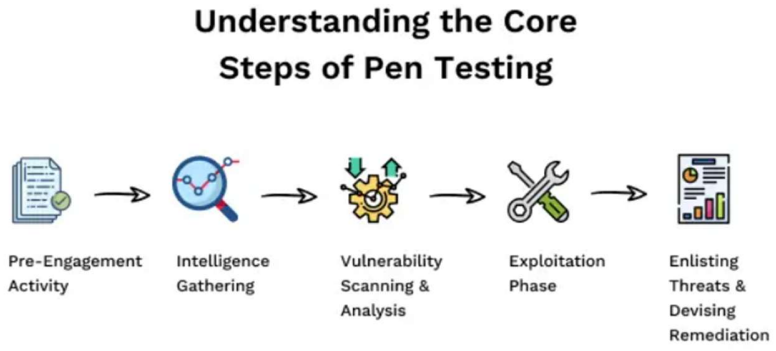

Figure 1.2: The basic stages of a penetration test *(SecureTriad, n.d.)*.

It is not the intention of the developer that a finalised program would be a replacement for a full penetration test owing to the manual intervention required to validate penetration test findings, particularly in those tests applied to environments that have custom functionality (Stuttard & Pinto, 2011). This paper will focus on the methods and techniques used in the development of a script for the discovery and enumeration of potential network security vulnerabilities only.

1.2 AIMS

This report aims to detail the development of a scripted program, written in Python, as a proof-of-concept for an automated penetration test, intended for use as a training tool. This broad aim can be broken down into several sub aims:

- Outline the development stages of the program composition, encompassing an automated network penetration test.
- Discuss the scope for incorporating both existing tools and the creation of custom tools that could be incorporated into a finished program.
- Describe the methods used by the program to produce multiple reports for different audiences.
- Assess the context in which the program could be used as a training tool, identifying how it could enhance an understanding of penetration test tools and processes, as well as identifying opportunities for users to customise the program.

2 PROGRAM DEVELOPMENT

2.1 OVERVIEW OF PROGRAM DEVELOPMENT

For the purposes of this report, the developer sought to design a script, written in Python, that would conduct an automated penetration test. It was the developer's intention that the script would act as a training tool that would aid learners in understanding the processes involved in administering a penetration test. The initial pseudo-code created for this program is outlined in Figure 2.1:

Figure 2.1: Overarching pseudo-code for program.

The developer considered it appropriate to incorporate established methodologies that would guide the penetration testing process. With this in mind, appropriate methodologies were selected for both network and web application penetration tests (though this paper encompasses the development of an automated network penetration test only). Figure 2.2 shows the chosen methodology for a network penetration test, adapted from the seven phases of penetration testing described by PTES (PTES).

Figure 2.2: Adopted process for network penetration test exercises.

The Web Security Testing Guide (WSTG) from the Open Source Foundation for Application Security details a testing methodology in a set of twelve stages (OWASP Foundation, 2020), as can be seen in Figure 2.3. This was to be the chosen methodology for conducting a web application penetration test.

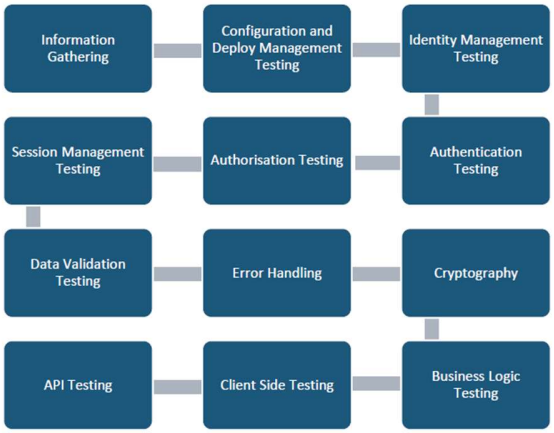

Figure 2.3: The twelve stages of the WSTG *(OWASP Foundation, 2020)*.

The development of the finalised script was not a linear process from the first line of code to the last. The program was written in the following order:

1. Menu design (main menu first, which led to the development of the network and web application penetration test sub-menus).
2. networkpentest function design, including its nested functions.
3. Inclusion of script enhancement components.
4. Finalised construction of the script from its component parts, including function definitions. This step was completed lastly to allow for the importing of all libraries and libraries required to achieve the desired functionality.

The developer employed a number of standard practices throughout the scripting process, and as such individual instances of these practices will not be commented upon in the report. The details of these practices are listed below:

- Comments have been used to describe each action taken by the script (indicated by the use of a hash character), with function definitions being separated using a header including a function title and description.
- Functions were used where appropriate (once their functionality as was confirmed as successful in isolation) to maintain the flow of outputted reports and invoked using standard Python definition and call practices.
- f-strings (implemented using an "`f`" prefix) were used for strings containing variable contents, where the variable is specified using curly braces (`{}`), e.g. `print(f"{user}\n")`.
- Raw strings (implemented using an "`r`" prefix) were used for strings containing special characters to be treated as part of the string, e.g. `text=r"^[0-9]"`.
- Progress messages were used throughout the script execution, with output to the screen achieved using the inbuilt `print()` instruction.
- Multiple lines of text were achieved by enclosing the text body in "`print("""`" opening and "`""")`" closing statements.
- Obtaining user input was achieved using the inbuilt `input()` statement and saved to local variables.

The developer made use of various tools and libraries to perform specific duties within the script. These tools, their versions (where applicable), and the reasoning for their use can be found in Appendix A.

For the purposes of this report, in instances where the developer used code containing a variable value (e.g. a string value), these are indicated with the use of chevrons (<>) within the code, for example:

```
print(<someStringHere>)
```

The developer utilised several platforms for varying purposes throughout the development process. These platforms and their purposes can be seen in Table 2.1:

Table 2.1: Platforms used for script development.

Platform name	Version number	Purpose
Windows 11	23H2 build	Hosting the VMware Workstation Pro virtualisation software.
VMware Workstation Pro	17.5.1	Hosting the test server and Kali virtual machine.
Windows Server 2019	1809	Used for running network infrastructure penetration tests against.
Kali (virtual machine)	5.18.0-kali5-amd64	Used for conducting the penetration test exercise, including the hosting of a vulnerability scanning platform. Python script was developed and run from this machine.
mousepad	0.5.10	GUI-based text editor used for composition of the Python.
Python	3.11.18	Programming language used for the script.
Nessus Essentials	10.3.0	Used to produce vulnerability assessment results.

With regards to the VMware Workstation Pro platform, the developer chose to utilise the Pro version of the VMware virtualisation software as it offers a snapshotting feature that its free version (VMware Player) does not. This snapshotting allowed the developer to reset the target machine to a non-corrupted state for further testing if needed.

Kali was the platform of choice because it is purpose built for penetration testing (Kali, n.d.) and ships with hundreds of tools pre-installed for this purpose. The developer chose to use Mousepad, a native Kali tool, as the text editor when designing the script as the program includes automatic colour-coding of scripting languages for ease of reading/writing. Nessus was chosen as the vulnerability scanner as its free version (Essentials) is easy to use, offers a number of comprehensive scan templates and produces reports that are simple to interpret and interact with (Awati, 2023).

The choice of name for the program (“runmethru”) is based on a simple comparison between the slang term “to run through”, which Merriam-Webster defines as “to penetrate or hold (something) with a pointed object” (n.d.), and the term “penetration test”. The developer considered the name “runmethru” for a program simulating a penetration test to be a fitting title with this context in mind.

2.2 PRE-REQUISITES

2.2.1 VMware Workstation Pro installation

VMware Workstation Pro is a paid-for software application, and as such its installation is out of scope for this report. The installer media can be obtained from the VMware website (account creation is required):

https://www.vmware.com/products/workstation-pro/workstation-pro-evaluation.html

The installation was a simple process, which required the developer to select all the defaults offered by the installer. The installer was run as an administrator within a Windows machine.

2.2.2 Nessus Essentials

Nessus Essentials was already installed on the Kali virtual machine that was used by the developer, but it is not part of the standard suite of tools pre-installed on Kali distributions. Installing Nessus, which is a simple but time-consuming process, is out of scope for this report. The installer media for Nessus Essentials can be downloaded (after completing a registration form for an activation key) from the Nessus website using the default options presented:

https://www.tenable.com/products/nessus/nessus-essentials

After downloading, the installation is performed from the command line in Kali using:

```
sudo dpkg -I <installationFileName>
```

The Nessus scanner is then started, also from the command line:

```
sudo systemctl start nessusd.service
```

The installation can be completed using an internet browser by navigating to the URL below.

https://kali:8834

The received activation key is entered here, and a username and password created for the administrator account for use with the Nessus Essentials console. The installation process can be monitored from the MyScans Dashboard. Once installation is complete, scans can be configured for use.

2.2.3 Libraries

Whilst the developer used several built-in Python libraries, some of the script's functionality relies of the importing of some libraries that are not native to a standard Python installation. These libraries can be installed with the following command:

```
pip install python-docx nmap colorama
```

2.2.4 SecLists

The SecLists repository of word lists was downloaded from a GitHub repository using the following command line in Kali:

```
git clone https://github.com/danielmiessler/SecLists
```

This downloaded the top-level folder and all of its children, as shown in Figure 2.4:

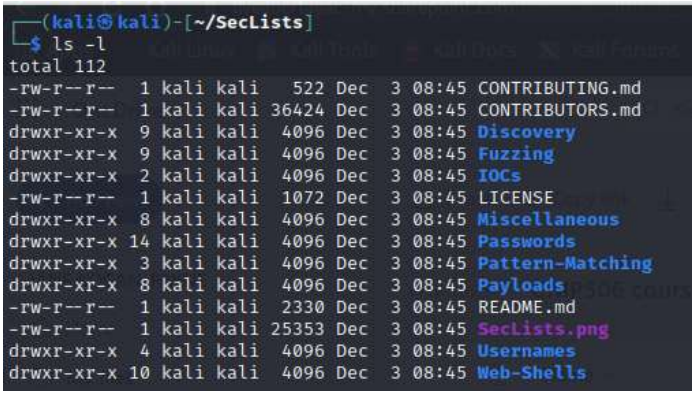

Figure 2.4: Directory listing of SecLists repository.

2.2.5 Xsltproc

Xsltproc was already installed on the Kali virtual machine that was used the developer, but it is not part of the standard suite of tools pre-installed on Kali distributions. It can be installed with the following command:

```
sudo apt update && sudo apt install xsltproc -y
```

2.3 MENUS

The script required the design of three menus:

1. A main menu.
2. A sub-menu for individual stages of a network penetration test.
3. A sub-menu for individual stages of a web application penetration test.

2.3.1 Main menu

The main menu was used to offer the user the following options:

1. Run an automated network penetration test in full.
2. Run an automated web application penetration test in full.
3. Navigate to a sub-menu to choose an individual stage of a network penetration test.
4. Navigate to a sub-menu to choose an individual stage of a web application penetration test.
5. Quit the program.

2.3.1.1 Code design

The developer first created an introductory body of text to be displayed to the user upon program launch, which included the text for the menu itself. With the knowledge that this menu was to offer the user multiple options to choose from, each of which resulted in the script following a different execution path, the developer chose to implement a switch-case statement instead of multiple if-else statements in order to maintain a clean layout for the script (Sharma, 2024). The desired functionality of this switch-case statement is demonstrated in the pseudo-code shown in Figure 2.5:

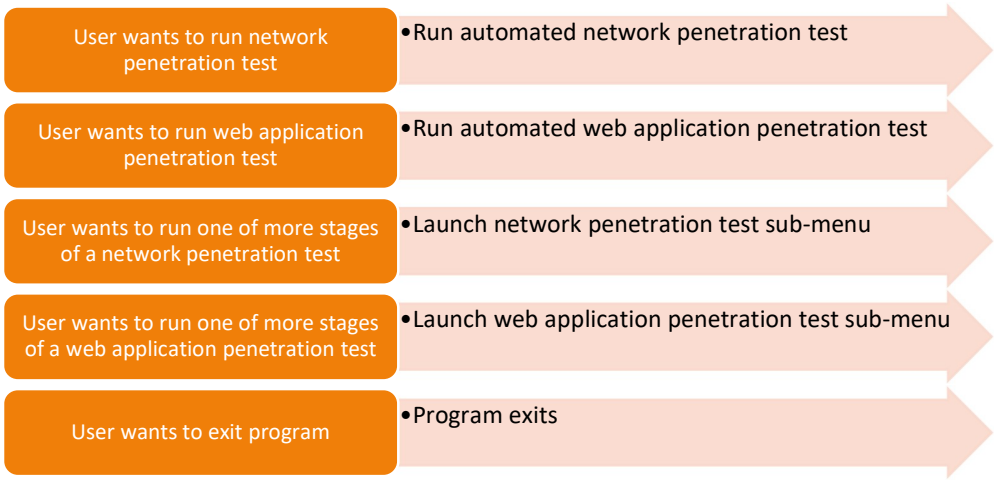

Figure 2.5: Pseudo-code for main menu functionality.

The resulting switch-case statements can be seen in Figure 2.6 below, which includes the `input()` statement for accepting user input. The developer chose to use functions designed to carry out specific functionality, as per standard Python best practice (Hart-Davis & Hart-Davis, 2022). Their invocations in accordance with the user's choice can be seen below:

```
choice=input("Please enter your choice: ")

match choice:
    case "1":
        networkpentest()

    case "2":
        webapppentest()

    case "101":
        networkmenu()

    case "201":
        webapplicationmenu()

    case "0":
        quit()
```

Figure 2.6: Switch-case statement for main menu choice.

