COMP 8505 Assignment 2

Linux Backdoor

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.

# 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
* o 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.

Diagram

Description automatically generated

### Client (Attacker)

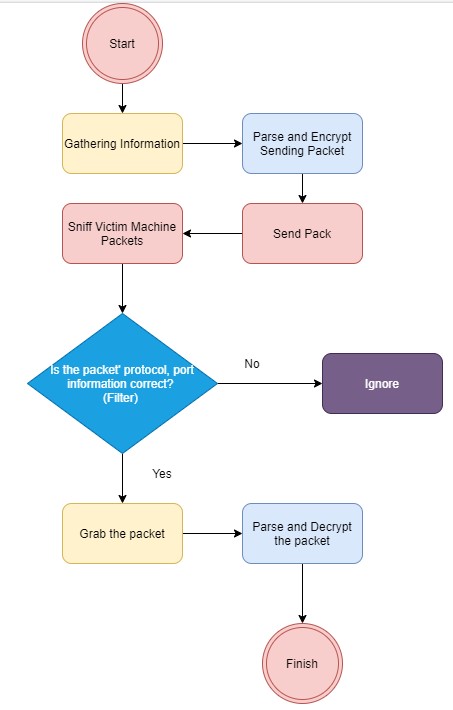
On the client-side, first users need to gather a few pieces of information:

* 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



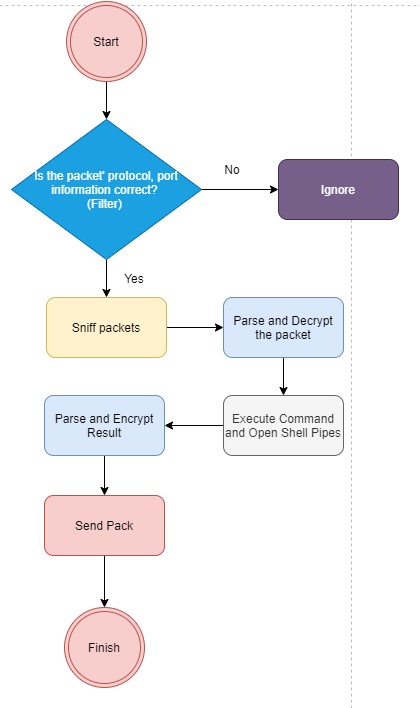
### Server(Victim)

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



### Cipher

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

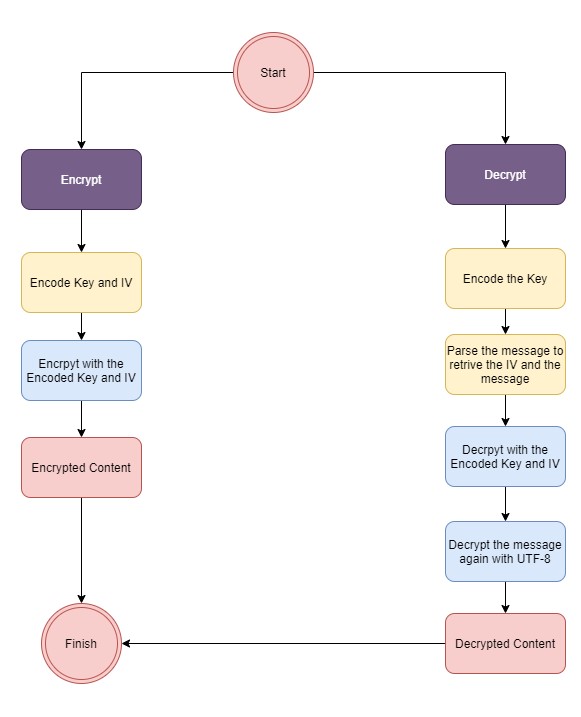
* Key: fixed data block size of 16 bytes
* IV: Initialization Vector is used by several modes to randomize the encryption and produce distinct ciphertexts even if the same plaintext is encrypted multiple times. For CFB mode, it must be 16 bytes long.

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



## Detail Design – File Exfiltrated

Diagram

Description automatically generated

### Client (Attacker)

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 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 (port knocker), the Attacker will open the specified firewall port for a while (10s). Then the packets will get through into the attacker machine.

### Server(Victim)

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 signal to a specific Attacker firewall port.
9. Send packet so the attacker machine through the open port.

