COMP 8505 Final Project

Xinghua Wei & Tao Yuan

A00978597 & A00952007

Introduction

In this assignment, I will implement a Linux backdoor. This backdoor will allow a user to remote control, execute a command, and receive the command result, monitoring and retrieving file, and keylogger.

Constraints

There are few constraints I need to follow:

Part 1

This portion of the project will deal strictly with the design and implementation of the main backdoor itself.

• This application will accept packets regardless of whether or not any firewall rules are in place once its service port has been opened by a separate application. This means that you will be implementing this part using **libpcap**.

• The application itself will run as a disguised process; you are required to make it as obscure as possible so as to avoid detection.

• The application will only accept those packets that have been authenticated. The authentication will be in the form of an encrypted password in the packet. This can be embedded in the payload or within one of the protocol header fields.

• Once the packet has been authenticated it will extract a command (also encrypted) from the payload portion and execute the command.

• The results of the command will then be sent back to the **attacker** machine (application) using a **covert channel**.

• In addition to sending back requested files, this component will also install a **keylogger** in the compromised system and send the key strokes file to the CNC server.

• One way to configure the application parameters is by means of a configuration file.

Part 2

• This portion of the project will implement the file exfiltration and port knocking component for the covert channel to access the **attacker** component and deliver the exfiltrated data.

• The application will watch for the creating of a specific file in a specific directory and when that occurs, it will automatically send the file to the **attacker machine** on the other side.

• The file will be delivered covertly using a special sequence of packets or "knocks", which the **attacker** machine will authenticate and provide access to the requested port and application.

• Once the exfiltration is complete your application must close access to the ports again.

• Access to the ports may be time-based or controlled by a separate sequence of packets. In other words, the user can remotely specify how and when to close access to the backdoor application.

Attacker Component

• The **attacker application** must have all of the features to connect to and control the remote system via the backdoor running on a compromised **victim machine**.

• Aside from simple executing remote commands the overall application must provide an exfiltration function. The user will be able to specify that the remote system search (or watch a directory or file for events) for a particular file and send the contents of that file (including the file containing keystrokes) back to the client application covertly.

• The attacker application will accept and decode the knock sequence, provide access to the port and service that will be accepting all the encrypted data that will contain either the results of the command execution, or the exfiltrated file contents from the server.

• I suggest that all the attacker application parameters be selected and set using a configuration file.

Design

Tools

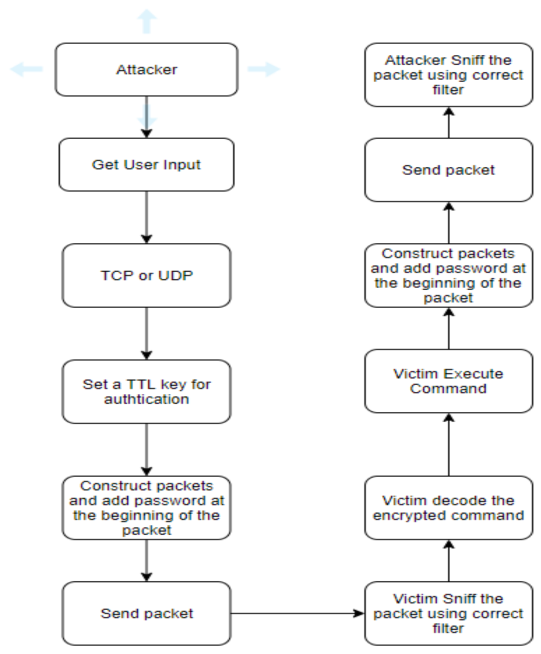
To satisfy all the constraints with Python, the following tools will be used:

- Python 3.8
- Scapy
  - Scapy has a sniff function that is great for getting packets. Sniff has an argument prn that allows users to pass a function that executes with each packet sniffed. Also, unlike traditional TCP\UDP connections, sniff does not need to specify buffer size, which gives me the flexibility to control the packets.
- Setproctitle
  - The setproctitle module allows a process to change its title as displayed by system tools such as ps. Therefore, anyone who is looking at the process table can hardly identify the backdoor process.
- Subprocess
  - Subprocess module allows users to spawn new processes, connect to their input, output and error pipes
  - Popen function takes arguments to set up the new process so the parent can communicate with it via pipes.
  - Stdout and Stderr will be pipes to open and will be where the command executes and where the results store
- Crypto
  - Crypto is a python cryptography toolkit o AES encryption will be used in this assignment
- Watchdog
  - Watchdog is a Python library that can monitor files at the time of its creation or its modification.
- Pyxhook
  - pyxhook is a library that allows you to listen for keyboard events on Linux. View license.

Detail Design – Covert Channel

A covert channel is a mechanism that allows users to send and receive data without the permission of the system. Attackers often make use of this technique to infiltrate the system of their target, either for retrieving data or modifying data. This can be achieved by embedding the data inside unintended fields of the packet, such as the IP ID field, or the IP TTL field.

In our implementation, we have chosen to code our covert channel in Python, and user could choose which protocol he will use.


Attacker
Get User Input
TCP or UDP
Set a TTL key for authtication
Construct packets and add password at the beginning of the packet
Send packet
Victim Sniff the packet using correct filter
Victim decode the encrypted command
Victim Execute Command
Construct packets and add password at the beginning of the packet
Send packet
Attacker Sniff the packet using correct filter

Client (Attacker)

On the client-side, first users need to gather a few pieces of information:

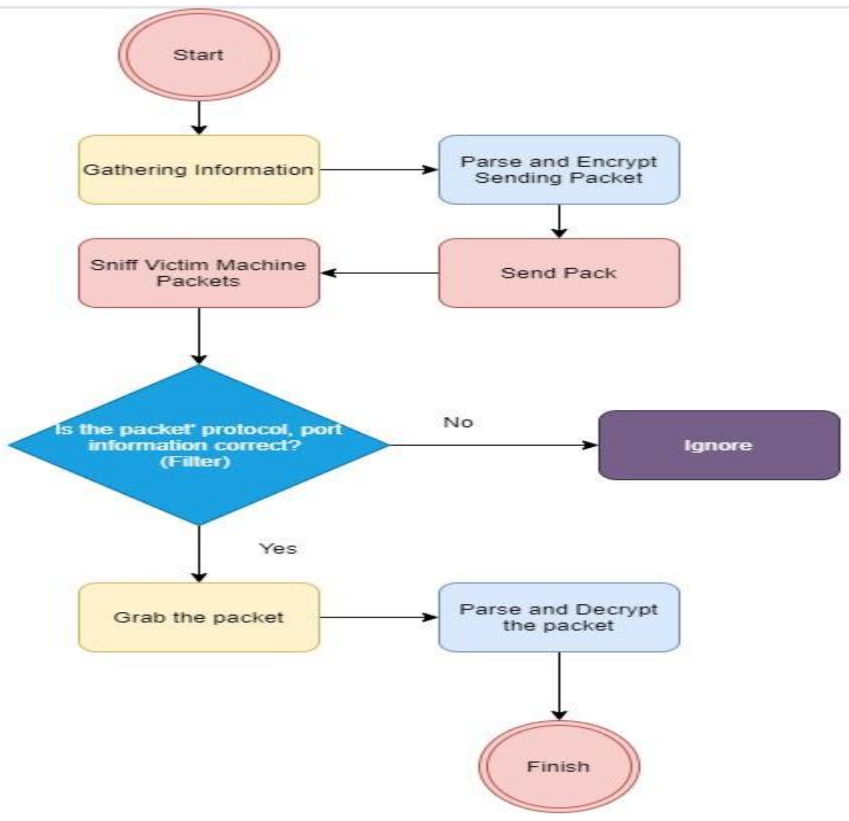
- Target IP address(Destination IP address)
- Target port (Destination port and source port) o I will use destination port 8000 and source port 8505 as default o UDP will be used as the default
- Attacker IP address(Source IP address)
- The backdoor title which camouflages the backdoor

- Protocol: UDP or TCP
- The command to send to execute on the victim machine

Once users have all the information, the program will start processing this information and send it to a victim machine.

1. Attacker machine start construct packets. TTL field in the IP header will be assigned a key that the victim needs to authenticate the key to identify which packets to sniff.
2. Concatenate the command and process title to one packet
3. Encrypt the packet with AES and UTF-8
4. Send the packet so the victim machine can sniff the packet
5. Sniff the packet in the victim machine using a filter, and it will use TTL authenticate key to check. Once the key matches, then the information we want would be sniffed.