2.3.1.2 Functionality verification

The resulting menu display can be seen in Figure 2.7:

```
Please choose the type of penetration test you want to conduct by selecting an option from the menu below or enter 0 to exit the program.

1. Network Infrastructure
2. Web Application

If you would rather run the tests for a specific stage of a penetration test, please choose which methodology you want to explore:

101. Network Infrastructure
201. Web Application

0. Exit program


Please enter your choice:
```

Figure 2.7: Main menu display.

2.3.2 Network testing stage choice sub-menu

The network testing sub-menu was used to offer the user the following options:

1. Run one of four stages of a network penetration test.
2. Return to the main menu.
3. Quit the program.

2.3.2.1 Code design

The developer used the same process for designing this menu as had been applied for the main menu. The desired logical flow for this menu is demonstrated in the pseudo-code shown in Figure 2.8:

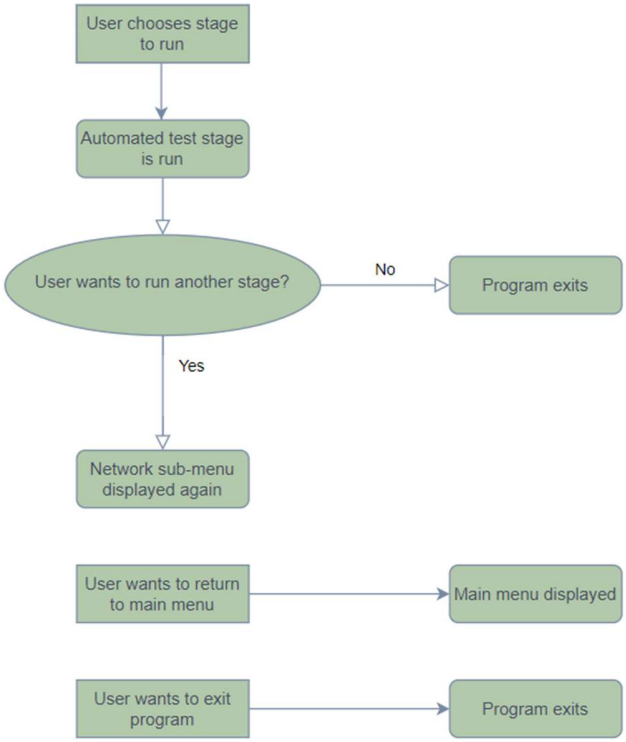

Figure 2.8: Pseudo-code for network sub-menu functionality.

An example of the resulting switch-case statements can be seen in Figure 2.9 below, which includes the `input()` statement for accepting the user input. Further case statements were included in the developing script to encompass the three other choices available to the user provided on the sub-menu but have been omitted here for brevity. Placeholders (commented) have been used for the functions that would be invoked in a finalised version of the script.

```
choice=input("Please enter your choice: ")

match choice:
    case "1":
        print("Calling Network Scanning Tools function")#Placeholder for calling networkenumeration function
        cont=input("Tests complete. Do you want to run any other network penetration testing stages? Y/N: ")
        cont=cont.lower()
        if cont=="n":
            print("Program exiting.")
            quit()
        elif cont=="y":
            print("Returning to menu.")
            networkmenu()

    case "99":
        print("Returning to main menu.")
        mainmenu()

    case "0":
        print("Program is exiting.")
        quit()
```

Figure 2.9: Switch-case statement for network sub-menu choice.

The developer considered it necessary to add some string formatting to the contents of the input variable used to determine whether the script should return to the sub-menu or quit the program. This was implemented to allow for the case sensitivity of the Python programming language, and can be seen in the following two lines of code:

```
cont=input("Tests complete. Do you want to run any other network penetration testing stages? Y/N: ")
```

```
cont=cont.lower()
```

The `.lower()` operation served to convert the string to lower case, ensuring that the user's use of case is irrelevant – should they enter their choice as lower case, no effective string manipulation takes place. With this input validation in place the code was able to evaluate the user's choice more effectively.

2.3.2.2 Functionality verification

The resulting menu display can be seen in Figure 2.10:

```
Please choose the stage of the network infrastructure penetration test you want to conduct by selecting an option from the menu.

1. Scanning
2. Enumeration
3. Pasword Hacking
4. System Hacking

99. Return to main menu.
0. Exit program


Please enter your choice:
```

Figure 2.10: Network sub-menu display.

2.3.3 Web application testing choice sub-menu

The web application testing sub-menu was used to offer the user the following options:

1. Run one of twelve stages of a web application penetration test.
2. Return to the main menu.
3. Quit the program.

2.3.3.1 Code design

The process for designing the display of this sub-menu was the same as that for the network sub-menu and will not be replicated here.

The developer also acknowledged that several of the stages of a web application penetration test might benefit from the availability of a DNS host name during testing, if the user had one available. The following lines of code were initially added to the top of the web application sub-menu code:

```
dns=input("Do you have a host name for the target? Y/N: ")
dns=dns.lower()
if dns=="y":
    host=input("Please input the host name: ")
```

After testing the functionality and the effect that these lines of code had on the flow of the script execution, the developer established that these lines of code were more effective inserted into the switch-case statement within the main menu function, ensuring that the user would not be prompted for this information multiple times should they wish to run multiple stages of the web application penetration test from the sub-menu. This resulted in the block of code shown in Figure 2.11 being used for the case statement within the main menu function.

```
case "201":
    #Obtain host name if possible
    #Initialise variable for host name
    host=0

    #Take input for host variable if host name is available
    dns=input("Do you have a host name for the target? Y/N: ")
    dns=dns.lower()
    if dns=="y":
        host=input("Please input the host name: ")
    webapplicationmenu()
```

Figure 2.11: Amended case statement for the main menu, incorporating the variable setting for a DNS host name.

2.3.3.2 Functionality verification

The resulting menu display can be seen in Figure 2.12:

Figure 2.12: Web application sub-menu display.

2.4 NETWORKPENTEST FUNCTION

The desired logical flow for this function is demonstrated in the pseudo-code shown in Figure 2.13:

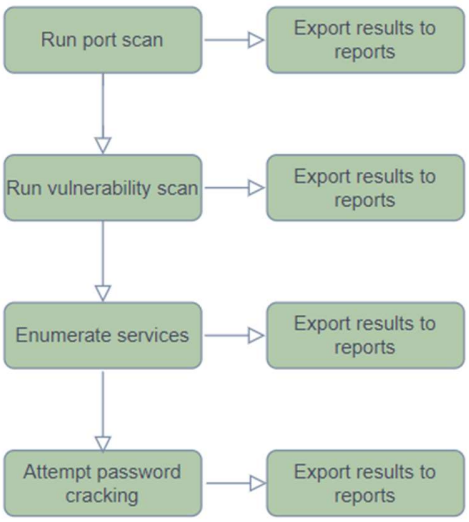

Figure 2.13: Pseudo-code for networkpentest function.

In order to achieve the desired functionality, a number of libraries and tools were used throughout the function, the details of which can be found in Table 2.2:

Table 2.2: Libraries and tools used in the networkpentest function.

Name of library	Version
subprocess	3.11.18
os	3.11.18
python-docx	0.8.11
nmap	0.7.1
ftplib	3.11.18
datetime	3.11.18
getpass	3.11.18
webbrowser	3.11.18
Name of tool	**Version**
nmap	7.95
grep	3.7
cut	8.32
tr	8.32
sed	4.8
SecLists	N/A
xsltproc	20914.10135.820
smbclient	4.16.3-Debian
enum4linux	0.9.1

cat	8.32
hydra	9.3

The developer sought to maintain the tidy layout of the script and ease of manageability of this function with the use of several nested functions that would achieve the following:

1. Perform a port scan against the target.
2. Perform a vulnerability scan against the target.
3. Perform enumeration against identified services:
    a. SMTP user enumeration.
    b. POP3 user enumeration.
    c. Enumeration of FTP directories.
    d. SMB enumeration.
4. Export the results of the enumeration to the Management and Learner Reports:
    a. SMB share information export.
    b. SMB users export.
5. Attempt to crack the passwords of discovered SMB users with hydra.

2.4.1 networkportscan nested function

The desired logical flow for this function is demonstrated in the pseudo-code shown in Figure 2.14:

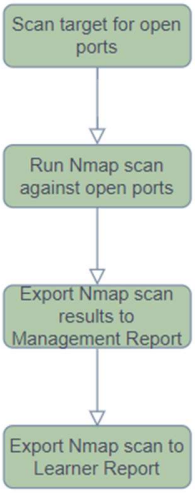

Figure 2.14: Pseudo-code for networkportscan function.

The details of the libraries and tools used for this function can be found in Table 2.3:

Table 2.3: Libraries and tools used in the networkportscan function.

Name of library	Version
datetime	3.11.18
subprocess	3.11.18
os	3.11.18
python-docx	0.8.11
nmap	0.7.1
Name of tool	**Version**
nmap	7.95
grep	3.7
cut	8.32
tr	8.32
sed	4.8
xsltproc	20914.10135.820
cat	8.32

2.4.1.1 Code design

It was necessary to provide a mechanism for the script user to provide an IP address to attack before the automated penetration test could commence. The developer also intended for a username to be added to the Learner Report generated during the penetration test process. It was determined that the most efficient method for this would be to store appropriate user-inputted values in local variables. The following lines of code were initially added to the top of the networkportscan function:

```
learnername=input("Enter your name: ")

print("Thank you. This will be used on the learner report produced during the testing process.")

targetipaddress=input("Enter the target IP address: ")
```

After testing the functionality and the effect that these lines of code had on the flow of the script execution, the developer established that these lines of code were more effective inserted into the main program as part of the program prologue. This ensured that this information could be used by multiple functions without the user being prompted for it multiple times (in the event that they wish to run multiple stages of either the network or web application penetration tests from their relative sub-menus).

In order to obtain a list of open ports on the target machine, the developer chose to invoke Nmap. With the knowledge that Nmap scans can be time consuming and network traffic intensive, the developer first wanted to implement a technique to define the range of open ports that could consequently be interrogated further for service information with the use the following command:

```
nmap -p- --min-rate=1000 <targetip> | grep ^[0-9] | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//
```

This command runs a TCP scan against a given IP address and then passes the output to a series of transformational commands. A breakdown of the command options and their purpose can be seen in Table 2.4:

Table 2.4: Explanation of command line switches when setting a variable for Nmap.

Option	Purpose
`-p-`	Specifies that all ports (1-65535) are to be scanned.
`--min-rate=1000`	Specifies that Nmap should send packets at or above 1000 per second. Stipulates that this is to be a fast scan.
`|`	Passes the output of the command before the pipe (`|`) into the command after it.
`grep`	Performs pattern matching.
`^[0-9]`	Regular expression. Instructs `grep` to find strings that begin with any integer between 0 and 9.
`cut`	String transformation command for removing sections from each line of text.
`-d '/'`	Instructs `cut` to use the `/` character as a delimiter.
`-f 1`	Instructs `cut` to select the first field in the delimited output.
`tr`	String transformation command for translating or deleting characters from text.
`'\n' ','`	Instructs `tr` to replace new lines with a comma (`,`).
`sed`	Stream editor that performs basic transformation commands on output.
`s/,$//`	Instructs `sed` to perform substitution on the output, using the `/` character as a delimiter.

It was necessary to for the subprocess library to handle this command (using the `.run` instruction) before the output could be effectively handled and used for further Nmap scanning. The developer achieved this with the block of code below, where it can be seen that the output from the command line (indicated by `subprocess.PIPE`) is assigned to the "openports" variable. The string was then further manipulated with the use of the `.stdout.strip` operation.

```
openports=subprocess.run(
      f"nmap -p- --min-rate=1000 {targetipaddress} | grep ^[0-9] | cut
-d '/' -f 1 | tr '\n' ',' | sed s/,$//",
      shell=True,
      stdout=subprocess.PIPE,
      stderr=subprocess.PIPE,
      universal_newlines=True
)
targetports=openports.stdout.strip()
```

The Nmap scan itself was performed using the simpler `os.system` instruction, as the Nmap command included the output redirection to a separate text file for later analysis. This was achieved using the following line of code:

```
os.system(f"nmap -sV {targetipaddress} -p {targetports} -oA nmapscan > nmapscan.txt")
```

An explanation of the switches used in this Nmap command can be found in Table 2.5:

Table 2.5: Explanation of switches when running the Nmap scan.

Option	Purpose
`-sV`	Attempts to ascertain version numbers of services running on open ports.
`-p`	Passes the comma separated list of ports held in the "targetports" variable to the switch for defining which ports to scan.
`-oA nmapscan`	Instructs Nmap to copy the results of the scan to an output file. The `A` option further stipulates that all 3 file formats (.gnmap, .nmap and .xml) should be produced and ensures that results can be viewed in a variety of visual formats.
`> nmapscan.txt`	Redirects the onscreen output to a .txt file called nmapscan.txt.

