CY350X - Computer Networks

Project #3 – Covert ICMP Messaging Channel

A CIA agent has successfully infiltrated FSO (the Russian equivalent of the NSA) and needs a covert means to pass information from their internal network to the case officer at a remote location. The network is highly secure, employing layered IPS, IDS, AI-driven firewalls, and real-time traffic monitoring. As a highly motivated Cyber Officer, you have been tasked with solving this problem.

In this programming assignment, you will design a network client that hides secret messages in the data section of an ICMP Echo packet (like that used in the *ping* program), as well as a network server that captures the Echo packets and decrypts/prints the secret messages. The packets MUST appear to be normal ICMP echo traffic, otherwise, the CIA agent will be compromised. You will write your code in Python 3, and your final code must execute on your assigned VM.

Checkpoint due 16 April 2023 at 2359.

Final project due 28 April 2023 at 2359.

Requirements

1. You must use the *scapy* library to craft your packets. A client shell (*client_shell.py*), server shell (*server_shell.py*), and config file to manage global variables (*config.py*) are provided.

Client

1. The client must create a properly formatted ICMP Echo packet with a text message in the data section and send it over the network to a remote server (that you will build). The packets you send must appear in Wireshark to be valid, properly formatted, ICMP Echo packets and should generate ICMP Echo Replies.
2. You cannot “hardcode” the secret message. It must come from either the user via user input or from an external file.
3. You cannot “hardcode” the destination address. It must come from either the user via user input or from an external file.
4. The secret message in the data section of the packet must be encrypted with symmetric encryption. You may assume the key has already been shared between the client and server. **You must use an encryption technique that is more robust than a simple Caesar cipher.** This can be from another Python library/module.
5. Your client must only send up to 80 bytes of the encrypted message per packet. Additionally, each packet in a series of packets containing the same message must be the same length. (If you had 120 bytes of an encrypted message, you should be sending 2x packets with each carrying 80 bytes of data.)

Server

1. The server will receive the ICMP Echo packet and print the message in human-readable text.
2. Your server must be able to recover and decode a message sent across multiple ICMP Echo packets.

3. Note: Your server does not need to generate ICMP Echo Reply packets. The target machine will generate those automatically in response to the ICMP Echo it receives.

- You may develop the project on any machine you choose, but it must run successfully on your VM.

Bonus Features

1. Two-Way Transmission: Implement the client and server such that both can send and receive covert data (will require multithreading). (5 points)
2. Image Transfer: Send a picture from the client to the server. (5 points)

Submissions

Checkpoint

You will submit:

- The RFC number for ICMP
- The encryption library you intend to use (this is a non-binding assertion)
- The ICMP types and codes you will need
- Two flowcharts of the basic structure of your program: one for your client and one for your server

Final Submission

Upload your client and server code to your canvas turn-in link. Files should be named as follows:

- <you>_client.py
- <you>_server.py
- Optional: <you>_client.py
- You must attach a signed coversheet to your submission.
- Along with your code, you will submit a one-page analysis of your design philosophy, problems that you encountered during implementation, and lessons learned.

Resources

Scapy.net and the scapy documentation and Python.org's library and documentation.
The zip folder *project3_shell_files* has three Python files you can start with.

Pro Tips

- Understand what layer ICMP functions at, and the appropriate other layers you will need to build.
- Ensure that you are using Python 3.
- You will have to run your program as root (sudo).
- The RFC for ICMP is very helpful.
- Start with a simple client and server program that sends data from client to server, then build up from there. Start without using any encryption.
- Get your programs running with localhost before you try to send traffic across the network.
- Wireshark is also very helpful. You can sniff the loopback interface while developing on localhost.
- Test your additions as you go. If you have a working program, back it up before adding features.