Load the sniffed raw packets and decrypt them into readable information


Start
Gathering Information
Parse and Encrypt Sending Packet
Send Pack
Sniff Victim Machine Packets
Is the packet' protocol, port information correct? (Filter)
No
Ignore
Yes
Grab the packet
Parse and Decrypt the packet
Finish

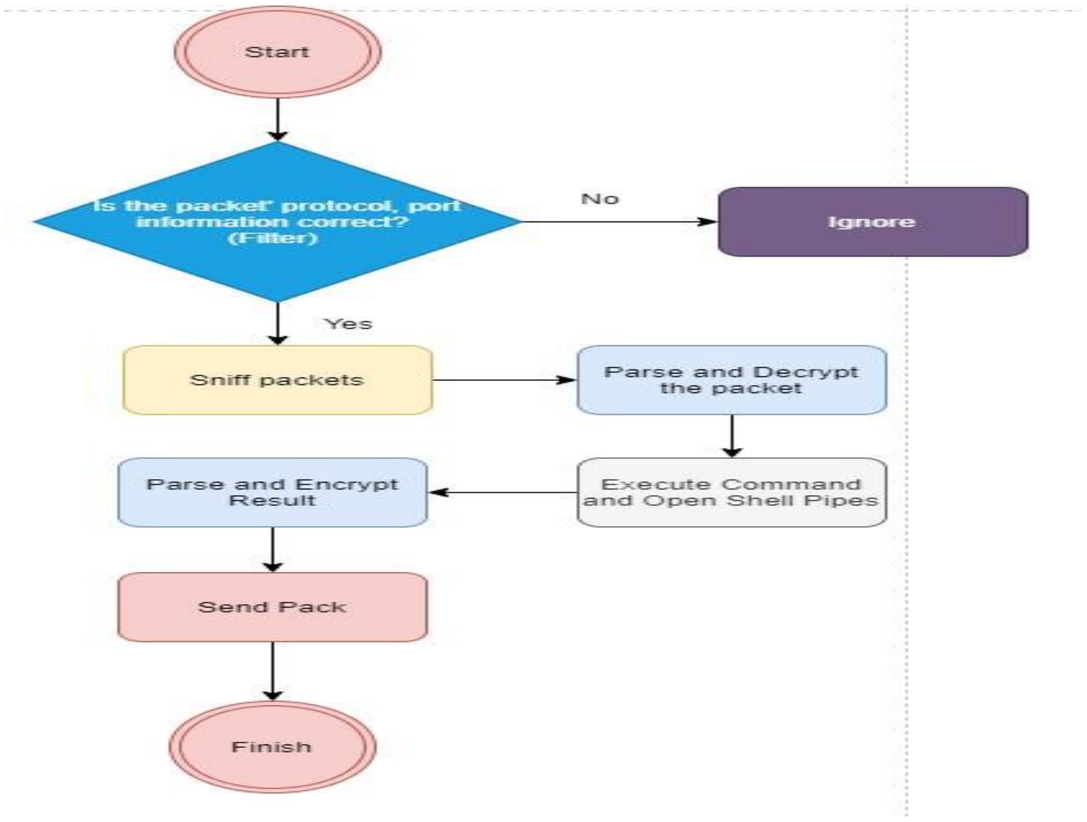
Server(Victim)

On the server-side, the default protocol, destination port and the source port will be set in this assignment.

- Protocol: UDP o
- Destination port: 8000
- Source Port: 8505

Once the program starts, the following actions will be done:

1. Check the TTL authenticate key
2. Load the packet with sniff
3. Parse and decrypt the packet to retrieve the command and the title
4. Use subprocess and Popen functions to create pips to execute the command
5. Camouflage the process with the title
6. Read results from stdout and stderr
7. Encrypt the result with AES and TTL key
8. Send packet so the attacker machine can sniff the packet


Start
Is the packet' protocol, port information correct? (Filter)
No
Ignore
Yes
Sniff packets
Parse and Decrypt the packet
Execute Command and Open Shell Pipes
Parse and Encrypt Result
Send Pack
Finish

Design - Cipher

For cipher, I will use AES with CFB 8 mode (8-bit cipher feedback mode). The following information is required:

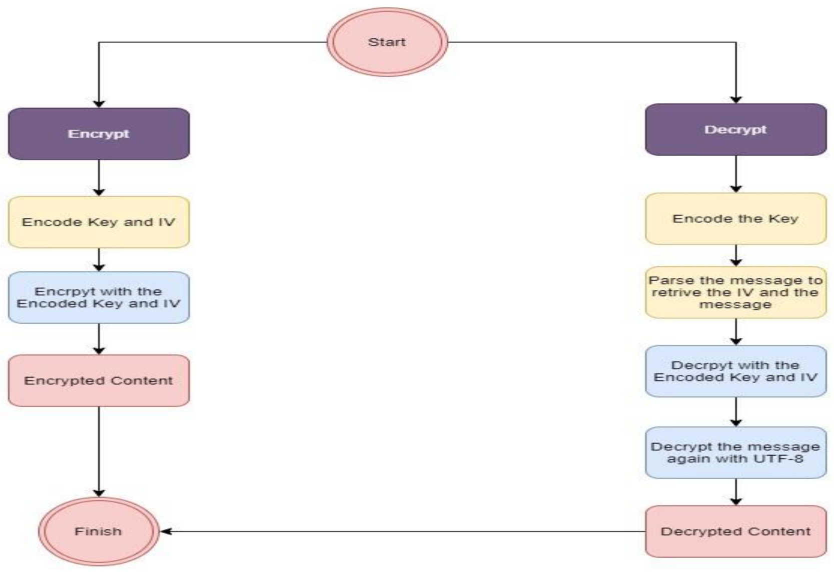
- Key: fixed data block size of 16 bytes
- IV: Initialization Vector is used by several modes to randomize the encryption and produce distinct ciphertexts even if the same plaintext is encrypted multiple times. For CFB mode, it must be 16 bytes long.

Encryption:

1. Encode the key and the initialization vector with UTF-8 because AES in Crypto cannot take a string.
2. Encrypt the message and return the IV and the message

Decryption:

1. Encode the key
2. Parse the message that the former 16 bytes will be the IV and the rest will be the message
3. Decrypt the message with the key and the IV
4. Decode the message again with UTF-8 because it is still in bytes format



Start
Encrypt
Encode Key and IV
Encrpyt with the Encoded Key and IV
Encrypted Content
Finish
Decrypt
Encode the Key
Parse the message to retrive the IV and the message
Decrpyt with the Encoded Key and IV
Decrypt the message again with UTF-8
Decrypted Content

Detail Design – File Exfiltrated

Client (Attacker)

1. The attacker machine starts to construct packets. The TTL field in the IP header will be assigned a key that the victim needs to authenticate the key to identify which packets to sniff.

2. Concatenate the fie name and process title to one packet

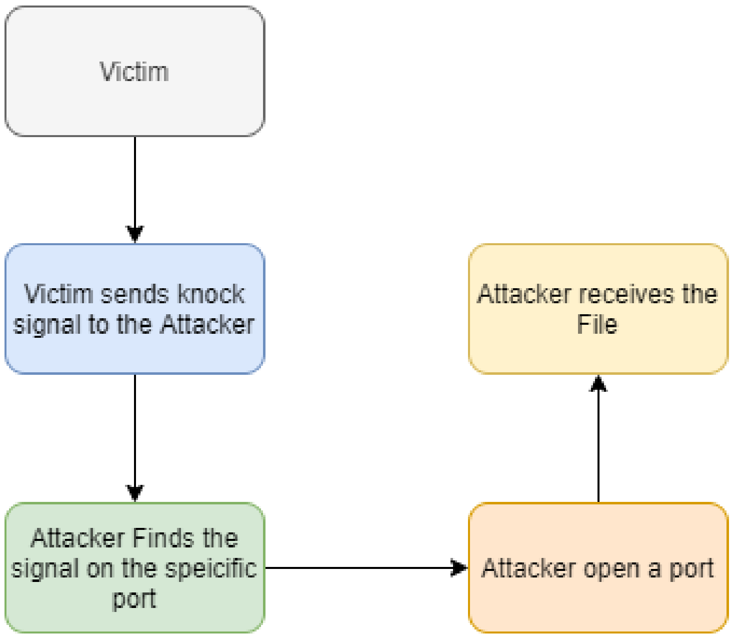
3. Encrypt the packet with AES and UTF-8

4. Send the packet so the victim machine can sniff the packet

5. Sniff the packet in the victim machine using a filter, and it will use TTL authenticate key to check. Once the key matches, then the information we want would be sniffed.