It was the developer's intention for the information obtained from the Nmap scans to be exported to the accompanying report documents at this point in the execution of the script. With this in mind, the Document module from the python-docx library was used to initialise two separate reports - one for a Management Report that would contain findings to report back to a potential customer, the other to contain further details about the findings and how they were obtained for the leaner. The developer also chose to create a working directory to contain all the output files for the penetration test being conducted. Figure 2.15 shows the lines of code that were initially added to the networkportscan function following the execution of the Nmap scan to achieve this:

```
#Create directory for learner files
print("Creating directory for learner files.")
now=datetime.now()
dirname=(f"{learnername} {targetipaddress} {now}")
os.mkdir(dirname)

#Change working directory
os.chdir(dirname)

#Initialise management report document
print("Creating the starter document for the Management Report.")
managementreport=Document()

#Add heading to management report
managementreport.add_heading(f"Network Infrastructure Penetration Test Report", level=0)
managementreport.add_paragraph(f"Target IP address: {targetipaddress}")

#Save the management report
managementreport.save("Management Report.docx")

#Initialise the learner report document
print("Creating the starter document for the Learner Report.")
learnerreport=Document()
learnerreport.add_heading(f"Network Infrastructure Penetration Test Report", level=0)
learnerreport.add_paragraph(f"Learner report for {learnername}'s network penetration test on {targetipaddress}.")

#Save the learner report
learnerreport.save("Learner Report.docx")
```

Figure 2.15: Creating a working directory and report documents in the networkportscan function.

The `datetime.now()` instruction was used to obtain the current time, whilst the `os.mkdir` instruction was used to create the directory with a name containing the learner's name, target IP address and current time. It can also be seen that the `os.chdir` instruction was used to move execution into the newly created directory. The "Document" object was initialised using the `=Document()` statement, a title (indicated by `level=0` in the argument passed to the `add_heading` instruction) and text added to each document, and the documents saved using the `.save` instruction of the Document module. After testing the functionality and the effect that these lines of code had on the flow of the script execution, the developer established that these lines of code were more effective inserted into the main program as part of the program prologue, ensuring that the folder and documents were not created multiple times.

Once the lines of code responsible for creating the Management and Learner Report documents had been relocated to the program prologue, it became necessary to initialise the documents before information could be added to them. This was achieved with the following line of code:

```
<report>=Document("<reportName.python-docx>")
```

At this point, a new heading was added to the report (using the `level=1` argument passed to the `add_heading` instruction) followed by some descriptive text. The developer created an empty table in the document using the following lines of code:

```
row=table.rows[0]

row.cells[0].text="Port number"

row.cells[1].text="Service name"
```

It was the developer's intention that the only information exported to this table was port numbers and service names, as suggested by the table column names specified. It was possible to obtain this information using one of the output files from the Nmap scan and some text manipulation with the following command:

```
cat nmapscan.nmap | grep 'open' | sed 's/\/tcp//g'| sed -e 's/\s\+/,/g' > services.txt
```

An explanation of the switches used in this Nmap command can be found in Table 2.6:

Table 2.6: Explanation of switches when santising the data to export to the report documents.

Option	Purpose
`cat`	Tool for reading file contents and strings
`'s/\/tcp//g'`	Removes the string "tcp" from the search results piped from the `cat` and `grep` commands.
`'s/\s\+/,/g'`	Replaces the white spaces from the output of the preceding `sed` command with commas.

The data to export was obtained with a `for` loop in the Python code, using the inserted comma as the delimiter for fields in the source file, before being inserted into the Management Report table. This was achieved with the following lines of code:

```
with open("services.txt") as f:
      for line in f:
            line=line.strip()
            currentline = line.split(",")
            portnumber=currentline[0]
            servicename=currentline[2]
            row=table.add_row()
            row.cells[0].text=portnumber
            row.cells[1].text=servicename
```

The Management Report was then saved before the Learner Report initialised using the same techniques that had been used with the Management Report. The developer intended to include information within the Learner Report relating to the construction of the commands used to scan the target IP address that would serve to educate the script user, and so the Learner Report was to contain three tables:

1. A table, described by the command used to enumerate open ports, containing the details of the switches used in the command and their purpose. The contents of this table were composed by the developer and inserted with a `row.cells[1].text="<cellText>"` statement.
2. A table, described by the command used to conduct the Nmap scan, containing the details of the switches used in the command and their purpose. The contents of this table were composed by the developer and inserted with a `row.cells[1].text="<cellText>"` statement.
3. A duplicate of the table created in the Management Report, created using the same `for` loop used for the Management Report.

The three tables were created using the same methods as the table in the Management Report and will not be duplicated here. The developer added one more line of code to this section of the networkportscan function that served to transform one of the Nmap output files into an easily readable format for further analysis, adding a line of text to the Learner Report to inform the user that this had been done and where to find the resulting report. This was achieved using the following lines of code (descriptive text has been shortened for brevity):

```
os.system("xsltproc nmapscan.xml -o nmapscan.html")
learnerreport.add_paragraph("The full report of open ports/services including the service version information can be found in the nmapscan.html file.”)
```

The developer concluded the function by saving the Learner Report using the `.save` operation from the Document module.

2.4.1.2 Functionality verification

The script's output to the display can be seen in Figure 2.16:

```
Obtaining list of open ports. Please wait ...
Running Nmap scan. Please wait ...
Strange read error from 192.168.10.1 (104 - 'Connection reset by peer')
Nmap scan complete.
Exporting port scanning results to Management Report.
Exporting port scanning results to Learner Report.
Converting Nmap scan results to easily readable HTML format.
```

Figure 2.16: Terminal display from the networkportscan function.

The files created by the networkportscan operations can be seen in Figure 2.17:

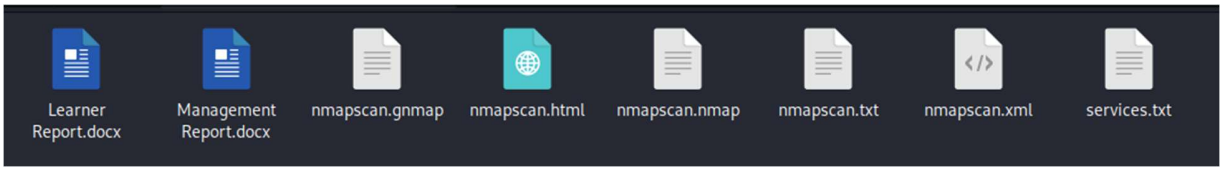

Figure 2.17: Files created by the networkportscan function.

The contents of the nmapscan files were created by native Linux tools and will not be included here, however the networkportscan function was successful in manipulating the contents of both Report documents and the services.txt file. Demonstrations of the result of this data manipulation can be seen in Appendix B.

2.4.2 vulnscan nested function

The desired logical flow for this function is demonstrated in the pseudo-code shown in Figure 2.18:

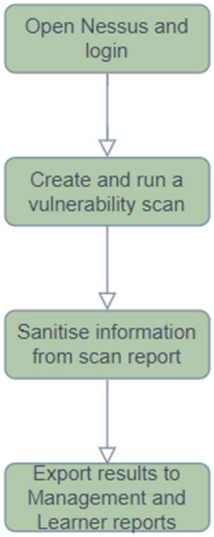

Figure 2.18: Pseudo-code for vulnscan function.

The details of the libraries and tools used for this function can be found in Table 2.7:

Table 2.7: Libraries and tools used in the vulnscan function.

Name of library	Version
webbrowser	3.11.18
os	3.11.18
python-docx	0.8.11
Name of tool	**Version**
cat	8.32
sed	4.8
tr	8.32

2.4.2.1 Code design

The developer composed a list of instructions that would be displayed to the user to guide them through the process of logging into the Nessus management console, creating a scan, running a scan, and downloading the resulting report. A web browser session, navigating directly to the login page for Nessus, was then invoked using the webbrowser library with the following line of code:

```
webbrowser.open_new("https://127.0.0.1:8834")
```

A line of code was then included with the purpose of specifying the file path for the Nessus report output and storing it in a variable, before the contents of the file could be manipulated for inclusion in the Management and Learner Reports. This information was taken from user input with the following line of code:

```
nessuscsv=input("Please enter the full file path (including the file extension) to the Nessus CSV file: ")
```

Knowing that the first line of the report CSV file would contain column headers, the developer used the `sed` command to remove the first line from the file. Furthermore, it was discovered that the results of the Nessus scan were enclosed within quotation marks. These were removed from the file contents with the `tr` command, allowing the contents to be parsed to the Management and Learner Reports in a suitably sanitised format. These two commands were processed in the script with the use of the `os.system` operation using the two lines of code shown below:

```
os.system(f'cat {nessuscsv} | sed "1d" > nessusstring.txt')
```

```
os.system(r'cat nessusstring.txt | tr -d "\"" > nessusresults.txt')
```

Once this content sanitisation had taken place, the developer used the same techniques as had been applied in the networkportscan (using the Document module functions and a `for` loop) to add a heading, some descriptive text, and a table containing the Nessus scan findings to both the Management and Learner Reports, before saving the updated report documents.

2.4.2.2 Functionality verification

The script's output to the display can be seen in Figure 2.19:

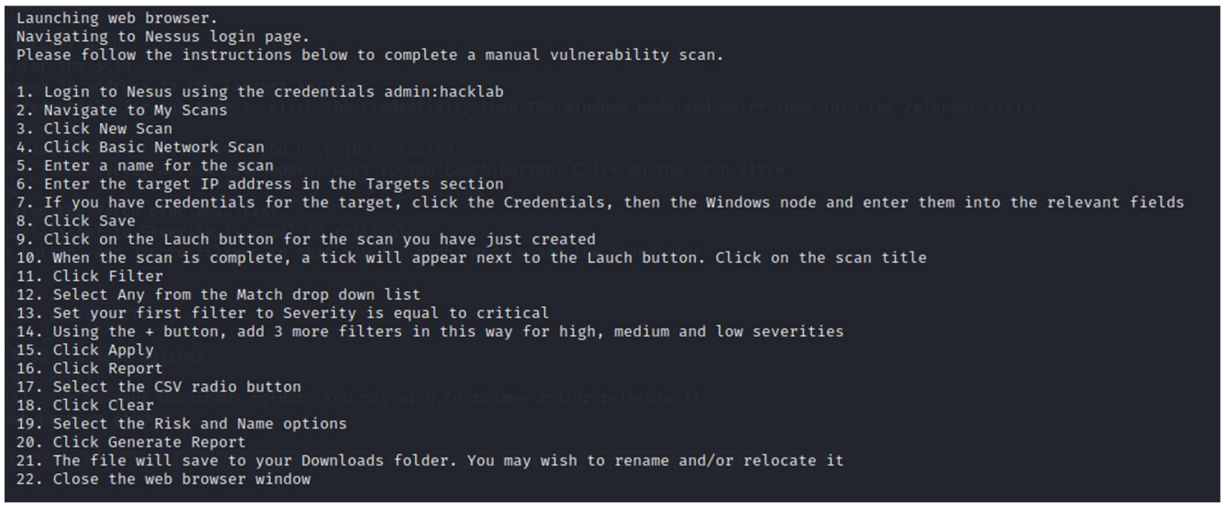

Figure 2.19: Terminal display from the vulnscan function.

The files created by the networkportscan operations can be seen in Figure 2.20:

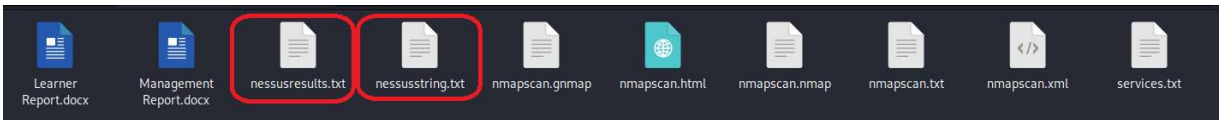

Figure 2.20: Files created by the vulnscan function.

Demonstrations of the file contents of the .txt files, as well as the newly created table and its contents from the Management and Learner reports can be seen in Appendix C.

2.4.3 Nested enumeration functions

The developer wrote a number of functions to complete specific tasks during the enumeration phase of the automated penetration test, the details of which can be found in Table 2.8:

Table 2.8: Nested enumeration functions in the networkpentest function.

Function name	Purpose
smtpusers	Attempt to enumerate valid SMTP users on the target machine using the smtp-enum-users script in Nmap.
pop3users	Attempt to enumerate valid POP3 users on the target machine using the pop3-brute script in Nmap.
ftpdir	Attempt to enumerate FTP directory contents using anonymous credentials.
smbscan	Attempt to enumerate the SMB service.
smbsharereport	Export information about discovered SMB shares to the Learner Report.
smbuserreport	Export information about discovered SMB users to the Learner Report.

2.4.3.1 smtpusers function

The desired logical flow for this function is demonstrated in the pseudo-code shown in Figure 2.21:

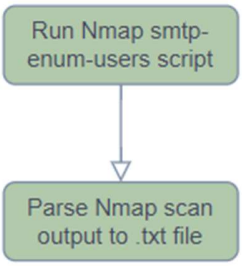

Figure 2.21: Pseudo-code for smtpusers function.

The details of the library used for this function can be found in Table 2.9:

Table 2.9: Library used in the smtpusers function.