## Detail Design –Keylogger

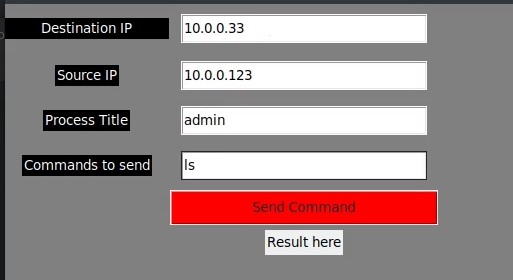
# Test

## Test Case 1 – The backdoor title (Camouflage the process)

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

* Destination IP: 10.0.0.33
* Source IP: 10.0.0.123
* Camouflaged title: admin
* Command: ls

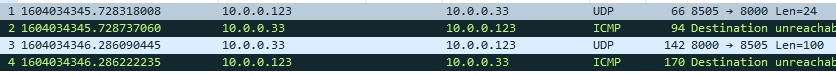
### Attacker



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



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



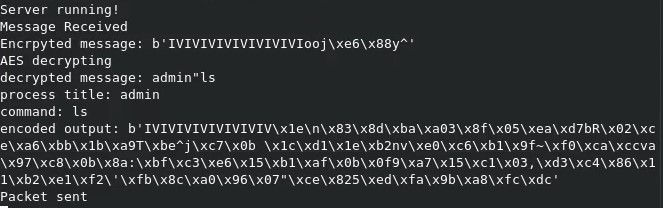


The Wireshark capture shows that the attacker sends the command to the victim and get the result. The result and the command are encrypted.

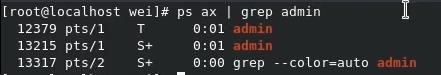


Also, the print out in the console shows the encrypted message being sent and the decrypted result.

### 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.



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

This test means the function to camouflage the process is a success.

## Test Case 2 – Encryption

Test case one will demonstrate encryption data transmission, including encrypted command and encrypted result.

* Destination IP: 10.0.0.33
* Source IP: 10.0.0.123
* Camouflaged title: root
* Command: uname -a
* Encryption

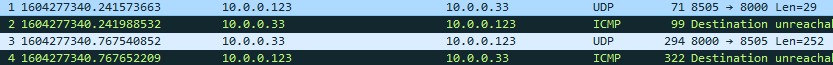
### Attacker



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



The result is back to the attack machine with the system information of the victim machine.



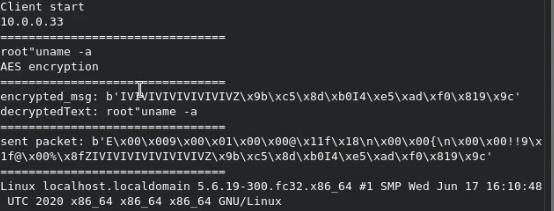


The Wireshark capture shows that the attacker sends the command to the victim and get the result. The result and the command are encrypted.

The encrypted also printed out in the console.

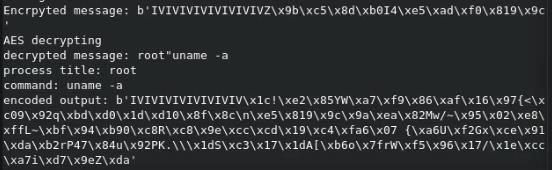


The data was encrypted using AES in bytes and IV combined with it.



Also, the print out in the console shows the encrypted message being sent and the decrypted result.

### Victim



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

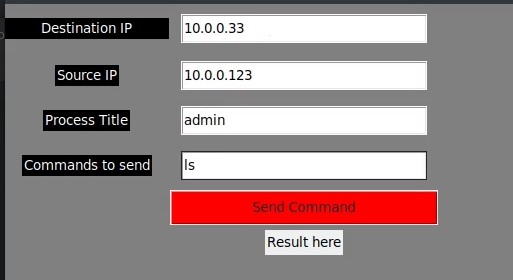
This test means the data encryption function is a success.

## Test Case 3 – List files

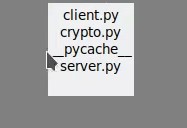
Test case one will demonstrate list files in the current directory remotely to the victim machine.

* Destination IP: 10.0.0.33
* Source IP: 10.0.0.123
* Camouflaged title: admin
* Command: ls

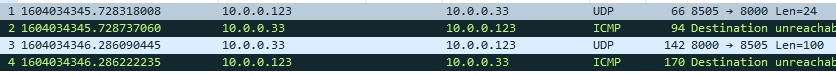
### Attacker



Users specify destination IP, source IP, process title and list files “ls” commands to send.



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



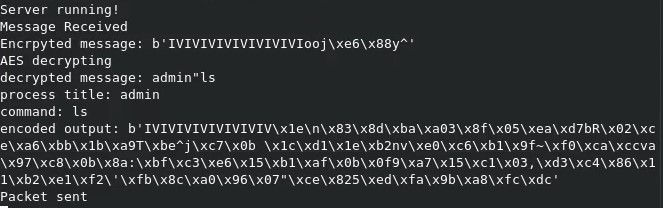


The Wireshark capture shows that the attacker sends the command to the victim and get the result. The result and the command are encrypted.