6. Once the victim receives the right port, the attacker will open the specified firewall port for a while (10s). Then the packets will get through into the attacker machine.

Server(Victim)


Victim
Victim sends knock signal to the Attacker
Attacker Finds the signal on the speicific port
Attacker open a port
Attacker receives the File

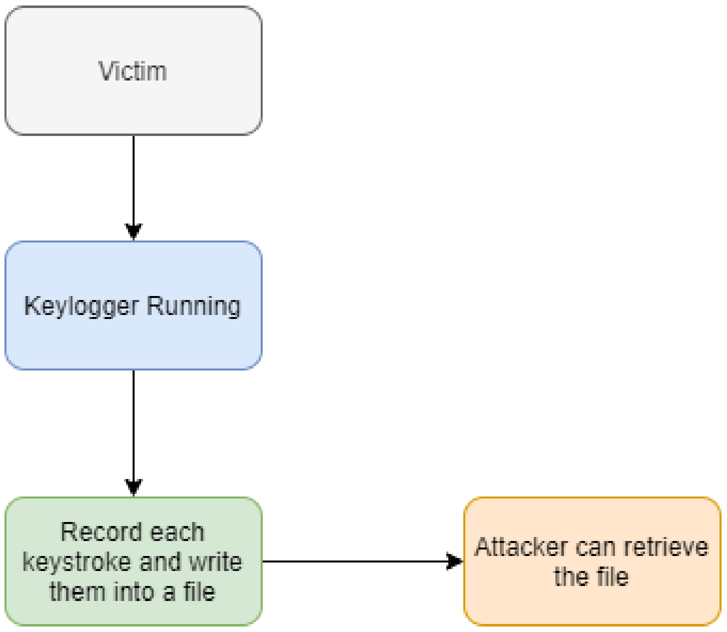
Once the program starts, the following actions will be done:

1. Check the TTL authenticate key
2. Load the packet with sniff
3. Parse and decrypt the packet to retrieve the command and the title
4. Use subprocess and Popen functions to create pips to execute the command
5. Camouflage the process with the title
6. Read results from stdout and stderr
7. Encrypt the result with AES and TTL key
8. Send a signal to a specific Attacker firewall port.
9. Send packet so the attacker machine through the open port.

Detail Design –Keylogger

For the keylogger, Pyxhook will be used for recording the keystrokes. Once the victim machine starts running the program, the keylogging will be running in the background and capture all keystrokes and record them into a file.

Attackers can use a terminal command to check the keylogger, or the attacker can retrieve the keylogger log file.


Victim
Keylogger Running
Record each keystroke and write them into a file
Attacker can retrieve the file

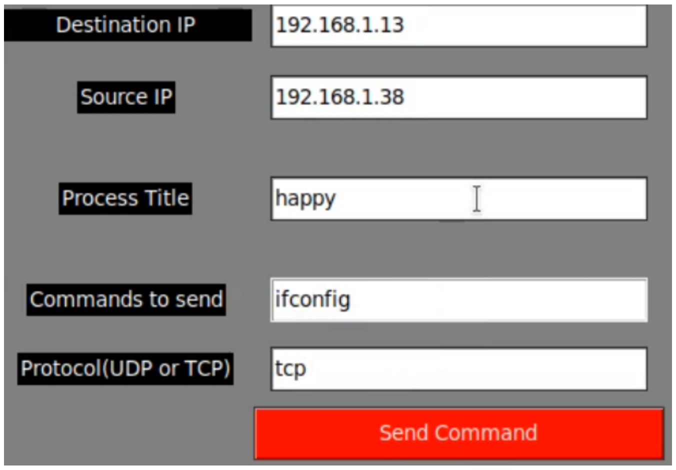
Test

Test Case 1 – Send command by using protocol TCP (Camouflage the process)

Test case one will demonstrate camouflaging the process title to the title users specified.

- Destination IP: 192.168.1.13
- Source IP: 192.168.1.38
- Camouflaged title: happy
- Command: ifconfig
- Protocol: TCP

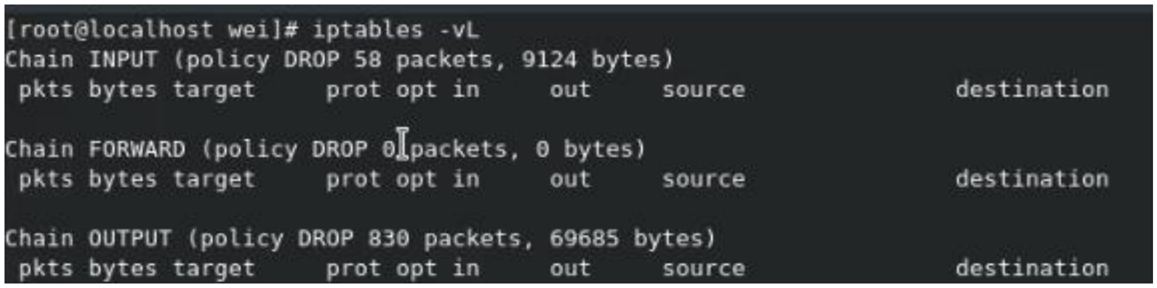
Attacker



Destination IP	192.168.1.13
Source IP	192.168.1.38
Process Title	happy
Commands to send	ifconfig
Protocol(UDP or TCP)	tcp

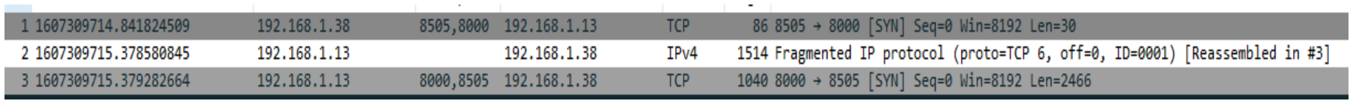
Users specify destination IP, source IP, process title and commands to send.

The firewall is closed.

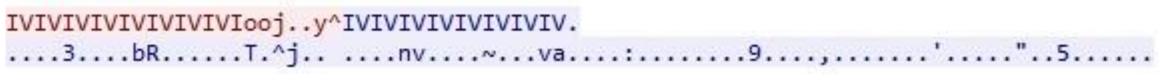


```
[root@localhost wei]# iptables -vL
Chain INPUT (policy DROP 58 packets, 9124 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy DROP 830 packets, 69685 bytes)
 pkts bytes target     prot opt in     out     source               destination
```

The result is back to the attack machine with lists of files in the backdoor file directory.

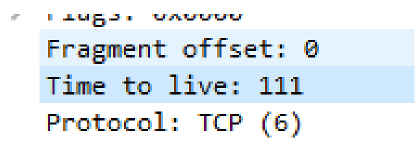


```
1 1607309714.841824509   192.168.1.38   8505,8000  192.168.1.13   TCP     86 8505 → 8000 [SYN] Seq=0 Win=8192 Len=30
2 1607309715.378580845   192.168.1.13              192.168.1.38   IPv4  1514 Fragmented IP protocol (proto=TCP 6, off=0, ID=0001) [Reassembled in #3]
3 1607309715.379282664   192.168.1.13   8000,8505  192.168.1.38   TCP   1040 8000 → 8505 [SYN] Seq=0 Win=8192 Len=2466
```



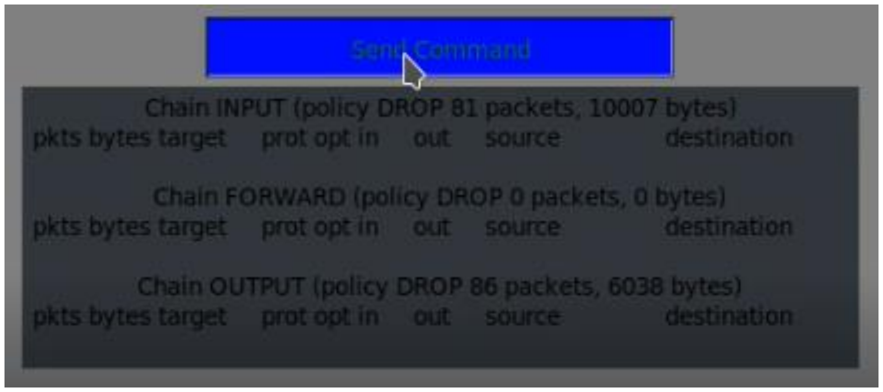
```
IVIVIVIVIVIVIVIVIooj..y^IVIVIVIVIVIVIVIV.
....3....bR......T.^j.. ....nv....~...va....:........9...,.......'....."..5......
```

The Wireshark capture shows that the attacker sends the command to the victim and get the result. The result and the command are encrypted. Also, the TTL key is set as we expect to authenticate packets.