Name of library	Version
nmap	0.7.1

2.4.3.1.1 Code design

When writing this function, the developer discovered that the output from a standard Nmap script scan did not offer an obvious method for parsing individual usernames if they were discovered. For this reason, the nmap library was chosen to provide the output for this function instead of invoking the Nmap command using an `os.system` operation. The differences between the outputs can be seen in Figure 2.22 and Figure 2.23 below:

```
└─$ nmap -p 25 192.168.10.1 --script=smtp-enum-users
Starting Nmap 7.92 ( https://nmap.org ) at 2024-04-21 08:41 EDT
Nmap scan report for 192.168.10.1
Host is up (0.0011s latency).

PORT   STATE SERVICE
25/tcp open  smtp
| smtp-enum-users:
|   root
|_  admin

Nmap done: 1 IP address (1 host up) scanned in 5.92 seconds
```

Figure 2.22: Output from Nmap smtp-enum-users script.

```
>>> nm.scan("192.168.10.1", "25", arguments="--script=smtp-enum-users")
{'nmap': {'command_line': 'nmap -oX - -p 25 --script=smtp-enum-users 192.168.10.1', 'scaninfo': {'tcp': {'method': 'connect', 'services': '25'}}, 'scanstats': {'timestr':
'Sun Apr 21 08:42:29 2024', 'elapsed': '5.89', 'uphosts': '1', 'downhosts': '0', 'totalhosts': '1'}}, 'scan': {'192.168.10.1': {'hostnames': [{'name': '', 'type': ''}], 'a
ddresses': {'ipv4': '192.168.10.1'}, 'vendor': {}, 'status': {'state': 'up', 'reason': 'syn-ack'}, 'tcp': {25: {'state': 'open', 'reason': 'syn-ack', 'name': 'smtp', 'prod
uct': '', 'version': '', 'extrainfo': '', 'conf': '3', 'cpe': '', 'script': {'smtp-enum-users': '\n  root\n  admin\n'}}}}}}
```

Figure 2.23: Output from smtp-enum-users script scan using the Nmap library in Python.

An Nmap PortScanner object was initialised, and a scan conducted using the following lines of code in the script:

```
nm=nmap.PortScanner()
```

```
nm.scan(f"{targetipaddress}", "25", arguments="--script=smtp-enum-users")
```

In order to extract the relevant information from the Nmap scan output, it was necessary to examine the data structure of the nm.scan object, which is shown in Figure 2.24:

Figure 2.24: Data structure of the nm.scan object from the smtp-enum-users script scan.

After confirming the data structure of the resulting nm.scan object, the developer was able to extract the required information (usernames) for an output file with the following line of code:

```
usernames_string=nm[f'{targetipaddress}'].tcp(25)['script']['smtp-enum-users']
```

The usernames could then be split into an array using the existing new line characters (`\n`) as the delimiting character for splitting the output. The line of code below as used to achieve this:

```
usernames=[username.strip() for username in usernames_string.strip().split('\n')]
```

Figure 2.25 shows the result of the string manipulation stored in the specified variable:

```
>>> print(usernames)
['root', 'admin']
```

Figure 2.25: Array containing SMTP usernames following string manipulation.

The developer was then able to use a `for` loop to iterate through the array and output each username to a separate line in a text file for later use during the password hacking stage of the automated penetration test using the following block of code:

```
for user in usernames:
    with open("smtpusers.txt", mode="at") as f:
        f.write(f"{user}\n")
```

2.4.3.1.2 Functionality verification

The successfully processed username information can be seen in Figure *2.26*:

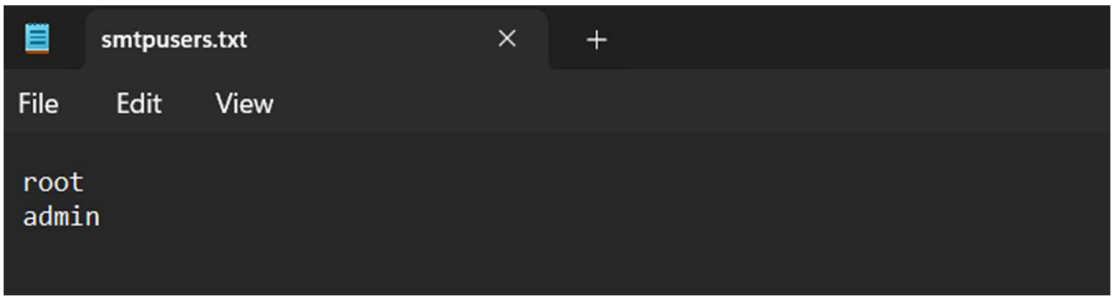

Figure 2.26: Output file from smtpusers function.

2.4.3.2 pop3users function

The desired logical flow for this function is demonstrated in the pseudo-code shown in Figure 2.27:

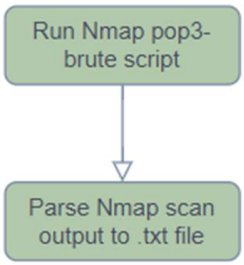

Figure 2.27: Pseudo-code for pop3users function.

The details of the library and tool used for this function can be found in Table 2.10:

Table 2.10: Library and tool used in the pop3users function.

Name of library	Version
nmap	0.7.1
Name of tool	**Version**
SecLists	N/A

2.4.3.2.1 Code design

The process for designing the display of this nested function was largely the same as that for the smtpusers function. One exception to this was in the amended line code required for the running of the Nmap scan, shown below:

```
nm.scan(f"{targetipaddress}", "110", arguments="--script=pop3-brute")
```

The output of this Nmap operation can be seen in Figure 2.28:

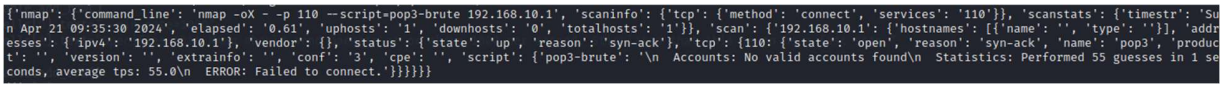

Figure 2.28: Output from pop3-brute script scan using Nmap library in Python.

It was necessary to examine the data structure of the nm.scan object to ensure that the string manipulation commands addressed the appropriate array field names. Figure 2.29 demonstrates the resulting data structure for this function:

Figure 2.29: Data structure of the nm.scan object from the pop3-brute script scan.

After confirming the data structure of the resulting nm.scan object, the developer was able to extract the desired information for an output file with the following line of code:

```
results_string=nm[f'{targetipaddress}'].tcp(110)['script']['pop3-brute']
```

The information returned with the string manipulation can be seen below:

```
\n  Accounts: No valid accounts found\n  Statistics: Performed 55 guesses in 1 seconds, average tps: 55.0\n  ERROR: Failed to connect.
```

Working within the limitations of the test system that the penetration test was being conducted against, it was not possible to determine what the output of the Nmap scripted scan would have contained if it had successfully enumerated any POP3 users. The developer instead chose to use an `if` statement to detect a string ("ERROR") that was returned in an unsuccessful scan to set a wordlist for use during the password hacking phase that would take place later in the automated penetration test. This was achieved using the following piece of code:

```
results_string=nm[f'{targetipaddress}'].tcp(110)['script']['pop3-brute']

if "ERROR" in (results_string):

    pop3userfile="/usr/share/wordlists/SecLists/Usernames/top-usernames-shortlist.txt"
```

2.4.3.2.2 Functionality verification

The code used in the pop3users function was successful in manipulating the Nmap pop3-brute scripted scan to set an appropriate word list for later use, as demonstrated in Figure 2.30:

```
>>> nm.scan('192.168.10.1', '110', arguments='--script=pop3-brute')
{'nmap': {'command_line': 'nmap -oX - -p 110 --script=pop3-brute 192.168.10.1', 'scaninfo': {'tcp': {'method': 'connect', 'services': '110'}}, 'scanstats': {'timestr': 'Sun Apr 21 10:14:48 2024', 'elapsed': '0.49', 'uphosts': '1', 'downhosts': '0', 'totalhosts': '1'}}, 'scan': {'192.168.10.1': {'hostnames': [{'name': '', 'type': ''}], 'addresses': {'ipv4': '192.168.10.1'}, 'vendor': {}, 'status': {'state': 'up', 'reason': 'syn-ack'}, 'tcp': {110: {'state': 'open', 'reason': 'syn-ack', 'name': 'pop3', 'product': '', 'version': '', 'extrainfo': '', 'conf': '3', 'cpe': '', 'script': {'pop3-brute': '\n  Accounts: No valid accounts found\n  Statistics: Performed 35 guesses in 1 seconds, average tps: 35.0\n  ERROR: Failed to connect.'}}}}}}
>>> results_string=nm[f'192.168.10.1'].tcp(110)['script']['pop3-brute']
>>> if "ERROR" in (results_string):
...     pop3userfile="/usr/share/wordlists/SecLists/Usernames/top-usernames-shortlist.txt"
...
>>> print(results_string)

  Accounts: No valid accounts found
  Statistics: Performed 35 guesses in 1 seconds, average tps: 35.0
  ERROR: Failed to connect.
>>> print(pop3userfile)
/usr/share/wordlists/SecLists/Usernames/top-usernames-shortlist.txt
```

Figure 2.30: Demonstration of suitable word list set dependent on Nmap pop3-brute script scan results.

2.4.3.3 ftpdir function

The desired logical flow for this function is demonstrated in the pseudo-code shown in Figure 2.31:

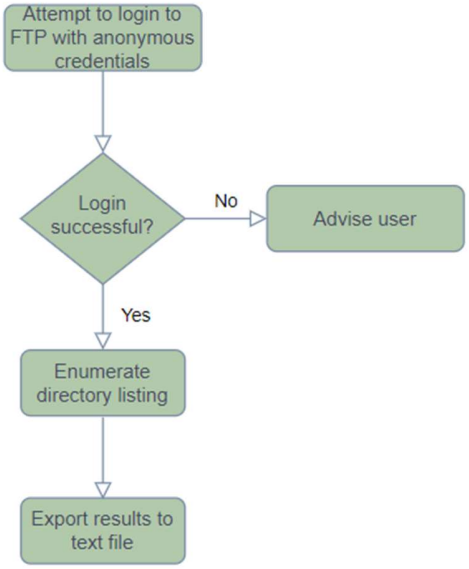

Figure 2.31: Pseudo-code for ftpdir function.

The details of the libraries used for this function can be found in Table 2.11:

Table 2.11: Libraries used in the ftpdir function.

Name of library	Version
ftplib	3.11.18
python-docx	0.8.11

2.4.3.3.1 Code design

The first stage of the ftpdir function was intended to test whether the FTP server was configured to allow "anonymous" logins. With the knowledge that the value for the password was typically unimportant in these setups (Weidman, 2014), the developer initialised an FTP session using the following lines of code, utilising the functionality from the ftplib library:

```
ftp=FTP()

ftp.connect(f"{targetipaddress}", 21)

login=ftp.login("anonymous", "anonymous")
```

A successful login message can be seen in Figure 2.32:

```
>>> ftp=FTP()
>>> ftp.connect(f"192.168.10.1", 21)
'220-Wellcome to Home Ftp Server!\n220 Server ready.'
>>> login=ftp.login("anonymous", "anonymous")
>>> print(login)
230 User Anonymous logged in.
```

Figure 2.32: Successful login message for FTP session.

The developer was then able to implement an `if` statement based on the message returned from the server, where the intended logical execution was to enumerate the directory listing and export the results to a text file (from which the results could be exported to the Learner Report using the Document module) in the event of a successful login, or simply write an informative line to the user in the event of an unsuccessful login. This was achieved with the following block of code:

```
#Check for successful login
if "logged in" in login:
    #Enumerate files and export to file if successfully logged in
    files=ftp.nlst()
    for file in files:
        with open("ftplisting.txt", mode="at") as f:
            f.write(f"{file}\n")

#Print failure message to screen if login unsuccessful
else:
    print("Anonymous login unsuccessful.")
```

Figure 2.33: Checking for successful login to the FTP server and actions to take.

The ftpdir function was concluded with a line of code to close the open FTP session, shown below:

```
ftp.close()
```

2.4.3.3.2 Functionality verification

The successfully processed directory listing information can be seen in Figure 2.34:

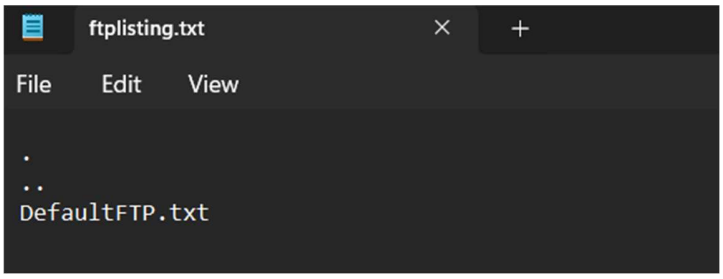

Figure 2.34: Output file from ftpdir function.

2.4.3.4 smbscan function

The desired logical flow for this function is demonstrated in the pseudo-code shown in Figure 2.35:

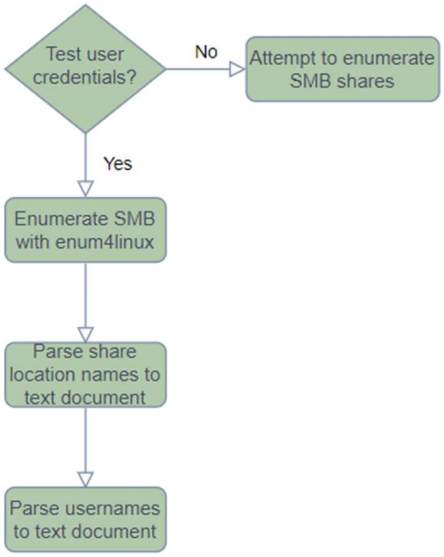

Figure 2.35: Pseudo-code for smbscan function.

The details of the libraries and tools used for this function can be found in Table 2.12:

Table 2.12: Libraries and tools used in the smbscan function.

