Report: Plugboard Proxy

Shubham Agrawal - 113166701

This assignment develops a "plugboard" proxy for adding an extra layer of protection to publicly accessible network services.

The program pbproxy.go provides implementation to work as a client and a server. When running as a client, it connects to the pbproxy server which then relays all traffic to the actual service. Here, the pbproxy client encrypts the data using a static symmetric key. The pbproxy server before relaying the traffic, decrypts it using a static symmetric key.

Output and Program Usage:

Server

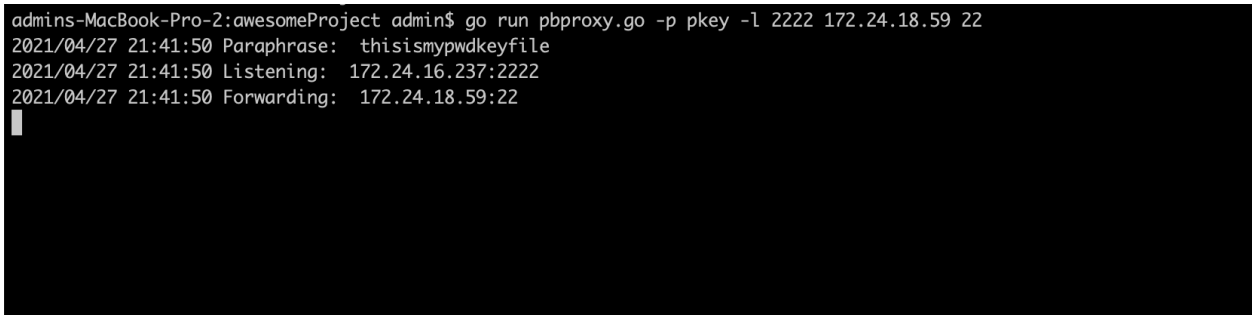
go run pbproxy.go [-l listenport] -p pwdfile destination port

-l Reverse-proxy mode: listen for inbound connections on <listenport> and relay them to <destination>:<port>

-p Use the ASCII text passphrase contained in <pwdfile>

1. **$ go run pbproxy.go -p pkey -l 2222 172.24.18.59 22**

“p” value takes the pwd file as input. This reverse proxy command starts listening on 2222 of the same host system and forwards all the incoming requests from 2222 to 172.24.28.59 (destination IP) and 22 (port)



Sample PWD File:

thisismypwdkeyfile

Invalid Input cases:

When Port is not provided in the reverse proxy command

```
admins-MacBook-Pro-2:awesomeProject admin$ go run pbproxy.go -p pkey -l 2222 172.24.18.59
2021/04/27 21:42:49 Hostname and port required
exit status 1
admins-MacBook-Pro-2:awesomeProject admin$
```

When PWD File is not provided

```
admins-MacBook-Pro-2:awesomeProject admin$ go run pbproxy.go -l 2222 172.24.18.59 22
2021/04/27 21:43:32 Missing password file input
exit status 1
admins-MacBook-Pro-2:awesomeProject admin$
```

Client

2. $ go run pbproxy.go -p pkey 172.24.16.237 2222

   -p Use the ASCII text passphrase contained in <pwdfile>

   Relay all the StdIn to the hostname and port provided as arguments

```
admins-MacBook-Pro-2:awesomeProject admin$ go run pbproxy.go -p pkey 172.24.16.237 2222
2021/04/27 21:46:28 Paraphrase:  thisismypwdkeyfile
2021/04/27 21:46:28 Forwarding:  172.24.16.237:2222
```