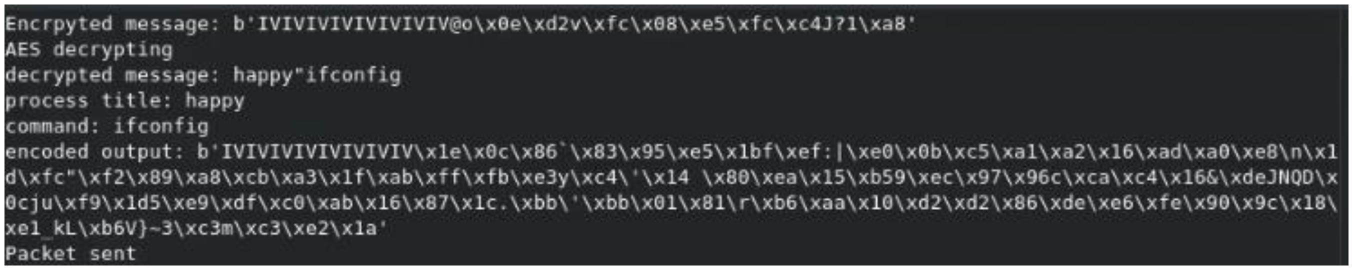
```
Fragment offset: 0
Time to live: 111
Protocol: TCP (6)
```

The result will be shown on the application.



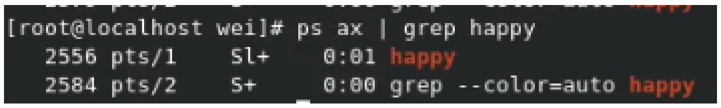
```
Chain INPUT (policy DROP 81 packets, 10007 bytes)
pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy DROP 0 packets, 0 bytes)
pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy DROP 86 packets, 6038 bytes)
pkts bytes target     prot opt in     out     source               destination
```

Victim



```
Encrpyted message: b'IVIVIVIVIVIVIVIV@o\x0e\xd2v\xfc\x08\xe5\xfc\xc4J?1\xa8'
AES decrypting
decrypted message: happy"ifconfig
process title: happy
command: ifconfig
encoded output: b'IVIVIVIVIVIVIVIV\x1e\x0c\x86`\x83\x95\xe5\x1bf\xef:|\xe0\x0b\xc5\xa1\xa2\x16\xad\xa0\xe8\n\x1
d\xfc"\xf2\x89\xa8\xcb\xa3\x1f\xab\xff\xfb\xe3y\xc4\'\x14 \x80\xea\x15\xb59\xec\x97\x96c\xca\xc4\x16&\xdeJNQD\x
0cju\xf9\x1d5\xe9\xdf\xc0\xab\x16\x87\x1c.\xbb\'\xbb\x01\x81\r\xb6\xaa\x10\xd2\xd2\x86\xde\xe6\xfe\x90\x9c\x18\
xe1_kL\xb6V}~3\xc3m\xc3\xe2\x1a'
Packet sent
```

On the victim machine, I printed out the information just to make sure everything is working. So, it shows the command and the title received, and the encrypted message that will be sent back. Because the encoded cipher could be very long, I only print the first 120 bytes.



```
[root@localhost wei]# ps ax | grep happy
  2556 pts/1    Sl+    0:01 happy
  2584 pts/2    S+     0:00 grep --color=auto happy
```

This image shows that the program successfully camouflaged the process title to the received title "admin."

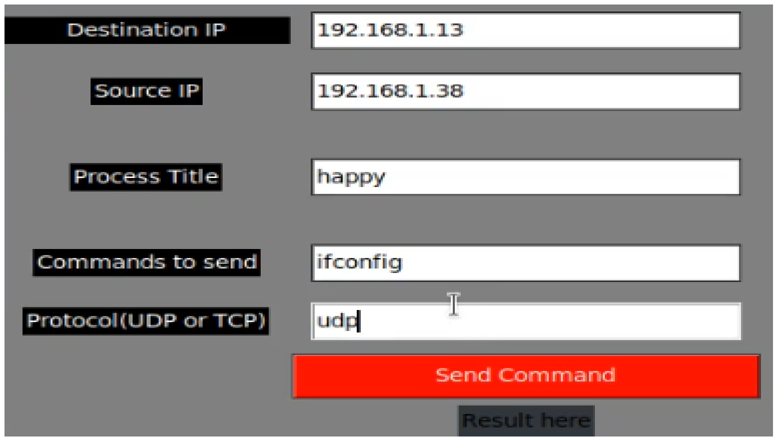
This test means the test is a success.

Test Case 2 – Send command by using protocol UDP

Test case one will demonstrate camouflaging the process title to the title users specified.

- Destination IP: 192.168.1.13
- Source IP: 192.168.1.38
- Camouflaged title: happy
- Command: ifconfig
- Protocol: UDP

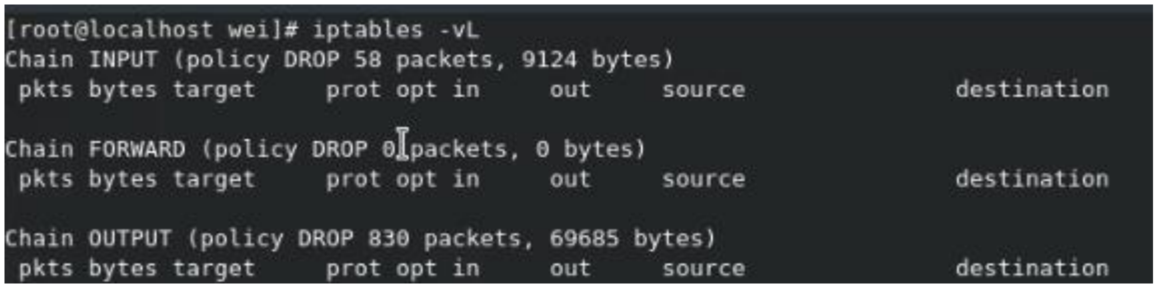
Attacker



Destination IP	192.168.1.13
Source IP	192.168.1.38
Process Title	happy
Commands to send	ifconfig
Protocol(UDP or TCP)	udp

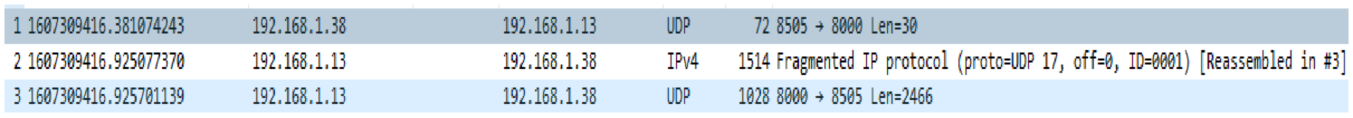
Users specify destination IP, source IP, process title and commands to send.

The firewall is closed.

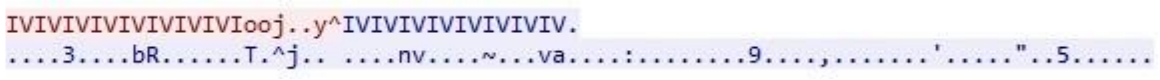


```
[root@localhost wei]# iptables -vL
Chain INPUT (policy DROP 58 packets, 9124 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy DROP 830 packets, 69685 bytes)
 pkts bytes target     prot opt in     out     source               destination
```

The result is back to the attack machine with lists of files in the backdoor file directory.