Name of library	Version
os	3.11.18
getpass	3.11.18
Name of tool	**Version**
smbclient	4.16.3-Debian
enum4linux	0.9.1
cat	8.32
grep	3.7
cut	8.32
tr	8.32

2.4.3.4.1 Code design

The developer considered that enumeration of SMB yields better results if a valid user has been provided to a penetration tester to use during testing. The developer also acknowledged that it is considered best practice to mask password input in the terminal using the `getpass` library (Heinz, 2021). With these points in mind, the following lines of code were initially added to the top of the smbscan function code:

```
testcreds=input("Have you been provided with credentials for the target machine? Y/N ")

testcreds=testcreds.lower()
```

```
if testcreds=="y":
     testuser=input("Please enter the test username: ")
     testuserpasswd=getpass.getpass(prompt="Please enter the test user password: ")
```

After testing the functionality and the effect that these lines of code had on the flow of the script execution, the developer established that these lines of code were more effective inserted into the main program as part of the program prologue, along with the prompts to accept a name and target IP address. This ensured that this information could be used by multiple functions without the user being prompted for it multiple times (in the event that they wish to run multiple stages of either the network or web application penetration tests from their relative sub-menus).

The developer was then able to use the "testcreds" variable in an `if` statement that would dictate whether the script simply attempted to enumerate SMB share location names (in the instance that no test credentials were available) or to attempt to fully enumerate SMB on the target machine. The tools used for this purpose were `smbclient` (for the former scenario) and `enum4linux` (for the latter scenario), and were handled by the `os.system` operation. This execution flow was achieved with the following block of code:

```
if testcreds=="n":
     os.system(f"smbclient -L {targetipaddress} --password=test > smbsharenames.txt")
     elif testcreds=="y":
     os.system(f"enum4linux -a -u {testuser} -p {testuserpasswd} {targetipaddress} > smbenum.txt")
```

A breakdown of the command options and their purpose can be seen in Table 2.13:

Table 2.13: Explanation of command line switches for enum4linux.

smbclient option	Purpose
`-L`	Instructs `smbclient` to list any shares discovered.
`--password`	Passes a password to `smbclient` if the server prompts for one.
enum4linux option	**Purpose**
`-a`	Instructs `enum4linux` to perform all simple tests that it can perform
`-u`	Specifies the username for `enum4linux` to use.
`-p`	Specifies the password for the username provided.

No further processing was performed on the theoretical contents of the smbsharenames.txt file as it would not be produced under the test conditions in use.

The developer then parsed any instances of SMB share location names from the smbenum.txt file using the following chain of commands:

```
cat smbenum.txt | grep "//{targetipaddress}/" | cut -d "/" -f 4 > smbshares_string.txt
```

A breakdown of this string manipulations is as follows:

1. Read the contents of the smbenum.txt file.
2. Search for the string "//{targetipaddress}/" (192.168.10.1 in this instance).
3. Use the "/" character as a delimiter and display the 4th field.
4. Output the results to a new file (smbshares_string.txt).

The contents of the smbshares_string.txt file were then subjected to further string manipulation with the following command, where the `tr` tool is used to replace any instances of tabulation (specified with "`\\t`") with a comma character:

```
cat smbshares_string.txt | tr "\\t" "," > smbshares.txt
```

These two commands were executed by Python using the `os.system` operation. It was necessary to process these as two separate instructions as they contained a mixture of an f-string and a raw string. The finalised block of code for this string manipulation was achieved with the following block of code:

```
filepath=f'cat smbenum.txt | grep "//{targetipaddress}/" | cut -d "/" -f 4 > smbshares_string.txt'

os.system(filepath)

filepath=r'cat smbshares_string.txt | tr "\\t" "," > smbshares.txt'

os.system(filepath)
```

The developer concluded the smbscan function with another interrogation of the smbenum.txt file to find a list of valid domain users discovered during enumeration. The following command was used to obtain this information (by passing to the `os.system` instruction):

```
cat smbenum.txt | grep -E "Domain Users.*member" | cut -d ":" -f 4 | cut -d "\\" -f 2 > smbusers.txt
```

A breakdown of this string manipulations is as follows:

1. Read the contents of the smbenum.txt file.
2. Search lines containing both "Domain Users" and "member" string values.
3. Use the ":" character as a delimiter and display the 4th field.
4. Use the "\" character as a delimiter and display the 2nd field.
5. Output the results to a new file (smbusers.txt).

The two new files generated would be used for exporting information to the Learner Report later in the program execution.

2.4.3.4.2 Functionality verification

The files created by the smbscan operations can be seen in Figure 2.36:

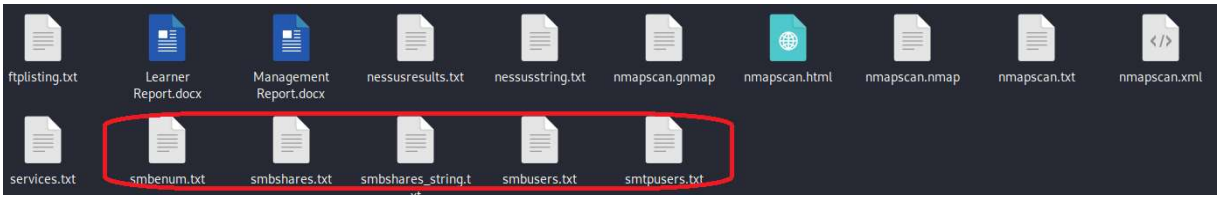

Figure 2.36: Files created by the smbscan function.

Demonstrations of the file contents of the .txt files can be seen in Appendix D.

2.4.3.5 smbsharereport function

The desired logical flow for this function is demonstrated in the pseudo-code shown in Figure 2.37:

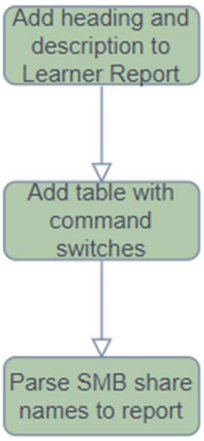

Figure 2.37: Pseudo-code for smbsharereport function.

The details of the library used for this function can be found in Table 2.14:

Table 2.14: Library used in the smbsharereport function.

Name of library	Version
python-docx	0.8.11

2.4.3.5.1 Code design

The developer intended this function to create a table in the Learner Report containing the commands and switches used in order to obtain the list of SMB share location names, as well as a bullet list of the SMB share location names (found in the smbshares.txt file created by the smbscan function). The methods for generating the content for the Learner Report from this function was the same as that for the networkportscan function and will not be replicated here.

2.4.3.5.2 Functionality verification

A demonstration of the output created by the smbsharereport function can be seen in Figure 2.38:

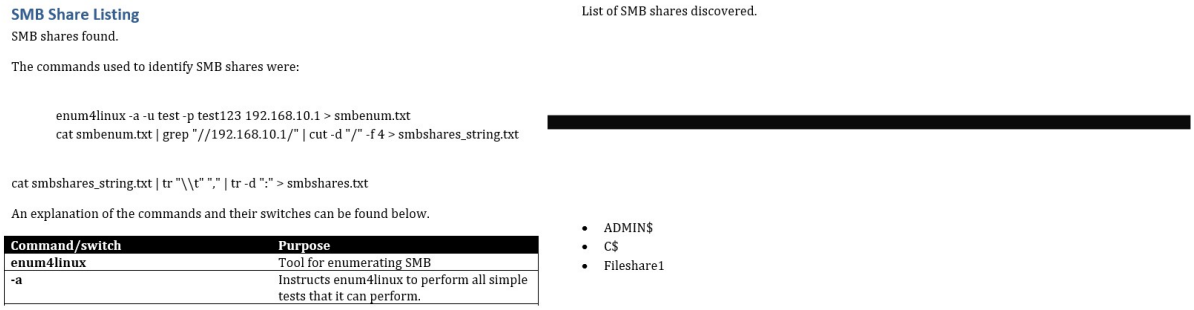

Figure 2.38: Output created by the smbsharereport function.

2.4.3.6 smbuserreport function

The desired logical flow for this function is demonstrated in the pseudo-code shown in Figure 2.39:

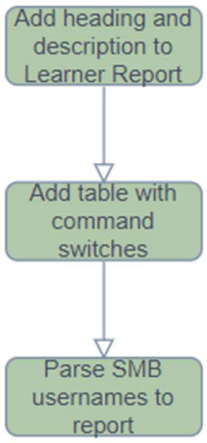

Figure 2.39: Pseudo-code for smbuserreport function.

The details of the library used for this function can be found in Table 2.15:

Table 2.15: Library used in the smbuserreport function.

Name of library	Version
python-docx	0.8.11

2.4.3.6.1 Code design

The developer intended this function to create a table in the Learner Report containing the commands and switches used in order to obtain the list of SMB usernames, as well as a bullet list of the SMB usernames (found in the smbusers.txt file created by the smbscan function). The methods for generating the content for the Learner Report from this function were the same as those used in the networkportscan function and will not be replicated here.

2.4.3.6.2 Functionality verification

A demonstration of the output created by the smbuserreport function can be seen in Figure 2.40:

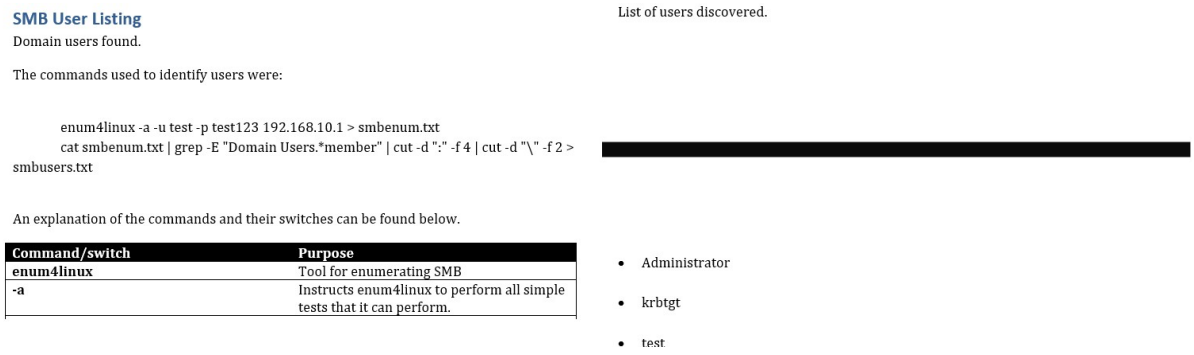

Figure 2.40: Output created by the smbuserreport function.

2.4.4 hydrasmb nested function

The desired logical flow for this function is demonstrated in the pseudo-code shown in Figure 2.41:

Figure 2.41: Pseudo-code for hydrasmb function.

The details of the libraries and tools used for this function can be found in Table 2.16:

Table 2.16: Libraries and tools used in the hydrasmb function.

Name of library	Version
os	3.11.18
python-docx	0.8.11
Name of tool	**Version**
hydra	9.3
SecLists	N/A
cat	8.32
grep	3.7
tr	8.32
cut	8.32

2.4.4.1 Code design

The developer sought to use a native Kali Linux tool to attempt to brute force the passwords for any users discovered when enumerating the SMB instance. This was achieved using the following line of code (which was then handled by the `os.system` instruction):

```
hydra -L smbusers.txt -P ../supersmalldictionary.txt
smb://{targetipaddress} -o bruteforced.txt
```

A breakdown of the command options and their purpose can be seen in Table 2.17:

Table 2.17: Explanation of command line switches for Hydra.

Option	Purpose
`-L`	Passes a wordlist containing usernames to Hydra.
`-P`	Stipulates that Hydra should use the values contained within a file to use with its brute force attempts.
`smb://`	Specifies the protocol for attempting to brute force the username and password combinations against.
`-o`	Directs any successfully cracked passwords to a text file with the specified name.

A very small file containing only two password possibilities was used for efficiency during the development of the proof of concept.

The developer then parsed any instances of successfully cracked passwords from the bruteforced.txt file using the following chain of commands (which was handled by the `os.system` operation within the script):

```
cat bruteforced.txt| grep "login" | tr -s " " | tr -d ":" | tr " " "," 
| cut -d "," -f 5,7 > smbcreds.txt
```

A breakdown of this string manipulations is as follows:

1. Read the contents of the bruteforced.txt file.
2. Search for the string "login".

3. Remove any instances of multiple white spaces.
4. Remove any colon characters.
5. Replace any spaces with the comma character.
6. Use the "," character as a delimiter and display the 5th and 7th fields.
7. Output the results to a new file (smbcreds.txt).

The developer then intended to create the following content in the report documents:

- A table in the Management Report containing the credentials discovered by Hydra (obtained from the bruteforced.txt file).
- A table in the Learner Report containing the commands and switches used in order to obtain the list of credentials.
- A table in the Learner Report containing the credentials discovered by Hydra (obtained from the bruteforced.txt file).

The methods for generating this content from this function were the same as those used in the networkportscan function and will not be replicated here.

2.4.4.2 Functionality verification

The file created by the hydrasmb operations can be seen in Figure 2.42:

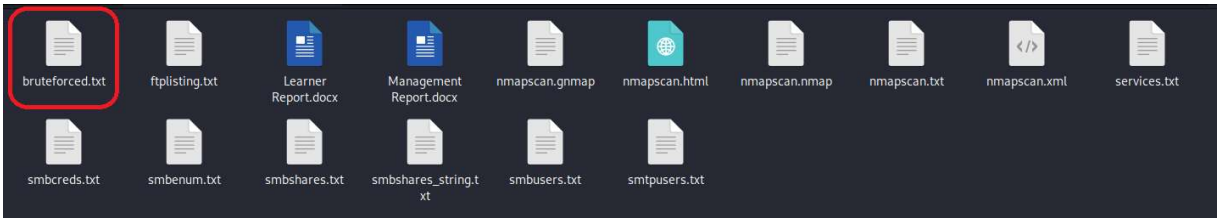

Figure 2.42: Files created by the hydrasmb function.

