1.0 - Objective

Create a insecure transmission at application level for training university newcomers into the ciber-security area.

Its recommended to not read the python code if you want to really break the communication playing like an external element

2.1 - requirements and characteristics:

2.1.1 - Requirements

R01 - The protocol simulates a secure private key exchange and remote execution of commands in the objective

R02 - The protocol is split in phases with independent connections between them

R03 - The client and server have a pre-shared value the is represented as “CommonValue”

R04 - The client and the server generates random privates values with timelife and they refresh that values when the lifetime ends

2.1.2 - Execution characteristics

There are two ways to execute the .py file:

1. Without arguments, the program will create only a server in the default port 4450
2. With the ip and port argument, the program will create a server in the specified port and tries to communicate with the specified address at the same port (Can be used with local-host to connect with itself)

2.1.3 - Environment

Executed with Python 3.8

Executed in windows environments

3.0 - Connection characteristics

C01 - For each Client to Server connection, we have a data send from the client with the request and a response from the server with the data related to the request.

C02 - After each server to client response each connection close.

C03 - Each message between the client and the server is authenticated by a TimeFlag.

------------------------------------Spoiler Alert Below(Protocol characteristics)-----------------------------------------

4.0 - Protocol phases ( In each phase the client opens a new connection with the server and in the end of the phase the connection is closed.

4.1 - Protocol characteristics:

4.1C01- There is always an identification header for each phase

4.1C02- The TimeFlag has always the reserved value “/” before

4.1C03- If we are sending data, after the identification header we have “:” and the data that we want to transfer

4.1C04- The TimeFlag mandatory in all the messages

4.1.2 - Characteristics of the execution windows

4.1.2C01 - Each window has a defined randomized token

4.1.2C02 - The windows are sequential one after another

4.1.2C03 - The lifetime of each window is defined by a random integer in the magnitude of seconds

4.1.2C04 - The server always send to the client the next operative window but only the starting moment

4.2 - Phases

4.2.1 - Phase 01

-Schematic-

C->S DH\_1/TimeFlag

S->C DH\_1:int(CommonValue + ServerPrivateValue)/TimeFlag

-Comment-

C->S;Petition from the client: Identifier "DH\_1/"

S->C;Response from the server: Identifier "DH\_1:" and common value pre-shared between them added to the private int value of the server

4.2.2 - Phase 02

-Schematic-

C->S DH\_2:int(ServerPrivateValue + ClientPrivateValue)/TimeFlag

S->C DH\_2:int(ClientPrivateValue+TimeToNexToken,ClientPrivateValue+Executiontoken)

-Comment-

C->S;Petition from the client: Identifier "DH\_2:" and the sum of the private server value and the client private value

S->C;Response of the server: Identifier "DH\_2:" and the sums separated by “,” of :

1)ClientPrivateValue and TimeToNexToken

2)ClientPrivateValue and Executiontoken

4.2.3 - Phase 03

-Schematic-

C->S PVT\_1:int(Executiontoken+CharDecValFromAscii1,Executiontoken+CharDecValFromAscii2,...)/TimeFlag

S->C PVT\_1:int(ClientPrivateValue+CharDecValFromAscii1,ClientPrivateValue+CharDecValFromAscii2,...)/TimeFlag

-Comment-

C->S;Petition from the client: Identifier "PVT\_1:" and the sum of the AscII value of each letter of our command add to the Executiontoken

S->C;Response of the server: Identifier "PVT\_1:" and the sum of the AscII value of each letter of the result add to the ClientPrivateValue

4.3 - Protocol flow diagram