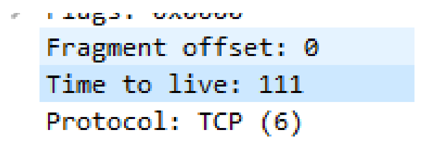


```
1 1607309416.381074243   192.168.1.38   192.168.1.13   UDP     72 8505 → 8000 Len=30
2 1607309416.925077370   192.168.1.13   192.168.1.38   IPv4  1514 Fragmented IP protocol (proto=UDP 17, off=0, ID=0001) [Reassembled in #3]
3 1607309416.925701139   192.168.1.13   192.168.1.38   UDP   1028 8000 → 8505 Len=2466
```



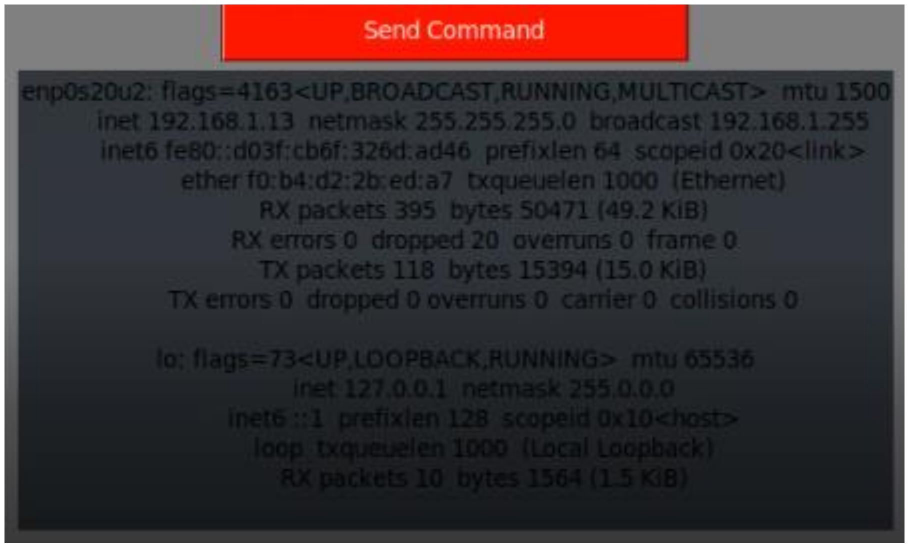
```
IVIVIVIVIVIVIVIVIooj..y^IVIVIVIVIVIVIVIV.
....3....bR......T.^j.. ....nv....~...va....:........9....,.......'....."..5......
```

The Wireshark capture shows that the attacker sends the command to the victim and get the result. The result and the command are encrypted. Also, the TTL key is set as we expect to authenticate packets.



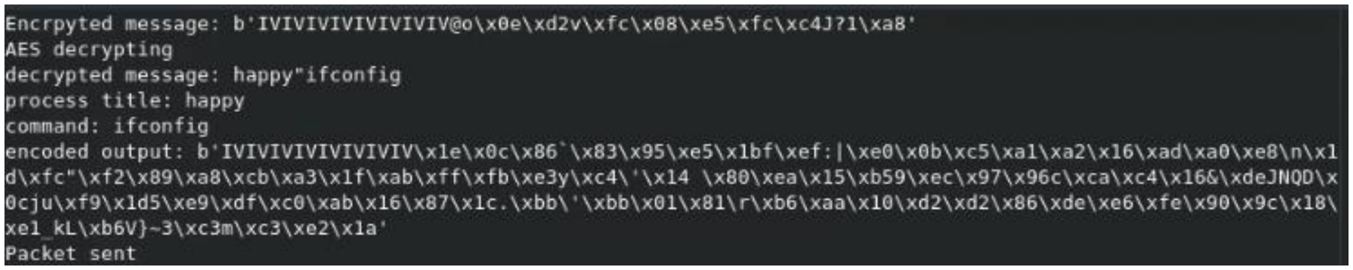
```
Fragment offset: 0
Time to live: 111
Protocol: TCP (6)
```

The result will be shown on the application.



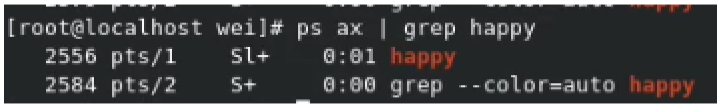
```
enp0s20u2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.13  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::d03f:cb6f:326d:ad46  prefixlen 64  scopeid 0x20<link>
        ether f0:b4:d2:2b:ed:a7  txqueuelen 1000  (Ethernet)
        RX packets 395  bytes 50471 (49.2 KiB)
        RX errors 0  dropped 20  overruns 0  frame 0
        TX packets 118  bytes 15394 (15.0 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

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 10  bytes 1564 (1.5 KiB)
```

Victim



```
Encrpyted message: b'IVIVIVIVIVIVIVIV@o\x0e\xd2v\xfc\x08\xe5\xfc\xc4J?1\xa8'
AES decrypting
decrypted message: happy"ifconfig
process title: happy
command: ifconfig
encoded output: b'IVIVIVIVIVIVIVIV\x1e\x0c\x86`\x83\x95\xe5\x1bf\xef:|\xe0\x0b\xc5\xa1\xa2\x16\xad\xa0\xe8\n\x1
d\xfc"\xf2\x89\xa8\xcb\xa3\x1f\xab\xff\xfb\xe3y\xc4\'\x14 \x80\xea\x15\xb59\xec\x97\x96c\xca\xc4\x16&\xdeJNQD\x
0cju\xf9\x1d5\xe9\xdf\xc0\xab\x16\x87\x1c.\xbb\'\xbb\x01\x81\r\xb6\xaa\x10\xd2\xd2\x86\xde\xe6\xfe\x90\x9c\x18\
xe1_kL\xb6V}~3\xc3m\xc3\xe2\x1a'
Packet sent
```

On the victim machine, I printed out the information just to make sure everything is working. So, it shows the command and the title received, and the encrypted message that will be sent back. Because the encoded cipher could be very long, I only print the first 120 bytes.



```
[root@localhost wei]# ps ax | grep happy
  2556 pts/1    Sl+    0:01 happy
  2584 pts/2    S+     0:00 grep --color=auto happy
```

This image shows that the program successfully camouflaged the process title to the received title "admin."

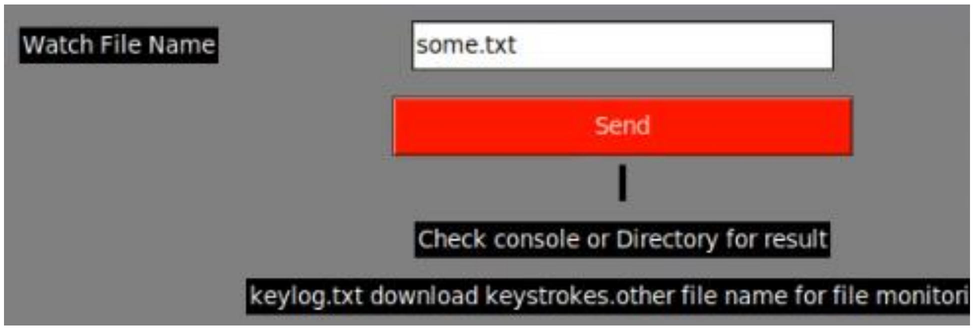
This test means the test is a success.

Test Case 3 – TCP File Monitoring and Port Knock

Test case one will demonstrate camouflaging the process title to the title users specified.

- Destination IP: 192.168.1.13
- Source IP: 192.168.1.38
- Camouflaged title: happy
- File: Some.txt
- Protocol: TCP

Attacker



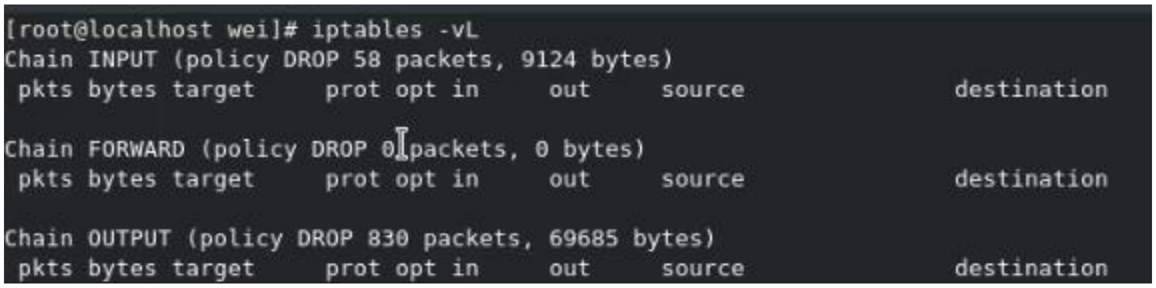
Watch File Name	some.txt

Check console or Directory for result

keylog.txt download keystrokes.other file name for file monitori

Users specify destination IP, source IP, process title and file to watch.