Demonstrations of the file contents of the bruteforced.txt file, as well as the contents generated into the Management and Learner Reports can be seen in Appendix E.

2.4.5 Constructing the networkpentest function

A more detailed pseudo-code for the construction of the networkpentest function, which incorporates the nested functions and demonstrates the logical flow of the function, can be found in Appendix F.

The details of the libraries used for the main body of this function can be found in Table 2.18:

Table 2.18: Libraries used in body of the networkpentest function.

Name of library	Version
os	3.11.18
python-docx	0.8.11
webbrowser	3.11.18

2.4.5.1 Code design

It was necessary for the developer to use a combination of local function calls and independent lines of code throughout this function. The first action to take involved a single line of code invoking the networkportscan function. In order to progress the automated penetration test, the developer

considered it appropriate to offer the script user the choice of whether or not to conduct a vulnerability scan – this was owing to the length of time that Nessus scans can take to complete. This was achieved with an `if` statement, with the qualifying condition based on user input, demonstrated in Figure 2.43, which shows the vulnscan function being called in the event that the user deems a vulnerability scan to be appropriate.

```
print("""
Port scanning has completed. The next stage of a penetration test is to scan the target for vulnerabilities.
Unfortunately, automated vulnerability scanning is not possible with the available software.
You may run a scan manually. If you choose to do so, instructions will be provided.
""")
print("NOTE: NESSUS SCANS CAN TAKE A CONSIDERABLE AMOUNT OF TIME TO COMPLETE\n")
vulngo=input("Do you wish to conduct a manual vulnerability scan? Y/N ")
vulngo=vulngo.lower()
if vulngo=="y":
    vulnscan()
```

Figure 2.43: `if` statement to call vulnscan function or continue.

The developer's design for the next section of the networkpentest function incorporated a search of the output from the Nmap scans (completed during the networkportscan function execution) for common services. It was intended that the script would then write a note to the Learner Report wherever a service was located (beneath a general header for the section), and that further enumeration be performed on those services later in the script's execution. The developer considered that the flow of the script should first output the common services found, before the enumeration of those services was added to the Learner Report, resulting in a report that maintains clear segmentation between its sections. In order to achieve this desired flow of execution, the developer used a combination of switch-case statements and variables that could be set to indicate the presence of a service when it is discovered. The variables used for this purpose proved particularly important for services that could be identified on multiple ports (e.g. FTP). The layout of the switch-case statements, based on the output of the services.txt file generated from the networkportscan function, can be seen in Appendix G.

With the knowledge that the variables used for the presence of a service would be used by other `if` statements later in the networkpentest function, it was necessary to initialise these variables with empty values. This ensured that the `if` statements in question did not error in occurrences where a service had not been discovered. These two contrasting outcomes are demonstrated in Figure 2.44:

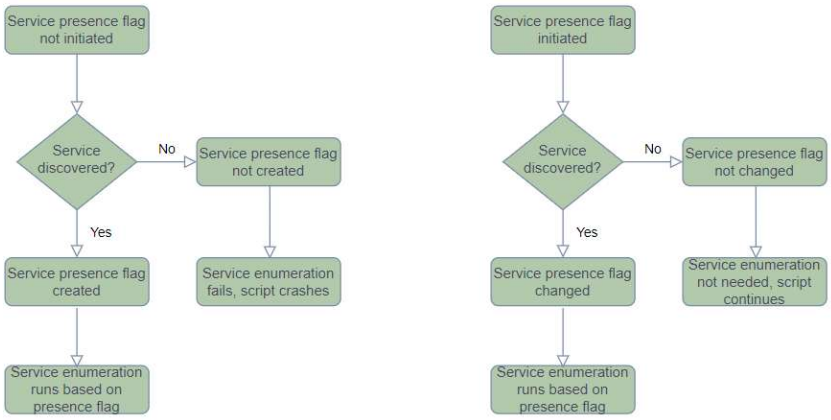

Figure 2.44: Flow charts showing script actions for variable initialisation scenarios.

These initialised empty variables, show in Figure 2.45, were included at the top of the networkpentest function block of code, ensuring that the script maintained a consistent structure, which would support any future development to include extended functionality.

```
#Initialise flag variables for use with services that can occupy multiple ports (for prevention of duplicate lines in learner report)
ftpflag=0
smbflag=0
httpflag=0

#Initialise flag variables for use with services for Hydra brute forcing
smtpbrute=0
pop3brute=0
```

Figure 2.45: Initialisation of empty variables.

In instances where services could be discovered by the presence of more than one port number, the enumeration of the service could not be performed within the switch-case statement alone. As such, these instances were handled by a series of if statements executed after the completion of the switch-case statement, shown in Figure 2.46 (code shortened for clarity):

```
case "20":
    ftpflag=1

case "21":
    ftpflag=1

if ftpflag==1:
print(Fore.CYAN+"FTP FOUND!! Attempting to enumerate with FTP client. Please wait...")
ftpdir()
```

Figure 2.46: Enumeration `if` statements for services that could run on multiple ports.

The developer chose to invoke the `.save` instruction of the python-docx library at this point in the script execution – this was implemented to ensure that the file contents were saved regardless of whether the remainder of the nested functions within the networkpentest function executed or not.

There followed several blocks of code whose responsibility it was to export the information obtained during the service enumeration to the Learner Report. These blocks run independently from one another and in instances where other functions were not called it was necessary to open those blocks with the initialisation of the Learner Report document and conclude them with the `.save` instruction to ensure the data exported was saved appropriately. The first in the sequence concerns the exportation of the FTP information and uses a conditional `if-else` statement based on the presence of the ftplisting.txt file created by the ftpdir function. The developer then implemented a `for` loop to export the information to the Learner Report if the file existed. The developer included an `else` clause to this particular statement to cover the eventuality that an FTP instance had been discovered but that anonymous login had been unsuccessful, in which case it was a requirement that the script write an informative line to the Learner Report. This if-else statement can be seen in Figure 2.47:

```
learnerreport=Document("Learner Report.docx")
if os.path.exists("ftplisting.txt"):
    #Add FTP directory listing findings to learner report
    print("Exporting FTP enumeration results to Learner Report.")
    learnerreport.add_heading("FTP Directory Listing", level=1)
    learnerreport.add_paragraph("List of files and directories discovered. Includes current (.) and parent (..) directories.")
    with open("ftplisting.txt", mode="r") as f:
        for line in f:
            learnerreport.add_paragraph(f"{line}", style="List Bullet")
else:
    if ftpflag==1:
        learnerreport.add_heading("FTP Directory Listing", level=1)
        learnerreport.add_paragraph("Possible FTP instance discovered. Anonymous login not successful.", style="List Bullet")

#Save the learner report
learnerreport.save("Learner Report.docx")
```

Figure 2.47: if-else statement for exporting FTP directory information.

The second of these blocks of code implemented the exportation of SMTP user information to the Learner Report and is achieved using a simpler `if` statement based on the existence on the smtpusers.txt file generated by the smtpusers function. The information is exported using the same `for` loop structure applied in the block of code used for the FTP information exportation and will not be replicated here.

The penultimate block of code in this section was responsible for exporting information regarding SMB share location names to the Learner Report. The developer had initially designed the code for the smbsharereport function in full at this point but concluded that the repeated lines necessary for designing a table containing the command line tools and their switches contributed to an untidy layout for the networkpentest function. These lines of code were therefore converted to a function in their own right, resulting in this `if` statement simply calling the smbsharereport function based on the existence of the smbshares.txt file generated by the smbscan function.

The final block of these blocks of code encompassed two functionalities:

1. The exportation of the SMB users information to the Learner Report.
2. The calling of the hydrasmb function.

These two functionalities were included within a single if statement (despite originating from different stages of the automated penetration test) as their execution relies on the same dependency being satisfied – the existence of the smbusers.txt file generated by the smbscan function. As with the smbsharereport function, the developer originally wrote the code for both of these actions in full but converted them to stand-alone functions later in the development of the script. Both of these functions are executed with the code shown in Figure 2.48:

```
#Add SMB findings to learner report
#Add SMB share findings to learner report
if os.path.exists("smbshares.txt"):
    print("Exporting SMB share results to Learner Report.")
    smbsharereport()

#Add SMB share findings to learner report
if os.path.exists("smbusers.txt"):
    print("Exporting SMB user results to Learner Report.")
    smbuserreport()

    #Run Hydra for SMB users
    print("SMB USERS FOUND!! Attempting to crack passwords with Hydra. Please wait...")
    hydrasmb()
```

Figure 2.48: Single if statement for SMB user result exportation and running Hydra.

The developer decided that the final action to be taken by the networkpentest function would be to open a browser window containing links to all the files that had been generated during the script execution. With the knowledge that the folder name would be different every time the script was run (as it uses a unique date and time as part of the naming convention), the `os.cwd` operation was used to obtain the working directory. This value could then be passed to the `webbrowser.open` instruction to open the browser window. This was achieved with the following lines of code:

```
dirpath=os.getcwd()

webbrowser.open(f"{dirpath}")
```

The developer concluded the function with a `quit()` instruction to ensure the program exited gracefully.

2.4.5.2 Functionality verification

The resulting display from the script execution, and the content added to the Learner report can be seen in Appendix H.

2.5 MAIN PROGRAM

The desired logical flow for this function is demonstrated in the pseudo-code shown in Figure 2.49:

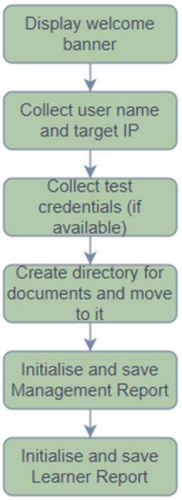

Figure 2.49: Pseudo-code for main function.

The details of the libraries used for the main function can be found in Table 2.19:

Table 2.19: Libraries used in the main function.

Name of library	Version
os	3.11.18
python-docx	0.8.11

2.5.1 Code design

The finalised main program was composed largely of elements that have already been outlined in this report, and so will not be discussed in detail here. The only new components the developer added to the script at this point was a welcome banner that would be displayed to the user and a function call to the mainmenu function.

2.5.2 Functionality verification

Figure 2.50 shows the resulting display generated from the main function:

```
Welcome to RunMeThru, an automated penetration testing tool that will guide you through the process of running your own penetration test.
This program uses established methodologies to perform basic tests on a target you will specify.
A simple report will be produced that you could present to a client (Management Report.docx).
There will also be a report produced for you to see the tools, commands and switches that have been used throughout the test (Learner Report.docx).
Suggestions will be made in the report for further areas of research you may wish to conduct or elements of the test you might investigate further.
This script is written in Python and can easily be adapted to make use of other/additional tools. Why not try it out?
Enjoy!

WARNING: THIS TOOL IS FOR EDUCATIONAL PURPOSES ONLY - NEVER SCAN ANYTHING WITH IT UNLESS YOU NOT HAVE WRITTEN PERMISSION TO DO SO

Enter your name: JKP
Thank you. This will be used on the learner report produced during the testing process.
Enter the target IP address: 192.168.10.1
Have you been provided with credentials for the target machine? Y/N y
Please enter the test username: test
Please enter the test user password:
Creating directory for learner files.
Creating the starter document for the Management Report.
Creating the starter document for the Learner Report.
```

Figure 2.50: Display generated by the main program.

2.6 ENHANCING THE SCRIPT

The developer considered that, given its intended role as a training tool, it would be appropriate to add some enhancements to the script to increase its appeal to the user. This was achieved with the use of additional elements to the finalised script:

- Application of ASCII art to the menu headings.
- Application of colour to any text displayed to the screen.

2.6.1 ASCII art

The design for the program welcome screen and three menus was generated using the website below:

https://patorjk.com/software/taag/#p=display&f=Graffiti&t=Type%20Something%20

2.6.1.1 Code design

Once generated, the developer was able to simply copy the ASCII art from the website and insert it as raw multi-line text, as demonstrated in Figure 2.51:

```
print(r"""
 ____              __  __      _____ _   _ ____
|  _ \ _   _ _ __ |  \/  | ___|_   _| | | |  _ \ _   _
| |_) | | | | '_ \| |\/| |/ _ \ | | | |_| | |_) | | | |
|  _ <| |_| | | | | |  | |  __/ | | |  _  |  _ <| |_| |
|_| \_\\__,_|_| |_|_|  |_|\___| |_| |_| |_|_| \_\\__,_|

""")
```

Figure 2.51: Code to display welcome banner in ASCII art.

2.6.1.2 Functionality verification

Figure 2.52 shows an example of the resulting display generated:

Figure 2.52: Welcome banner displayed in ASCII art.

2.6.2 Colours

The details of the library used to provide text colour formatting options can be found in Table 2.20:

Table 2.20: Library to apply colour options to display text.

Name of library	Version
colorama	0.4.5

2.6.2.1 Code design

Table 2.21 shows the colour scheme that the developer chose to apply to the output:

Table 2.21: Colour scheme applied to finalised script.

