Backdoor Packet Sniffing

Comp 8505 Assignment 3

Shane Spoor

Mat Siwoski

[**Introduction**](#_ptz2tr4ab60a) **3**

[Constraints](#_oyg10dwnx8k6) 3

[Dependencies:](#_ahk1hog4usxj) 3

[**Running the Application**](#_jqnjpl1j4tub) **4**

[**Design**](#_7rhnyu1sph3f) **6**

[Packet Sniffing](#_nj0ecnzcnt05) 6

[**Pseudocode**](#_if29cwpa5ea2) **7**

[Backdoor Server](#_5dcaeird5r4t) 7

[Backdoor Client](#_ftmdrmna79mm) 8

[**Testing**](#_dnwzoj6rgcq7) **9**

# 

# 

# 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:

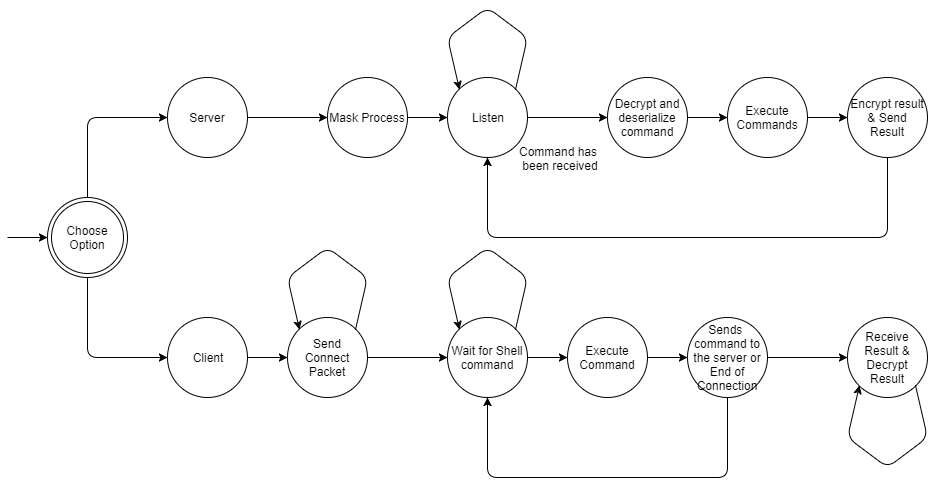
python main.py client listen port server port -s server host [-p password] [-k aes key]  
where

* 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



*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()

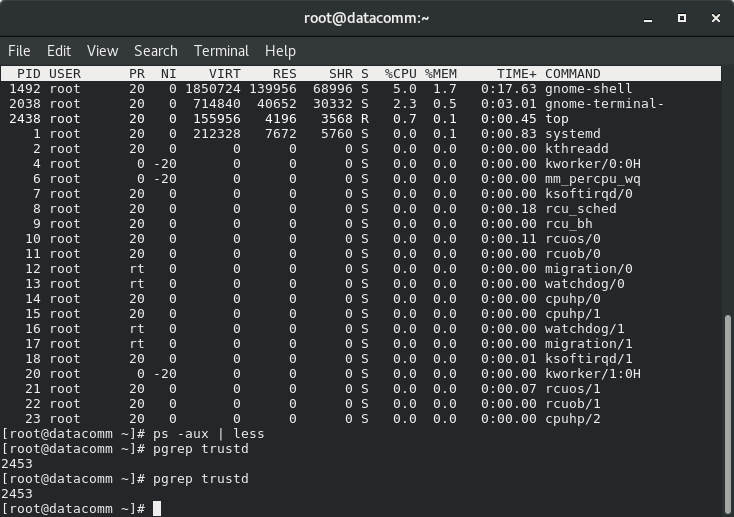
result = Command.Result.from\_bytes(bytes)  
 return result

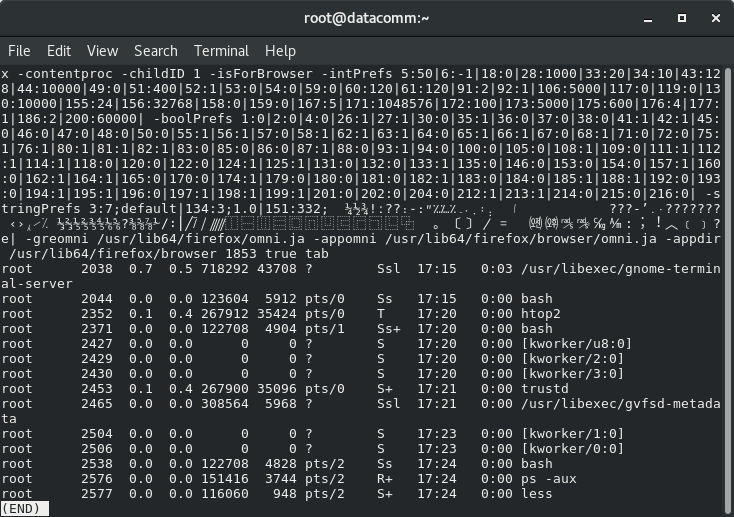
# 

# 

# 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) |
| *Fig. 2: Help screen with all available arguments* | | |
| client1.png *Fig. 3: Connected to Server* | | |
| client2.png *Fig. 4: Send a command from the client to the server* | | |
| server1.png *Fig. 5: Waiting for Client to connect* | | |
| server2.png *Fig. 6: Client Connected to the server* | | |
| Server3.png *Fig. 7: Client sends a command* | | |

*Fig. 8: Process found on the machine*

*Fig. 9: Process currently running*