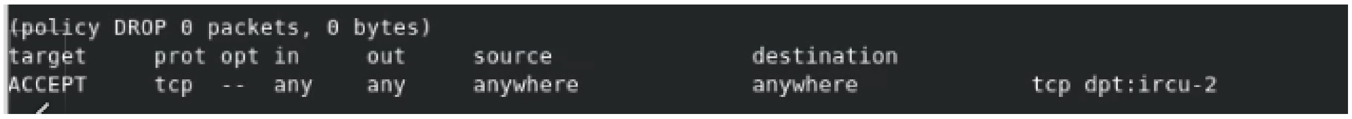
The firewall is closed.



```
[root@localhost wei]# iptables -vL
Chain INPUT (policy DROP 58 packets, 9124 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy DROP 830 packets, 69685 bytes)
 pkts bytes target     prot opt in     out     source               destination
```

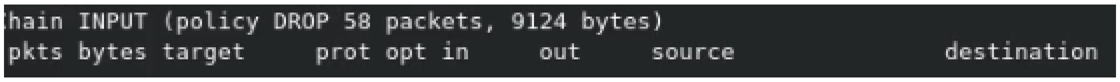
The result is back to the attack machine with lists of files in the backdoor file directory.

Once the victim sends the firewall signal to port 6666, the attacker will open a firewall port on port 6666.



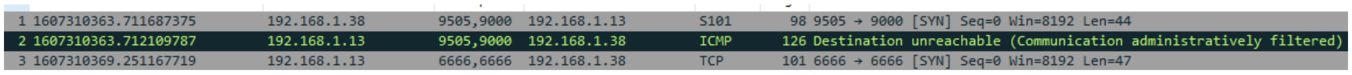
```
(policy DROP 0 packets, 0 bytes)
target     prot opt in     out     source               destination
ACCEPT     tcp  --  any    any     anywhere             anywhere             tcp dpt:ircu-2
```

And the port will be close 10 seconds after the transmission.

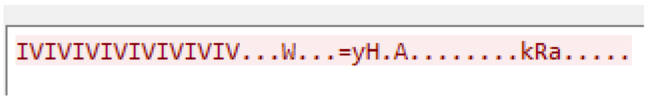


```
hain INPUT (policy DROP 58 packets, 9124 bytes)
pkts bytes target     prot opt in     out     source               destination
```

Once the port opens, the victim will send the file through port 6666.



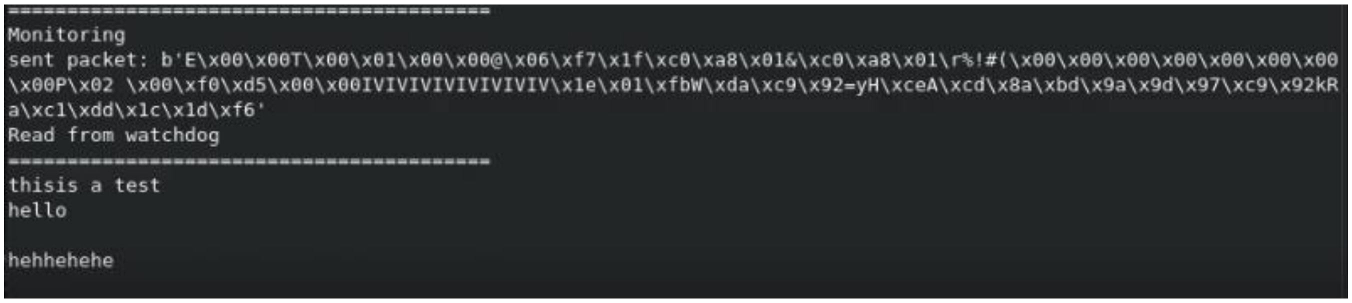
```
1 1607310363.711687375    192.168.1.38    9505,9000  192.168.1.13    S101    98 9505 → 9000 [SYN] Seq=0 Win=8192 Len=44
2 1607310363.712109787    192.168.1.13    9505,9000  192.168.1.38    ICMP   126 Destination unreachable (Communication administratively filtered)
3 1607310369.251167719    192.168.1.13    6666,6666  192.168.1.38    TCP    101 6666 → 6666 [SYN] Seq=0 Win=8192 Len=47
```



```
IVIVIVIVIVIVIVIV...W...=yH.A........kRa.....
```

The Wireshark capture shows that the attacker sends the command to the victim and get the result. The result and the command are encrypted.

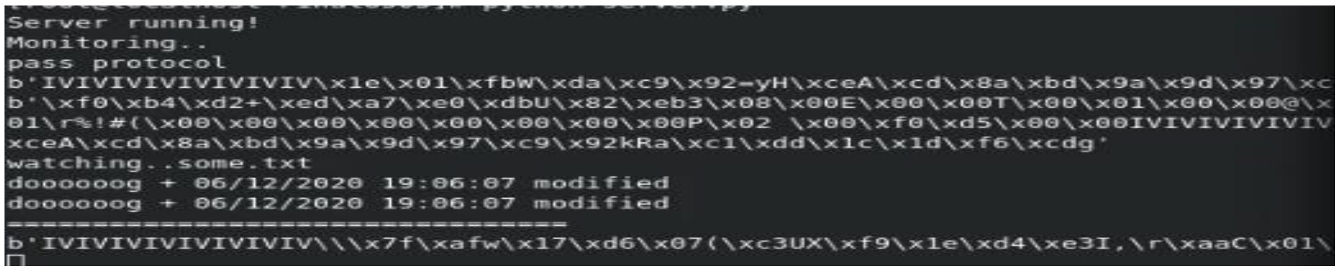
The result will be shown on the console.



```
========================================
Monitoring
sent packet: b'E\x00\x00T\x00\x01\x00\x00@\x06\xf7\x1f\xc0\xa8\x01&\xc0\xa8\x01\r%!#(\x00\x00\x00\x00\x00\x00\x00
\x00P\x02 \x00\xf0\xd5\x00\x00IVIVIVIVIVIVIVIV\x1e\x01\xfbW\xda\xc9\x92=yH\xceA\xcd\x8a\xbd\x9a\x9d\x97\xc9\x92kR
a\xc1\xdd\x1c\x1d\xf6'
Read from watchdog
========================================
thisis a test
hello

hehhehehe
```

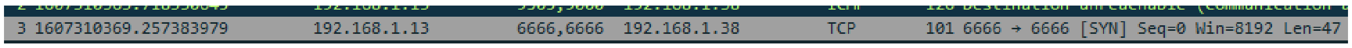
Victim

On the victim machine, I printed out the information just to make sure everything is working. So, it shows the command and the title received, and the encrypted message that will be sent back. Because the encoded cipher could be very long, I only print the first 120 bytes.



```
Server running!
Monitoring..
pass protocol
b'IVIVIVIVIVIVIVIV\x1e\x01\xfbW\xda\xc9\x92=yH\xceA\xcd\x8a\xbd\x9a\x9d\x97\xc
b'\xf0\xb4\xd2+\xed\xa7\xe0\xdbU\x82\xeb3\x08\x00E\x00\x00T\x00\x01\x00\x00@\x
01\r%!#(\x00\x00\x00\x00\x00\x00\x00\x00P\x02 \x00\xf0\xd5\x00\x00IVIVIVIVIVIV
xceA\xcd\x8a\xbd\x9a\x9d\x97\xc9\x92kRa\xc1\xdd\x1c\x1d\xf6\xcdg'
watching..some.txt
dooooooog + 06/12/2020 19:06:07 modified
dooooooog + 06/12/2020 19:06:07 modified
==============================
b'IVIVIVIVIVIVIVIV\\\x7f\xafw\x17\xd6\x07(\xc3UX\xf9\x1e\xd4\xe3I,\r\xaaC\x01\
```

The console output shows that the some.txt has been modified at what time. And then, the victim machine will send the file to the attacker machine using port knocking.



```
3 1607310369.257383979        192.168.1.13        6666,6666  192.168.1.38        TCP        101 6666 → 6666 [SYN] Seq=0 Win=8192 Len=47
```

The Wireshark capture shows that the victim is sending the file through port 6666.

This test means the test is a success.

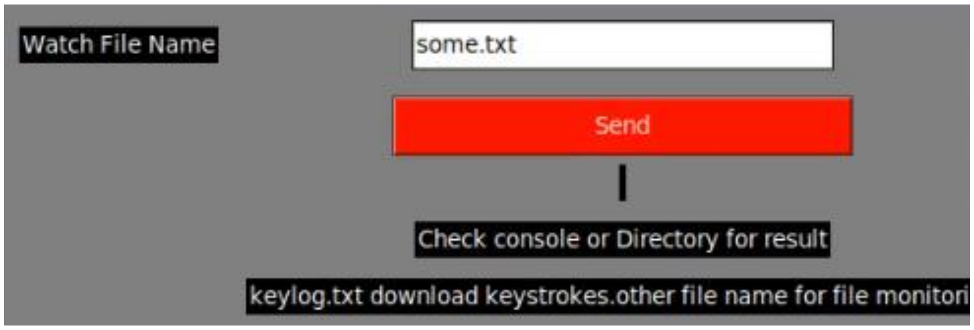
Test Case 4 – UDP File Monitoring and Port Knock

Test case one will demonstrate camouflaging the process title to the title users specified.