Message type	Colour
Warnings	Red
Prompt for user input	Magenta
Menus/response to user input	White
Progress information	Cyan
Information found	Yellow

Colours were applied to `print()` or `input()` statements used in the script using the following code structure:

```
print(Fore.<COLOUR>+ "<string>")
```

The developer chose to apply this construct to all `print()` and `input()` statements explicitly, regardless of whether the chosen colour was already in effect at the point in the script execution. This was done to provide consistency and support for any future development of the script in ensuring that the desired colour scheme is applied, regardless of preceding code execution. An example of this can be seen in Figure 2.53:

```
#Store credentials for authenticated scanning if available
testcreds=input(Fore.MAGENTA+"Have you been provided with credentials for the target machine? Y/N ")
testcreds=testcreds.lower()
if testcreds=="y":
    testuser=input(Fore.MAGENTA+"Please enter the test username: ")
    testuserpasswd=getpass.getpass(prompt="Please enter the test user password: ")
```

Figure 2.53: Explicit declaration of colour scheme in consecutive lines of code.

The developer found it necessary to reset the colour styling once in the script. This was achieved with the following line of code:

```
print(Style.RESET_ALL)
```

This was inserted as part of the block of code that included the call to the hydrasmb function and was required to ensure that the display output from Hydra (which could not be suppressed) would be unaffected by the colorama library operations.

2.6.2.2 Functionality verification

Figure 2.54 shows an example of the resulting display generated:

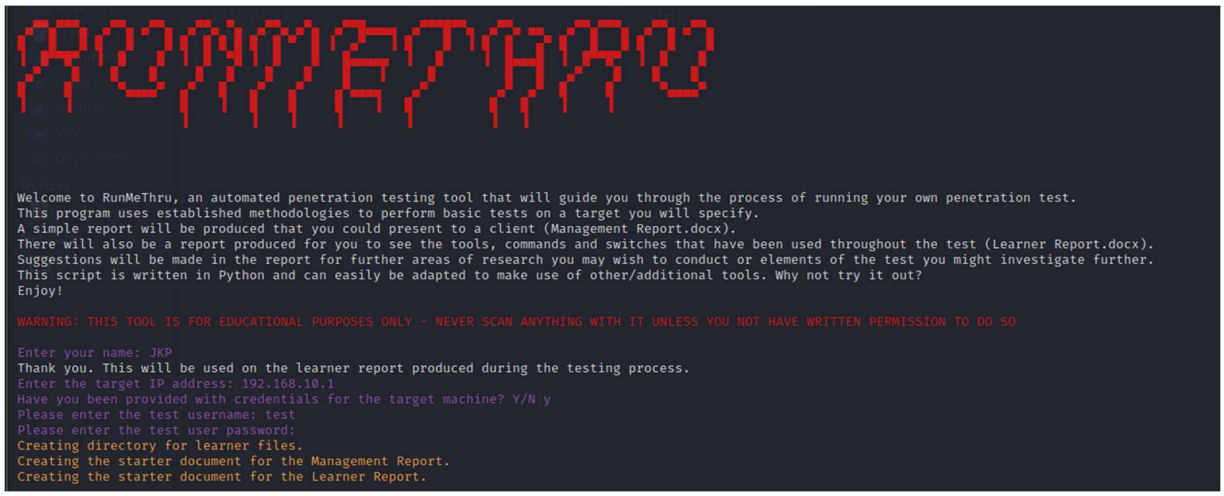

Figure 2.54: Coloured output display.

2.7 SCRIPT STRUCTURE

The finalised script took on the overall structure shown in Figure 2.55, which falls in line with best practice (Novotny, 2024):

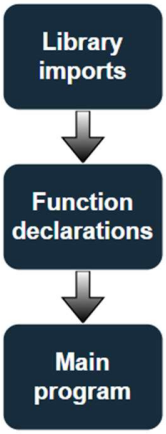

Figure 2.55: Overarching structure of the finalised script.

The subprocess, os, getpass, nmap, and webbrowser libraries were imported using the import statement without the need to specify the modules to include in the import. The remaining libraries required particular modules to be specified within the import statement in order for the desired functionality to be delivered. These are outlined in Table 2.22:

Table 2.22: Specific libraries required for the import process.

Library name	Module name
python-docx	Document
ftplib	FTP
datetime	datetime
colorama	init; Fore; Style

Furthermore, the colorama library had to be initialised before its functionality could be accessed. This was achieved with the following line of code, that was be entered directly beneath the library and library import statements:

```
init()
```

Figure 2.56 demonstrates the lines of code used at the beginning of the script to import these libraries and relevant modules:

```
import subprocess
import os
from docx import Document
import nmap
from ftplib import FTP
from datetime import datetime
import getpass
import webbrowser
from colorama import init
from colorama import Fore, Back, Style
init()
```

Figure 2.56: Library/library import statement.

Functions were defined using the `def <functionname>()` instruction and appropriate indentation. Each function was clearly separated with the use of a header and description, enclosed in hash (#) symbols.

3 DISCUSSION

3.1 GENERAL DISCUSSION

This paper set out with the aim of documenting the development of a Python script capable of conducting an automated penetration test intended for use as a training tool. It was clear to the developer at the beginning of the project that a functional script that covered both network and web application penetration tests would be outside of the scope of a single report, and as such the focus of the script design was on the development of a network penetration test function. Furthermore, it was clear that a script capable of providing coverage for every possible outcome in a network penetration test would require extensive testing with multiple differently configured test environments. With that in mind, the script that has been produced serves effectively as a proof of concept only (limited by the services available on the test server that the developer chose to use) and should not be considered complete without further significant work to extend its capabilities.

When considering the possibility of tools to incorporate into the script, the developer was fortunate to have an extensive library of tools that would facilitate the completion of many of the tasks required to undertake a penetration test built-in to the Kali virtual machine that was used when developing the program. The majority of the tools used were able to produce an acceptable result that could be further manipulated to assist with the creation of the reports. Furthermore, their widespread distribution and use within the penetration testing community provided assurances that recommendations could be made for learners to seek guidance about their customisation possibilities in official and readily available documentation (Roberts, 2024). This notwithstanding, the developer discovered two aspects of the script in which automation could be improved with the use of custom-built tools:

- It was ascertained that a fully automated Nessus vulnerability scan from the command line was only possible with Nessus instances that had additional commercial components that were not available in the test environment. Furthermore, the automation of a browser-based scan creation and initialisation would be possible only with the installation of further Python and operating system components, or with the development of a custom tool. As such, the developer concluded that the investment of time required to achieve this automation exceeded the requirements to satisfy the proof of concept in the larger project.
- Erickson (2008) describes the process for designing a custom tool to conduct an online dictionary attack as being a simple one and as it was not possible to suppress the terminal output from Hydra during the password cracking stage of the penetration test, the creation of a custom tool to complete this task instead of using a pre-existing one could address this minor aesthetic issue.

The developer considered it highly likely that more opportunities for creating custom tools would present themselves should the script be developed past this initial proof-of-concept phase.

The script, in its current state and when used against the chosen test environment, has achieved the basic aim that the developer had intended in producing two separate reports in a largely automated manner – one that would be produced to a customer, the other that contained information that would

be beneficial to a user's education in the tools and processes used when conducting a network penetration test. The resulting reports, whilst structured in a clear and concise manner, have a rustic appearance. The library used to create the reports (python-docx) is a popular one for working with Microsoft Word documents in Python (Das, 2023) and its functionality is extensive. The developer believed that the formatting of both the Learner and Management reports could be refined to produce more professional looking documents but concluded that this was a minor matter of aesthetics and that further development in the script for this purpose went beyond the requirements to satisfy the proof of concept.

The logical execution of the script could aid a learner in their understanding of the importance of using a methodology when conducting a penetration test in order to achieve a structured and comprehensive outcome. The ability for the user to conduct entire or part-penetration testing and the choice to use a combination of progress messages displayed to the screen, detailed information about the usage of tools, and script enhancements to increase the appeal of the program provided a learner with ample opportunity to recognise learning development possibilities when gaining an understanding the practical implementation of a network penetration test. Furthermore, the use of Python as a scripting language, with its readily available documentation and easy-to-learn syntax (Donaldson, 2009), would allow a user to customise the script to their own preferences and for bespoke configurations to be quickly adapted for different organisational needs which may not be available with other commercially available training tools.

3.2 FUTURE WORK

The potential scope for this script is considerable and a completed version with full functionality would take a significant amount of time to design, however a fully featured program could uncover further weaknesses and vulnerabilities that would offer a richer learning experience for the user. The following improvements are considered by the developer as aspects that could be implemented as a starting point:

- Incorporate input validation to enable error checking with the use of try-except blocks.
- Provide options to increase/decrease the amount of feedback provided to the learner in the Learner Report.
- Develop the script to take an IP range, not just a single IP address.
- Include functionality for the user to initiate a new test immediately after one has finished, without the need to exit and restart the program.
- Expand the scope of capabilities for the detection and enumeration of services (e.g. LDAP, RDP and Kerberos).
- Provide fully automated integration with Nessus with the use of the Selenium package.
- Develop a custom tool for brute forcing to enable the suppression of real-time Hydra output to the display.
- Extend the password hacking stage to attempt to brute force users in other services than SMB.

The developer's intention was for the script to also offer an automated web application penetration test. This is in and of itself is a considerable undertaking and presents the possibility for custom designed tools for the interrogation of a web application. A fully functional automated web application

penetration test could offer the learner a significant number of opportunities for their learning development. The completion of this work would require both a very high level of competence with the Python language and a thorough understanding of the functionality of web applications, as well as the ability to apply these concepts to an effective training model.

4 REFERENCES

Awati, R., 2023. *Nessus.* [Online]
Available at: https://www.techtarget.com/searchnetworking/definition/Nessus
[Accessed 22 April 2024].

Coutinho, S. et al., 2023. *Cyber security skills in the UK labour market 2023,* s.l.: Department for Science, Innovation & Technology.

Das, M., 2023. *Python-docx: A Comprehensive Guide to Creating and Manipulating Word Documents in Python.* [Online]
Available at: https://medium.com/@HeCanThink/python-docx-a-comprehensive-guide-to-creating-and-manipulating-word-documents-in-python-a765cf4b4cb9
[Accessed 25 April 2024].

Donaldson, T., 2009. Python. In: Berkeley: Peachpit Press.

Erickson, J., 2008. Hacking: The Art of Exploitation. In: 2 ed. San Francisco: No Starch Press.

Halock, 2020. *What are the different options for pen testing?.* [Online]
Available at: https://www.halock.com/faq/different-options-pen-testing
[Accessed 10 December 2023].

Hart-Davis, G. & Hart-Davis, T., 2022. *Python.* Hoboken, New Jersey: John Wiley & Sons Inc..

Heinz, M., 2021. *Secure Password Handling in Python.* [Online]
Available at: https://martinheinz.dev/blog/59
[Accessed 21 April 2024].

Kali, n.d. *Kali Linux Overview.* [Online]
Available at: https://www.kali.org/features/
[Accessed 3 December 2023].

Li, C., 2015. Penetration Testing Curriculum Development in Practice. *Journal of Information Technology Education: Innovations in Practice,* Volume 14, pp. 85-99.

Merriam-Webster, n.d. *run through.* [Online]
Available at: https://www.merriam-webster.com/thesaurus/run%20through
[Accessed 22 April 2024].

mrb3n, 2023. *How to become a penetration tester in 2023: (Practical) career guide.* [Online]
Available at: https://www.hackthebox.com/blog/how-to-become-a-pentester
[Accessed 25 April 2024].

Novotny, J., 2024. *How to Write and Run a Python Script.* [Online]
Available at: https://www.linode.com/docs/guides/how-to-write-and-run-python-script/
[Accessed 22 April 2024].

OWASP Foundation, 2020. *OWASP Web Security Testing Guide.* [Online]
Available at: https://owasp.org/www-project-web-security-testing-guide/
[Accessed 27 January 2024].

Polaris Market Research, 2022. *Penetration Testing Market Size Global Report, 2022 - 2030,* s.l.: s.n.

PTES, 2014. *High Level Organization of the Standard.* [Online]
Available at: http://www.pentest-standard.org/index.php/Main_Page
[Accessed 5 December 2023].

Roberts, S., 2024. *What Is Kali Linux? A Beginner's Guide.* [Online]
Available at: https://www.theknowledgeacademy.com/blog/what-is-kali-linux/
[Accessed 25 Aoril 2024].

Schmeelk, S. & Dragos, D., 2023. *Penetration Testing and Ethical Hacking: Risk Assessments and Student Learning.* College Station, IEEE, pp. 1-6.

SecureTriad, n.d. *Web Application Penetration Testing: Steps, Methods, and Tools.* [Online]
Available at: https://securetriad.io/web-applications-penetration-testing/
[Accessed 10 December 2023].

Sharma, R., 2024. *How to Implement Switch Case Functions in Python? [2024].* [Online]
Available at: https://www.upgrad.com/blog/how-to-implement-switch-case-functions-in-python/
[Accessed 18 April 2024].

Stuttard, D. & Pinto, M., 2011. *The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws.* 2 ed. Indianapolis: John Wiley & Sons, Inc..

Weidman, G., 2014. *Penetration Testing A Hands-On Introduction to Hacking.* 13 ed. San Francisco: No Starch Press, Inc..

Williams, H. et al., 2021. *Understanding the Cyber Security Recruitment Pool,* s.l.: Department for Digital Culture, Media & Sport.

APPENDICES

APPENDIX A TABLE OF TOOLS AND LIBRARIES USED IN THE FINALISED SCRIPT

Libraries

All Python libraries used have readily available documentation.