Also, the print out in the console shows the encrypted message being sent and the decrypted result.

### Victim



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.

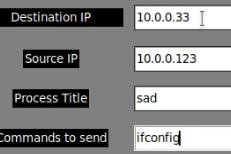
This test means the list file function is a success.

## Test Case 4 – Ifconfig

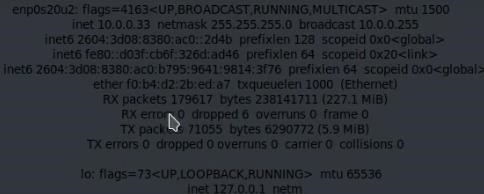
Test case one will demonstrate Ifconfig command in the current directory remotely to the victim machine.

* Destination IP: 10.0.0.33
* Source IP: 10.0.0.123
* Camouflaged title: sad
* Command: ifconfig

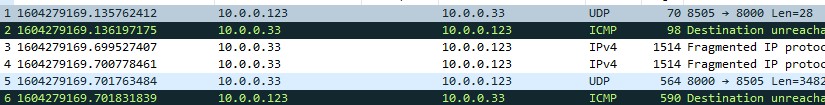
### Attacker

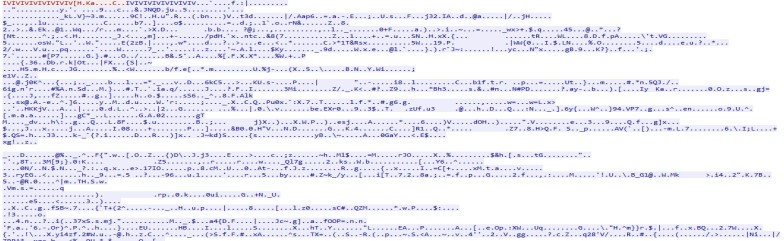


Users specify destination IP, source IP, process title and Ifconfig commands to send.

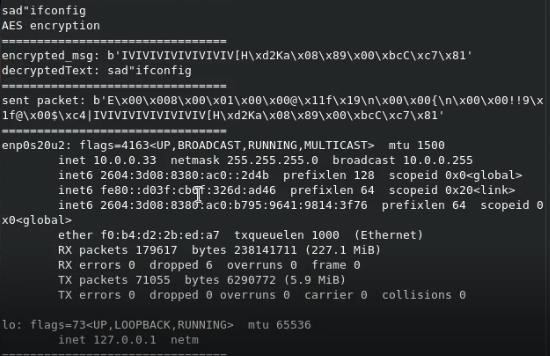


The result is back to the attack machine with Ifconfig of the victim machine. It shows victim machine IP address, network card names, IPv6 address, flags and etc.



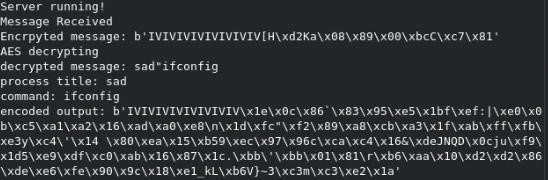


The Wireshark capture shows that the attacker sends the command to the victim and get the result. The result and the command are encrypted. Because the Ifconfig result is large, so the encryption will be extensive too.



Also, the print out in the console shows the encrypted message being sent and the decrypted result.

Victim



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. The following is the complete encrypted data.

This test means the Ifconfig command is a success.

## Test Case 5 – Create, Execute and Remove an Executable

Test case one will demonstrate list files in the current directory remotely to the victim machine.

* Destination IP: 10.0.0.33
* Source IP: 10.0.0.123
* Camouflaged title: sad
* Command: echo / bash / less / rm

### Attacker

#### Create Executable

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

First, I will send “echo echo hello > hi.sh” to create a executable bash file.

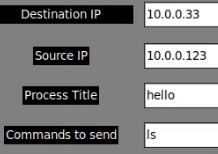


The console shows the plaintext command and encrypted command.



#### Validate The Executable

Next, I need to check if the file has been created. I use “ls” command to list all files in that directory.



The result shows that the hi.sh has been created.



The console shows the plaintext command and encrypted command.



Then I want to use less command to check the content of the executable.



The result shows that the executable is what I expected.



The console shows the plaintext command and encrypted command.



#### Execute The Executable

Then I need to run the hi.sh remotely. I used bash hi.sh command.



The result shows that it echoed hello in the console.

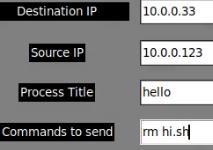


The console shows the plaintext command and encrypted command.



#### Remove The Executable

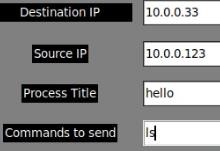
Now, after executing it, I want to remove it from the victim machine. I use rm hi.sh command.