- Destination IP: 192.168.1.13
- Source IP: 192.168.1.38
- Camouflaged title: happy
- File: Some.txt

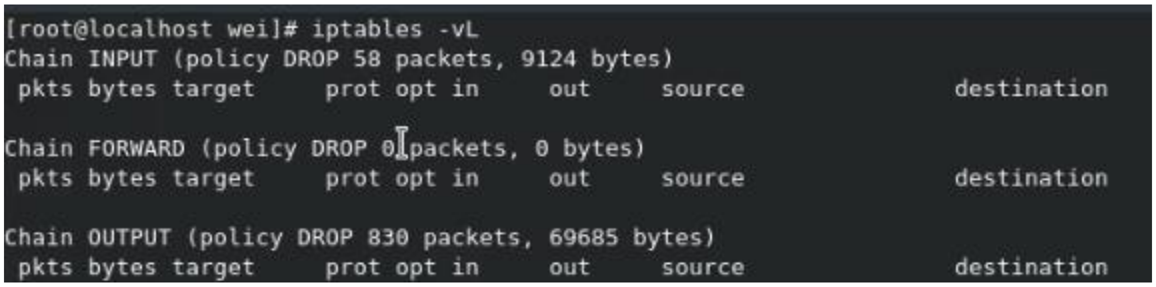
- Protocol: UDP

Attacker


Watch File Name
some.txt
Send
Check console or Directory for result
keylog.txt download keystrokes.other file name for file monitori

Users specify destination IP, source IP, process title and file to watch.

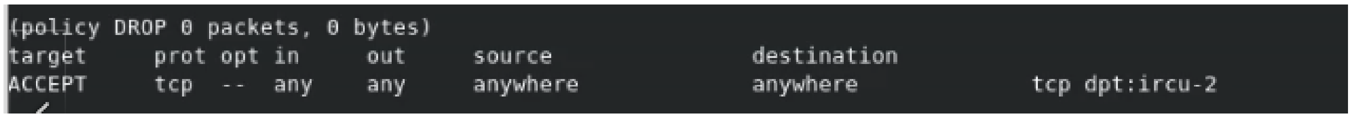
The firewall is closed.



```
[root@localhost wei]# iptables -vL
Chain INPUT (policy DROP 58 packets, 9124 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy DROP 830 packets, 69685 bytes)
 pkts bytes target     prot opt in     out     source               destination
```

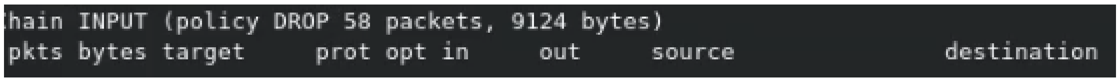
The result is back to the attack machine with lists of files in the backdoor file directory.

Once the victim sends the firewall signal to port 6666, the attacker will open a firewall port on port 6666.



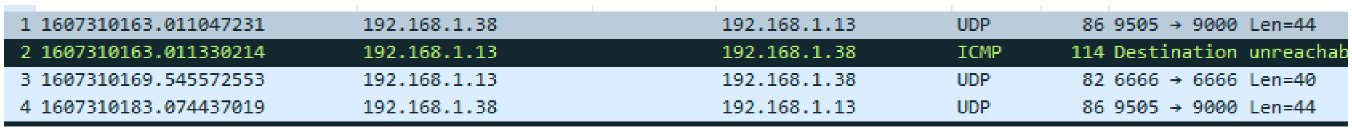
```
(policy DROP 0 packets, 0 bytes)
target     prot opt in     out     source               destination
ACCEPT     tcp  --  any    any     anywhere             anywhere             tcp dpt:ircu-2
```

And the port will be close 10 seconds after the transmission.

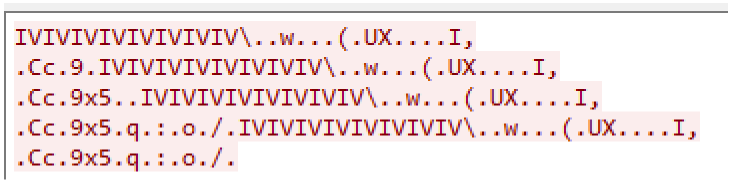


```
hain INPUT (policy DROP 58 packets, 9124 bytes)
pkts bytes target     prot opt in     out     source               destination
```

Once the port opens, the victim will send the file through port 6666.



```
1 1607310163.011047231    192.168.1.38    192.168.1.13    UDP     86 9505 → 9000 Len=44
2 1607310163.011330214    192.168.1.13    192.168.1.38    ICMP   114 Destination unreachab
3 1607310169.545572553    192.168.1.13    192.168.1.38    UDP     82 6666 → 6666 Len=40
4 1607310183.074437019    192.168.1.38    192.168.1.13    UDP     86 9505 → 9000 Len=44
```

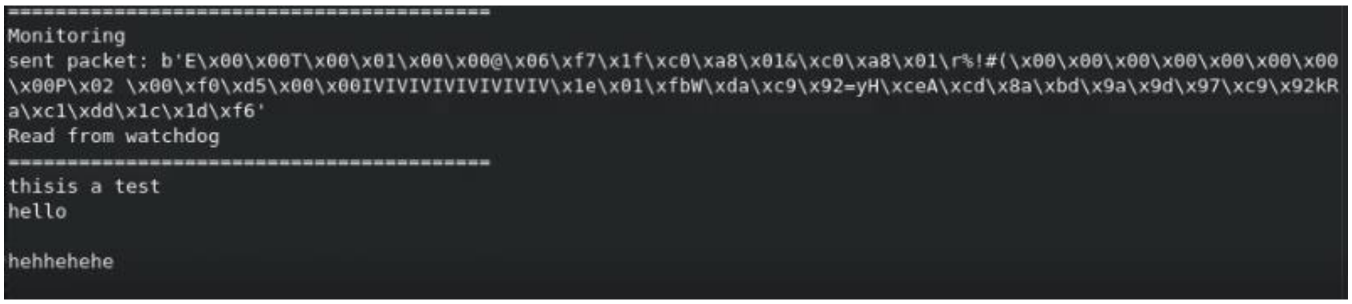


```
IVIVIVIVIVIVIVIV\..w...(.UX....I,
.Cc.9.IVIVIVIVIVIVIVIV\..w...(.UX....I,
.Cc.9x5..IVIVIVIVIVIVIVIV\..w...(.UX....I,
.Cc.9x5.q.:.o./.IVIVIVIVIVIVIVIV\..w...(.UX....I,
.Cc.9x5.q.:.o./.
```

The Wireshark capture shows that the attacker sends the command to the victim and get the result.

The result and the command are encrypted.

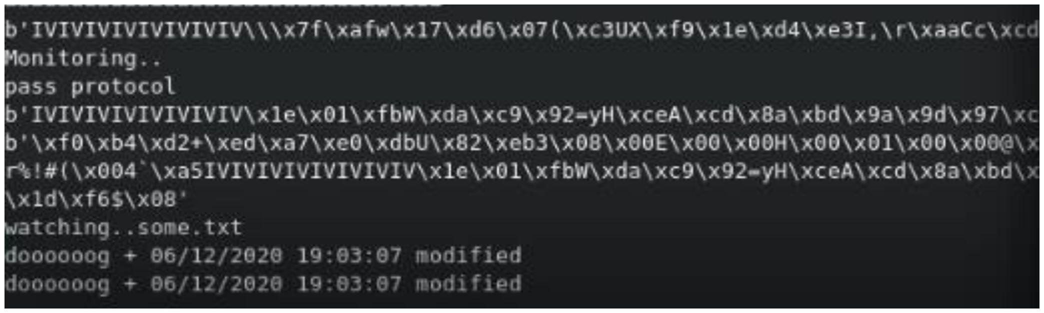
The result will be shown on the console.



```
=========================================
Monitoring
sent packet: b'E\x00\x00T\x00\x01\x00\x00@\x06\xf7\x1f\xc0\xa8\x01&\xc0\xa8\x01\r%!#(\x00\x00\x00\x00\x00\x00\x00
\x00P\x02 \x00\xf0\xd5\x00\x00IVIVIVIVIVIVIVIV\x1e\x01\xfbW\xda\xc9\x92=yH\xceA\xcd\x8a\xbd\x9a\x9d\x97\xc9\x92kR
a\xc1\xdd\x1c\x1d\xf6'
Read from watchdog
=========================================
thisis a test
hello

hehhehehe
```