Invalid Inputs

When port is not provided

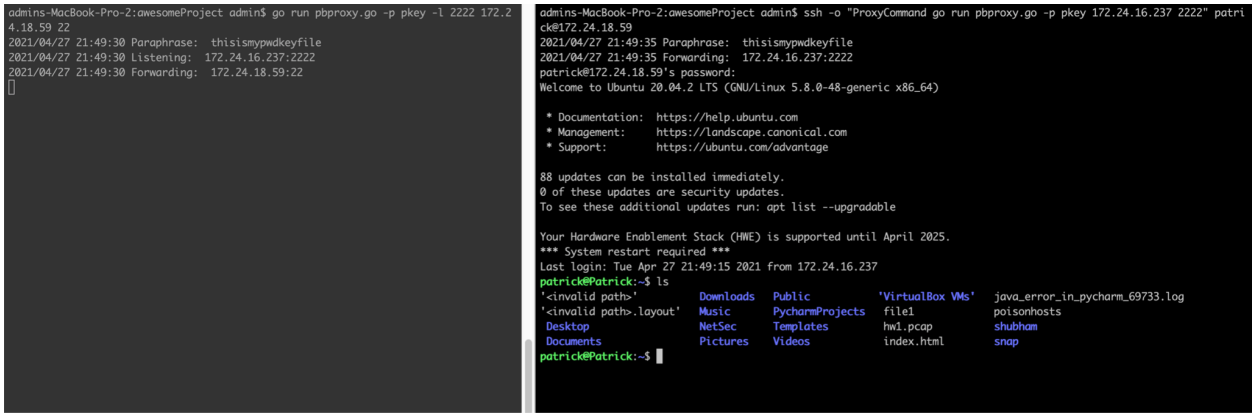
```
admins-MacBook-Pro-2:awesomeProject admin$ go run pbproxy.go -p pkey 172.24.16.237
2021/04/27 21:47:00 Hostname and port required
exit status 1
admins-MacBook-Pro-2:awesomeProject admin$
admins-MacBook-Pro-2:awesomeProject admin$
admins-MacBook-Pro-2:awesomeProject admin$
admins-MacBook-Pro-2:awesomeProject admin$
```

When PWD File is not provided

```
admins-MacBook-Pro-2:awesomeProject admin$ go run pbproxy.go 172.24.16.237 2222
2021/04/27 21:47:31 Missing password file input
exit status 1
admins-MacBook-Pro-2:awesomeProject admin$
admins-MacBook-Pro-2:awesomeProject admin$
admins-MacBook-Pro-2:awesomeProject admin$
admins-MacBook-Pro-2:awesomeProject admin$
```

Testing:

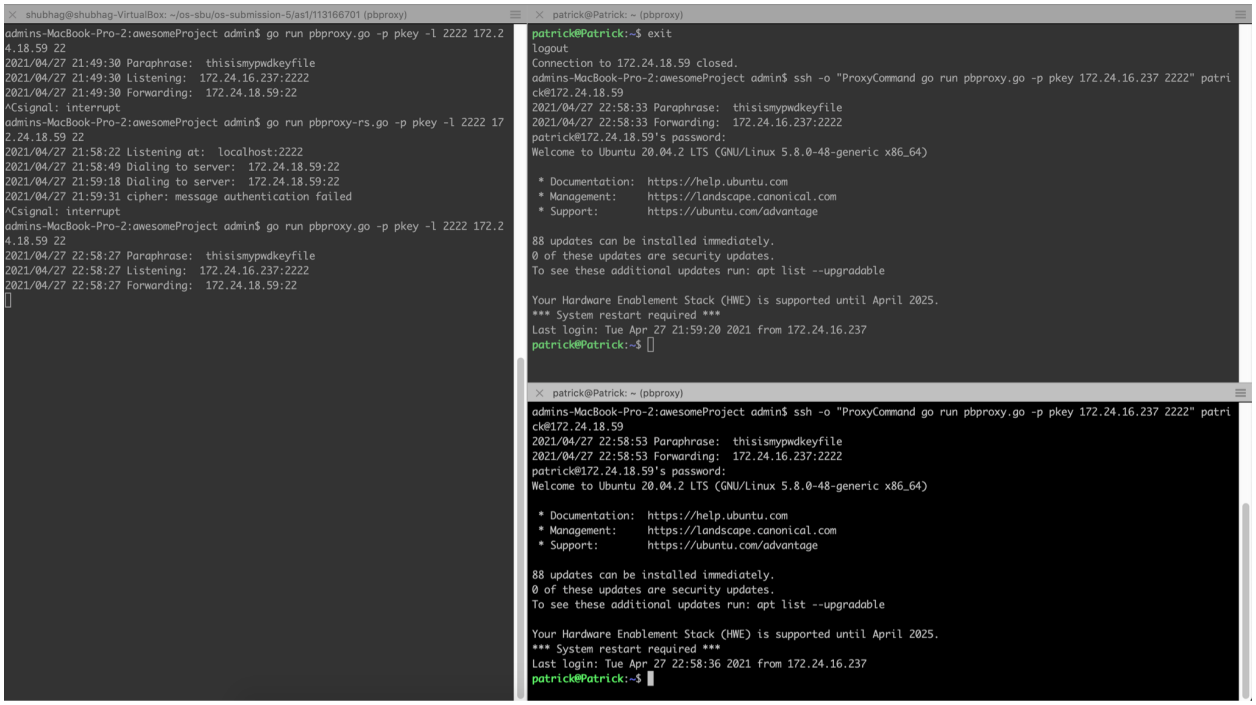
1. **Single Connection Testing**



On the left terminal, Running the server which listens on 2222 port on the local system and relays all the traffic to 22 port of 172.24.18.59 system

On the right terminal, Running the proxy command to connect to local system IP and port 2222. If this is successful , perform ssh to the 172.24.18.59 with the username.