The console shows the plaintext command and encrypted command.



Finally, I use ls command to check if the hi.sh was deleted or not.



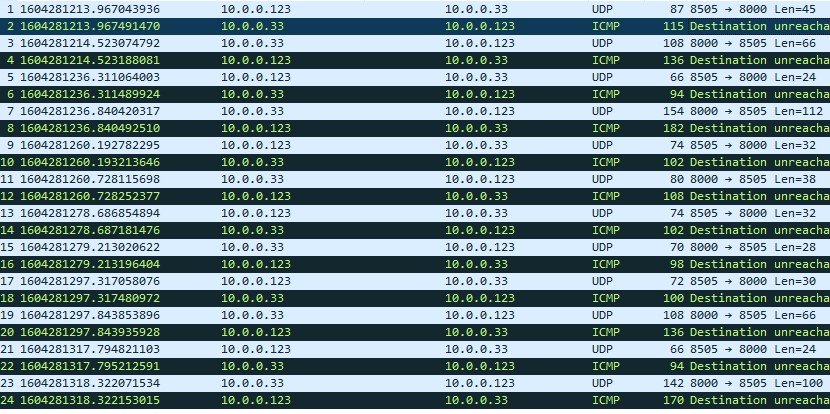
The result shows that hi.sh was removed.



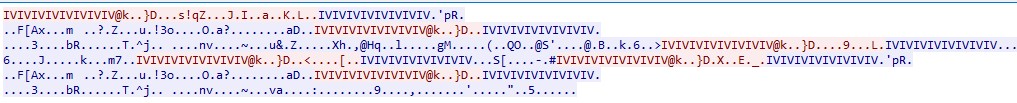
In total, I used six commands for all the steps:

1. Echo echo hello > hi.sh
2. Ls
3. Less hi.sh
4. Bash hi.sh
5. Rm hi.sh
6. Ls

Each step will generate 2 UDP requests. So in total, there will be 12 UDP requests. (ignore the ICMP bad requests.)



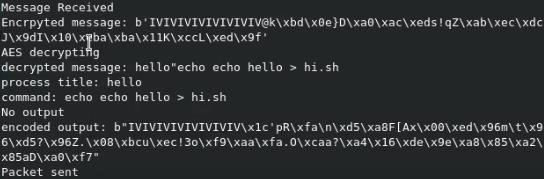
Wireshark also shows that each command and transmission is encrypted.



### Victim

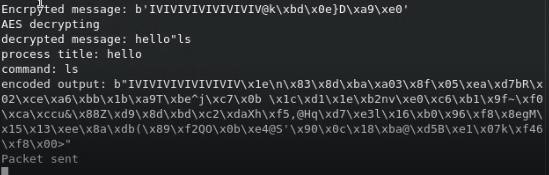
In total, I used six commands for all the steps, so the victim machine will react six times.

#### Echo echo hello > hi.sh



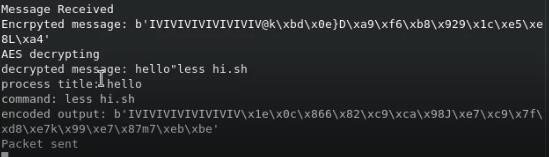
It shows the command and the title received, and the encrypted message that will send back.

*Ls*



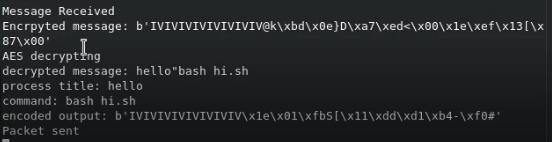
It shows the list file command, and the title received, and the encrypted message that will send back.

*Less hi.sh*



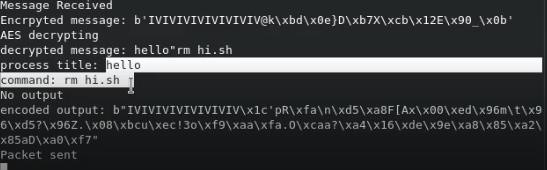
It shows the “less hi.sh” command, the title received, and the encrypted message sent back.

#### Bash hi.sh



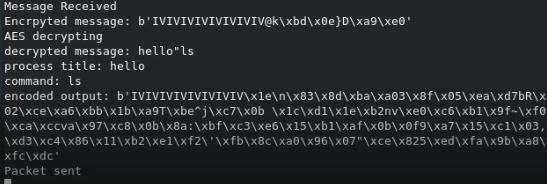
It shows the “bash hi.sh” command, the title received, and the encrypted message sent back.

*Rm hi.sh*



It shows the remove command, the title received, and the encrypted message sent back.

*Ls*



It shows the list file command, the title received, and the encrypted message sent back.

This test means the function to create, execute and remove an executable is a success.