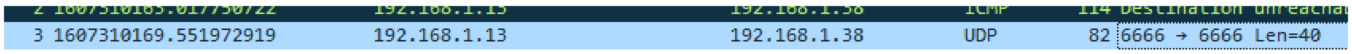
Victim

On the victim machine, I printed out the information just to make sure everything is working. So, it shows the command and the title received, and the encrypted message that will be sent back. Because the encoded cipher could be very long, I only print the first 120 bytes.



```
b'IVIVIVIVIVIVIVIV\\\x7f\xafw\x17\xd6\x07(\xc3UX\xf9\x1e\xd4\xe3I,\r\xaaCc\xcd
Monitoring..
pass protocol
b'IVIVIVIVIVIVIVIV\x1e\x01\xfbW\xda\xc9\x92=yH\xceA\xcd\x8a\xbd\x9a\x9d\x97\xc
b'\xf0\xb4\xd2+\xed\xa7\xe0\xdbU\x82\xeb3\x08\x00E\x00\x00H\x00\x01\x00\x00@\x
r%!#(\x004`\xa5IVIVIVIVIVIVIVIV\x1e\x01\xfbW\xda\xc9\x92=yH\xceA\xcd\x8a\xbd\x
\x1d\xf6$\x08'
watching..some.txt
dooooooog + 06/12/2020 19:03:07 modified
dooooooog + 06/12/2020 19:03:07 modified
```

The console output shows that the some.txt has been modified at what time. And then, the victim machine will send the file to the attacker machine using port knocking.



```
2 1607310165.017750722   192.168.1.13   192.168.1.38   ICMP   114 Destination unreachal
3 1607310169.551972919   192.168.1.13   192.168.1.38   UDP     82 6666 → 6666 Len=40
```

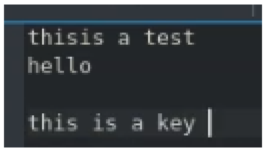
The Wireshark capture shows that the victim is sending the file through port 6666.

This test means the test is a success.

Test Case 5 – Keylogger

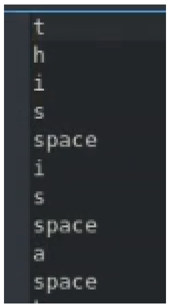
Victim

Once the victim machine starts the program, the program will starts keylogger.py in the background. Anything the user type will be recorded into a file.



```
thisis a test
hello

this is a key
```

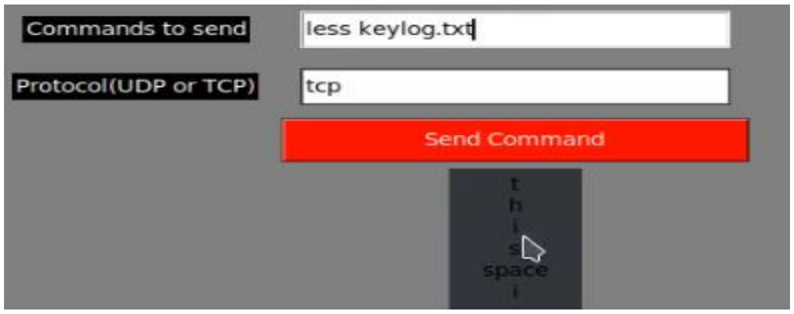
The above image simulates a user is typing something. Then every keystroke will be recorded into keylog.txt file.



```
t
h
i
s
space
i
s
space
a
space
```

Attacker

Suppose the attacker wants to see the keylogger file. There are 2 ways to accomplish that. One way is to call a command like "less keylog.txt" through the application's command field. The result will be sent back to the application.

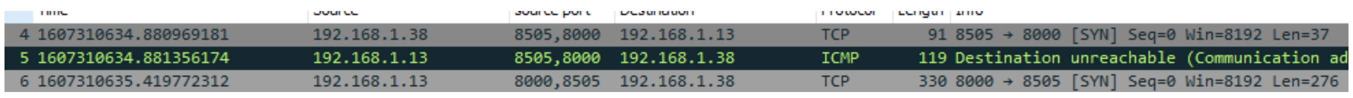


Commands to send: less keylog.txt

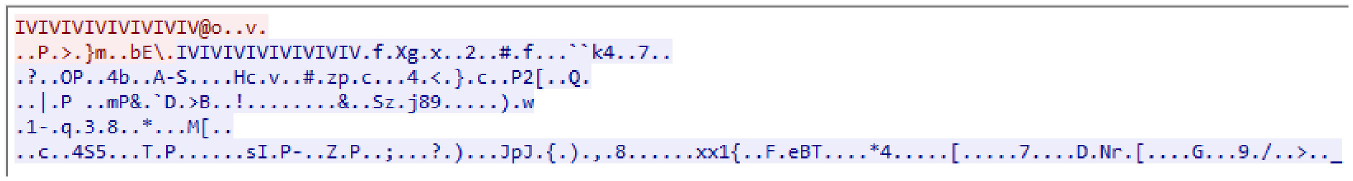
Protocol(UDP or TCP): tcp

```
t
h
i
s
space
i
```

The anther way is to retrieve the keylogger file by typing the keylogger file name in the file monitoring field. If the file monitoring field detects the file name is the same as the keylogger file name. It will send the whole file back to the attacker.



```
4 1607310634.880969181   192.168.1.38   8505,8000  192.168.1.13   TCP     91 8505 → 8000 [SYN] Seq=0 Win=8192 Len=37
5 1607310634.881356174   192.168.1.13   8505,8000  192.168.1.38   ICMP   119 Destination unreachable (Communication ad
6 1607310635.419772312   192.168.1.13   8000,8505  192.168.1.38   TCP    330 8000 → 8505 [SYN] Seq=0 Win=8192 Len=276
```



```
IVIVIVIVIVIVIVIV@o..v.
..P.>.}m..bE\.IVIVIVIVIVIVIVIV.f.Xg.x..2..#.f...``k4..7..
.?..OP..4b..A-S....Hc.v..#.zp.c...4.<.}.c..P2[..Q.
..|.P ..mP&.`D.>B..!........&..Sz.j89.....).w
.1-.q.3.8..*...M[..
..c..4S5...T.P......sI.P-..Z.P..;...?.)...JpJ.{.).,.8......xx1{..F.eBT....*4.....[.....7....D.Nr.[....G...9./..>.._
```

The Wireshark capture shows that the attacker sends the command to the victim and get the result. The result and the command are encrypted.

This test means the test is a success.

Conclusion and Prevention

In conclusion, In this project, we design and implement a complete covert application. In computer security, a covert channel is a type of attack that creates a capability to transfer information objects between processes that are not supposed to be allowed to communicate by the computer security policy. Covert channels can be hard to detect. However, there are still some ways to detect and prevent users from being attacked by such attacks:

- Users should set up firewalls machines with NAT. NAT devices allow you to use private addresses on your internal network and public addresses on your external one that all traffic of the host should first go through to the NAT device, and the NAT device will DROP any traffic except the ones from internal addresses. This will also hide ports of internal hosts that the external network can only see the NAT device.
- Users should install covert channels detecting IDS software such as Suricate and Bro IDS. With proper setup, those IDS can detect covert channels no matter the storage channel or timing channels.

- If a channel cannot be eliminated, its capacity should be reduced. What capacity is acceptable depends on the amount of information leakage that is critical. For example, if the channel is so small that classified information cannot be leaked before it is outdated, the channel capacity is tolerable.
- Host security cannot remove covert network channels, but it can prevent their exploitation in some scenarios. So, users should install firewalls and virus detection software.
- Users should block protocols that are susceptible to covert channels. For example, ICMP is blocked by many firewalls these days.
- In a closed network, protocols prone to covert channels could be blocked or replaced by versions with fewer or limited covert channels.
- The leakage of classified information from a high-security system to a low-security system is prevented by a network design where only hosts on the same security level are allowed to communicate. Such an approach may be practical for highly secure networks, but not for large open networks such as the Internet
- Many of the simple covert channels are eliminated by normalizing protocol headers, padding, and header extensions. For example, unused or reserved bits and padding are set to zero, and unknown header extensions are removed
- One way to counter the address modulation channel is limiting the number of possible addresses, effectively limiting the allowed host-to-host connections.
- Users can introduce random noise to mask the covert channel or fixing packet rates, reduces the capacity of packet and message sequence timing channels. Noise can be introduced by buffering and delaying packets, preventing senders from exactly timing outgoing packets and receivers from

accurately measuring the timing