2. **Multiple (Concurrent) Connection Testing**



Here, instead of one Proxy Command, running 2 proxy commands concurrently. We can see that both the clients (depicted on the right side) have successfully logged in to the server via the proxy server (depicted on left side).

Code/ Implementation Walkthrough:

1. Accept input parameter values , -l (for listen port), -p (for pwd file) and flag.Args (for Hostname and port)
2. For both server and client, pwd file is must. Hostname and port is also necessary. For clients, hostname and port refers to the address to which the encrypted message needs to be sent. For server, hostname and port refers to the address to which the incoming message needs to be forwarded to. If these three parameters are not present, it throws an error and the program stops
3. Read paraphrase from the file location provided in the pwd flag argument
4. Depending on the input parameters, deciding whether to run pbproxy as a server or a client. If listen port is provided, it acts as a server i.e. reverse proxy communication, else it acts as a client
5. For reverse proxy, it takes the IP of the system as the default IP and starts listening to it
6. For server, i.e. reverse proxy:
    a. We initiate a listen TCP Connection to the local default IP of the system and the port provided in the listen flag parameter
    b. After a new connection is established to the listen TCP connection, it initiates a thread which does the below task. Again, it keeps listening for new TCP connections to accept multiple requests
    c. We initiate a Dial TCP connection to the hostname and port provided as the arguments.
    d. We wait for any communication on the socket of listen and Dial TCP connections.
    e. As soon as any message is received from the listen TCP connection, we decrypt the message and write to the Dial TCP connection which acts as a reverse proxy feature.
    f. Similarly, as soon as we receive any message on the Dial TCP connection, we encrypt the message and write to the listen TCP connection to send to the client pbproxy.
    g. If the client disconnects, we close the connection since it is of no use and breaks the go thread as well which was writing to the same connection
7. For Client,
    a. We initiate a Dial TCP connection to the hostname and port provided as the arguments
    b. Take input using Stdin
    c. Encrypt the input data and write to the Dial TCP connection established above
    d. In a parallel go routine, keep listening on the Dial TCP connection for any incoming message
    e. As soon as any message is received from the Dial TCP connection, decrypt the data and write to Stdout and flush the buffer

8. Encryption Logic:
    a. Make a random salt
    b. Extract AES key from the paraphrase using PBKDF2 library from the random generated salt and paraphrase using AES-256 in GCM mode
    c. Cipher blocks are initialized
    d. Make Random Nonce
    e. Cipher the plain text using the nonce and key generated, add nonce to the cipher text as well so that the receiving end will be able to extract the nonce
    f. Appending the salt with the cipher text received above so that it can be extracted at the receiving end.
9. Decryption Logic:
    a. Extract salt and cipher text with Nonce from the cipher data received
    b. Extract AES key from the paraphrase using PBKDF2 library from the salt extracted above and paraphrase using AES-256 in GCM mode
    c. Cipher blocks are initialized
    d. Extract nonce from the remaining data to get data without Nonce
    e. Get the plain text using the AESGCM library
    f. Return the plain text

PS: There should not be any conversions like string, etc. on the data. During SSH, the client and server communicate data which can get tampered while string or other conversion and thus, fail the SSH.

References:

1. https://cryptobook.nakov.com/symmetric-key-ciphers/cipher-block-modes
2. https://pkg.go.dev/net
3. https://pkg.go.dev/crypto/cipher
4. https://pkg.go.dev/golang.org/x/crypto/pbkdf2
5. https://medium.com/@yanzay/implementing-simple-netcat-using-go-bbab37507635
6. https://stackoverflow.com/questions/8461462/how-can-i-use-go-append-with-two-byte-slices-or-arrays
7. https://play.golang.org/p/BDt3qEQ_2H
8. https://github.com/vfedoroff/go-netcat/blob/master/main.go

Submissions:

1. pbproxy.go - go program for server/client
2. pwd file - key file used to test the program
3. Report