Library Name	Version	Purpose
subprocess	3.11.18	Built-in Python library. Provides ability to spawn new processes, connect to their input/output/error pipes and read return codes as strings.
os	3.11.18	Built-in Python library. Supplies methods for implementing native OS functionality.
python-docx	0.8.11	Library for reading, creating, and updating Microsoft Word documents.
nmap	0.7.1	Provides an interface for Python to run Nmap scans and manipulate the results.
ftplib	3.11.18	Built-in Python library. Implements the client side of an FTP connection to a remote computer, allowing the automation of file transfer operations.
datetime	3.11.18	Built-in Python library. Supplies classes to work with date and time objects.
getpass	3.11.18	Built-in Python library. Provides a platform for accepting sensitive information (e.g. passwords) at the command-line in a secure (masked) manner.
webbrowser	3.11.18	Built-in Python library. Web browser controller. Provides for the display of documents to users within a graphical web browser session.
colorama	0.4.5	Adds colours and styling to text in command-line outputs.

Tools

Tool Name	Version	Purpose
nmap	7.92	Native Kali tool. Comes pre-installed with hundreds of scripts for enumerating various services on a target machine. No installation or configuration required. Easy-to-use and documentation is readily available.
grep	3.7	Native Linux tool. Very commonly used tool for pattern matching in strings. No installation or configuration required. Easy-to-use and documentation is readily available.
cut	8.32	Native Linux tool. String transformation command for removing sections from each line of text. No installation or

		configuration required. Easy-to-use and documentation is readily available.
tr	8.32	Native Linux tool. String transformation command for translating or deleting characters from text. No installation or configuration required. Easy-to-use and documentation is readily available.
sed	4.8	Native Linux tool. Stream editor that performs basic transformation commands on output. Documentation is readily available.
SecLists	N/A	Large repository of word lists available from a central location. Word lists have been categorised according to potential use. Many word lists are available in multiple sizes to enable more efficient brute forcing.
xsltproc	20914.10135.820	Performs simple file conversion (XML to HTML). Easy to use.
smbclient	4.16.3-Debian	Native Kali tool. Simple command line tool used for obtaining information from an SMB service instance. Allows for easy listing of file shares. No installation or configuration required. Easy-to-use and documentation is readily available.
enum4linux	0.9.1	Native Kali tool. Simple command line tool used for obtaining information from an SMB service instance. Provides comprehensive information in a centralised output. No installation or configuration required. Easy-to-use and documentation is readily available.
cat	8.32	Native Linux tool. Very commonly used tool for reading file contents and strings. No installation or configuration required. Easy-to-use and documentation is readily available.
hydra	9.3	Native Kali tool. Can be used against multiple services with a standardised command line. Quick and highly customisable. No installation or configuration required. Easy-to-use and documentation is readily available.

Management Report.txt (example output)

Network Infrastructure Penetration Test Report

Target IP address: 192.168.10.1

Port Scanning Results

The following open ports/services were discovered on 192.168.10.1:

Port number	Service name
21	ftp
22	ssh
25	smtp
53	domain
79	finger

services.txt (example output)

```
21,open,ftp
22,open,ssh,OpenSSH,for_Windows_8.6,(protocol,2.0)
25,open,smtp,ArGoSoft,Freeware,smtpd,1.8.2.9
53,open,domain,Simple,DNS,Plus
79,open,finger,ArGoSoft,Mail,fingerd
```

Learner Report.python-docx (example output)

Network Infrastructure Penetration Test Report

Learner report for JKP's network penetration test on 192.168.10.1.

Learner report for JKP's network penetration test on 192.168.10.1.

Port Scanning Results

The command used to identify open ports was:

```
nmap -p- --min-rate=1000 192.168.10.1 | grep ^[0-9] | cut -d '/' -f 1 | tr '
' ',' | sed s/,$//
```

An explanation of the command and its switches can be found below.

Command/switch	Purpose
nmap	Tool for port scanning
-p-	Specifies that all ports (1-65535) are to be scanned.
--min-rate=1000	Specifies that Nmap should send packets at or above 1000 per second. Stipulates that this is to be a fast scan.

The ports Nmap found to scan were 21,22,25,53,79,80,88,90,110,135,139,389,445,464,593,636,3268,3269,3389,5985,9389,47001,49664,49665,49666,49668,49671,49674,49675,49676,49679,49683,49696,56596

The Nmap command used to conduct the scan was:

```
nmap -sV 192.168.10.1 -p
21,22,25,53,79,80,88,90,110,135,139,389,445,464,593,636,3268,3269,3389,5985,9389,47001,49664,49665,49666,49668,49671,49674,49675,49676,49679,49683,49696,56596 -oA
nmapscan > nmapscan.txt
```

Command/switch	Purpose
nmap	Tool for port scanning
-sV	Attempts to ascertain version numbers of services running on open ports.

The full report of open ports/services including the service version information can be found in the nmapscan.html file. You should research these versions for known vulnerabilities - the Exploit-DB website is a good place to start.

The following open ports/services were discovered on 192.168.10.1:

Port number	Service name
21	ftp
22	ssh
25	smtp
53	domain
79	finger

nessusstring.txt (example output)

```
"Medium","Finger Recursive Request Arbitrary Site Redirection"
"Medium","Finger Recursive Request Arbitrary Site Redirection"
"Low","DHCP Server Detection"
"Medium","HTTP TRACE / TRACK Methods Allowed"
"Medium","HTTP TRACE / TRACK Methods Allowed"
```

nessusresults.txt (example output)

```
Medium,Finger Recursive Request Arbitrary Site Redirection
Medium,Finger Recursive Request Arbitrary Site Redirection
Low,DHCP Server Detection
Medium,HTTP TRACE / TRACK Methods Allowed
Medium,HTTP TRACE / TRACK Methods Allowed
```

Management/Learner report (example output)

Vulnerability Scan results

Details of critical, high, medium and low vulnerabilities found on 192.168.10.1.

Risk	Vulnerability Name
Medium	Finger Recursive Request Arbitrary Site Redirection
Medium	Finger Recursive Request Arbitrary Site Redirection
Low	DHCP Server Detection
Medium	HTTP TRACE / TRACK Methods Allowed
Medium	HTTP TRACE / TRACK Methods Allowed

smbshares.txt (example output)

```
ADMIN$,Mapping: DENIED Listing: N
C$,Mapping: DENIED Listing: N
Fileshare1,Mapping: OK Listing: OK Writing: N
Fileshare2,Mapping: OK Listing: OK Writing: N
HR,Mapping: OK Listing: OK Writing: N
```

smbshares_string.txt (example output)

```
ADMIN$  Mapping: DENIED Listing: N
C$      Mapping: DENIED Listing: N
Fileshare1      Mapping: OK Listing: OK Writing: N
Fileshare2      Mapping: OK Listing: OK Writing: N
HR      Mapping: OK Listing: OK Writing: N
```

smbusers.txt (example output)

```
Administrator
krbtgt
test
K.Thompson
V.Nelson
```

bruteforced.txt

```
# Hydra v9.3 run at 2024-04-21 12:14:37 on 192.168.10.1 smb (hydra -L
smbusers.txt -P ../supersmalldictionary.txt -o bruteforced.txt
smb://192.168.10.1)
[445][smb] host: 192.168.10.1   login: W.Holt   password: interception
[445][smb] host: 192.168.10.1   login: A.Kennedy   password: interception
```

Management Report.python-docx

SMB Password Hacking Results

The following SMB passwords were cracked on 192.168.10.1:

SMB Username	SMB Password
W.Holt	interception
A.Kennedy	interception

Learner Report.python-docx (example output)

SMB Password Hacking Results

SMB passwords have been brute forced with Hydra.

The commands used to crack passwords were:

hydra -L smbusers.txt -P ../supersmalldictionary.txt smb://192.168.10.1 -o bruteforced.txt

cat bruteforced.txt| grep "login" | tr -s " " | tr -d ":" | tr " " "," | cut -d "," -f 5,7 > smbcreds.txt

An explanation of the command and its switches can be found below.

Command/switch	Purpose
hydra	Tool for enumerating password brute forcing.
-L	Passes a dictionary file of usernames to hydra.

The following SMB passwords were cracked on 192.168.10.1:

SMB Username	SMB Password
W.Holt	interception
A.Kennedy	interception

APPENDIX F NETWORKPENTEST PSEUDO-CODE

```
run networkportscan function
run vulnerability scan?
      if yes:
            run vulnscan function
      if no:
            continue
check for ftp
      if present:
            set ftp presence flag
check for ssh
      if present:
            write note to learner report
check for telnet
      if present:
            write note to learner report
check for smtp
      if present:
            set smtp presence flag
            run smtpusers function
            write note to learner report
check for dns
      if present:
            write note to learner report
            continue
check for http
      if present:
            set http presence flag
check for smb
      if present:
            set smb presence flag
check for pop3
      if present:
            set pop3 presence flag
            run pop3users function
            write note to learner report
if ftp presence flag set:
      run ftpfir function
if smb presence flag set:
      if test credentials present:
            run smbscan function
            write note to learner report
      if test credentials not present:
            run smbscan function
            write note to learner report
if http presence flag set:
      write note to learner report
```

```
if ftplisting.txt exists:
      export contents to learner report
else:
      if ftp presence flag set:
            write note to learner report
if smtpusers.txt exists:
      export contents to learner report
if smbshares.txt exists:
      run smbsharereport function
if smbusers.txt exists:
      run smbuserreport function
if smbusers.txt exists:
      run hydrasmb function
open web browser window with directory of created files
quit
```

APPENDIX G SWITCH-CASE STATEMENTS FOR SERVICE ENUMERATION

```
with open('services.txt') as f:
      for line in f:
            line=line.strip()
            currentline=line.split(",")
            portnumber=currentline[0]
            match portnumber:
                  case "20":
                        ftpflag=1

                  case "21":
                        ftpflag=1

                  case "22":
                        learnerreport.add_paragraph("Possible SSH instance
discovered. If you have credentials, you may be able to gain access using an
SSH client.", style="List Bullet")

                  case "23":
                        learnerreport.add_paragraph("Possible Telnet instance
discovered. You may be able to gain access using a Telnet client.",
style="List Bullet")

                  case "25":
                        print("SMTP FOUND!! Enumerating with Nmap smtp-enum-
users script. Please wait...")
                        smtpbrute=1
                        smtpusers()
                        learnerreport.add_paragraph("Possible SMTP instance
discovered. Enumerated with Nmap smtp-enum-users script. Users enumerated and
shown below.", style="List Bullet")

                  case "53":
                        learnerreport.add_paragraph("Possible DNS instance
discovered. You may be able to enumerate this with tools such as dnsenum.",
style="List Bullet")

                  case "80":
                        httpflag=1

                  case "110":
                        print("POP3 FOUND!! Enumerating with Nmap pop3-brute
script. Please wait...")
                        pop3brute=1
                        pop3users()
                        learnerreport.add_paragraph("Possible POP3 instance
discovered. Enumerated with the pop3-brute Nmap script.", style="List
Bullet")
```

```
case "135":
      smbflag=1

case "139":
      smbflag=1

case "443":
      httpflag=1

case "445":
      smbflag=1

case "8080":
      httpflag=1

case "8443":
      httpflag=1
```

networkportscan progress messages

```
Obtaining list of open ports. Please wait ...
Running Nmap scan. Please wait ...
Nmap scan complete.
Exporting port scanning results to Management Report.
Exporting port scanning results to Learner Report.
Converting Nmap scan results to easily readable HTML format.
```

vulnscan prompt

```
    Port scanning has completed. The next stage of a penetration test is to scan the target for vulnerabilities.
    Unfortunately, automated vulnerability scanning is not possible with the available software.
    You may run a scan manually. If you choose to do so, instructions will be provided.

NOTE: NESSUS SCANS CAN TAKE A CONSIDERABLE AMOUNT OF TIME TO COMPLETE

Do you wish to conduct a manual vulnerability scan? Y/N y
```

Service enumeration progress messages

```
Checking for vulnerable services - FTP, SSH, SMB, DNS, HTTP/HTTPS, Telnet, SMTP and POP3.
Exporting enumerated service details to Leaner Report.
SMTP FOUND!! Enumerating with Nmap smtp-enum-users script. Please wait ...
POP3 FOUND!! Enumerating with Nmap pop3-brute script. Please wait ...
FTP FOUND!! Attempting to enumerate with FTP client. Please wait ...
SMB FOUND!! Attempting to perform authenticated enumeration with enum4linux. Please wait ...
Exporting FTP enumeration results to Learner Report.
Exporting SMTP user results to Learner Report.
Exporting SMB share results to Learner Report.
Exporting SMB user results to Learner Report.
SMB USERS FOUND!! Attempting to crack passwords with Hydra. Please wait ...
```

Learner Report.python-docx

Enumerated service details

- Possible SSH instance discovered. If you have credentials, you may be able to gain access using an SSH client.
- Possible SMTP instance discovered. Enumerated with Nmap smtp-enum-users script. Users enumerated and shown below.
- Possible DNS instance discovered. You may be able to enumerate this with tools such as dnsenum.
- Possible POP3 instance discovered. Enumerated with the pop3-brute Nmap script.
- Possible SMB instance discovered. Enumerated with enum4linux
- Possible HTTP(S) instance(s) discovered.

Remember to check the Nmap results for services to enumerate that are less commonly used, or that might be running on non-standard ports.

Function completion

```
Exporting SMB password hacking results to Management Report.
Exporting SMB password hacking results to Leaner Report.
Tests are complete. Files have been saved to /home/kali/runmethru/JKP 192.168.10.1 2024-04-22 11:11:35.150922. Program will exit.
Opening folder containing files in web browser...
```

Browser window