Atividade de Criptografia

Ferramentas

- Python 3.8.11;
- Biblioteca de criptografia do Python (cryptography.fernet);
- Wireshark;

Objetivos

- Criar uma aplicação de comunicação escrita peer-to-peer (P2P)
- Garantir a confidencialidade da comunicação
- Utilizar a arquitetura cliente-servidor
- Utilizar Sockets
- Verificar se a política de segurança é satisfeita por meio de um sniffer de rede.
- Alterar o código fonte do programa de tal sorte que não seja possível observar o comportamento do sniffer de rede
- Garantir que todos os clientes consigam ver a informação apresentada na interface do terminal e que o fluxo TCP observado no Wireshark seja completamente indiscernível

Procedimentos

Foi usado um código de uma aplicação em Python que estabelece comunicação entre diferentes clientes utilizando um servidor (nó central) por meio de sockets. A aplicação possui dois arquivos: o Serve.py e o Client.py. O primeiro é o servidor cujo IP para fins de teste é o 127.0.0.1 (seu próprio computador) e opera na porta 5535 (é possível alterar diretamente no script ou tornar a porta selecionável pelo usuário). Já o Cliente.py é utilizado para comunicar-se com o servidor como uma sala de bate-papo. Sugere-se que abra um terminal para cada instância (primeiro servidor e depois cliente) e ao final em um novo terminal execute um novo cliente, assim quando enviar uma mensagem pelo cliente do Terminal 2 o do Terminal 3 também receberá a mensagem.

Abaixo estão listados comandos e atalhos para observar o fluxo TCP da aplicação utilizada.

- Antes de ligar servidor e cliente inicialize o Wireshark e a captura de pacotes na interface de loopback (em alguns casos pode estar como lo ou 127.0.0.1).
- Abra o servidor e quantos clientes quiser.
- Envie mensagens de um cliente para o outro.
- Pause a captura de pacotes.
- Na barra de filtros do Wireshark digite "tcp.port == 5535"desta forma apenas os pacotes cuja porta TCP 5535 (porta padrão configurada no arquivo, se você alterar a porta no código fonte deve também alterar a porta aqui) aparecerão na lista de pacotes.
- Selecione um pacote e aperte CTRL + ALT + SHIFT + T

Resultados

A figura abaixo mostra a troca de mensagem sem criptografia usando dois clientes:

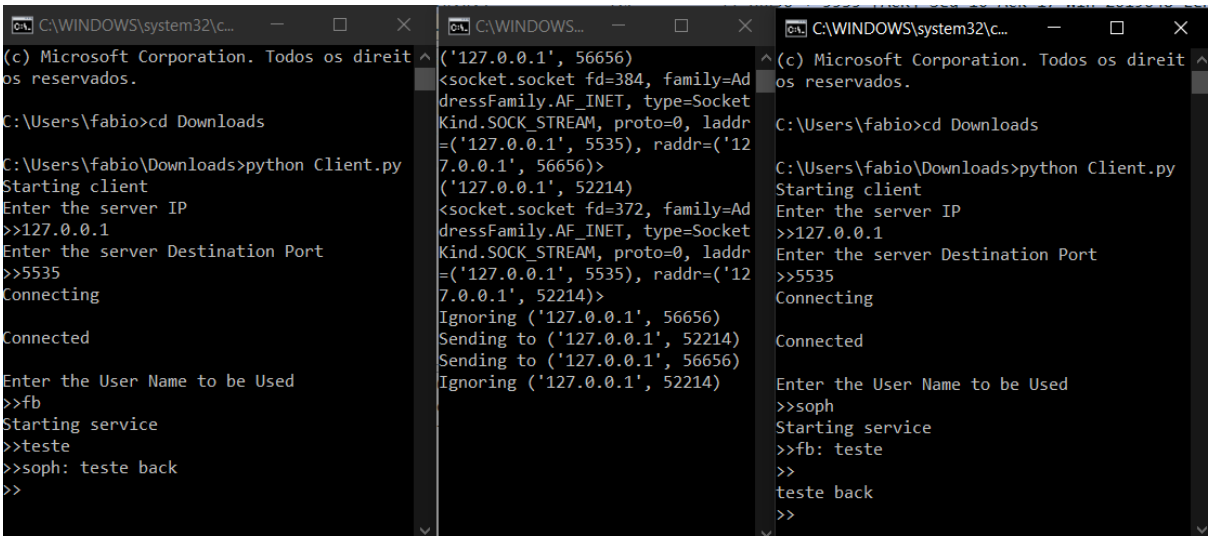

Figura 1

A figura 2 mostra através do wireshark que as mensagens do cliente 1 podem ser lidas pelo sniffer de rede pois não estão criptografadas:

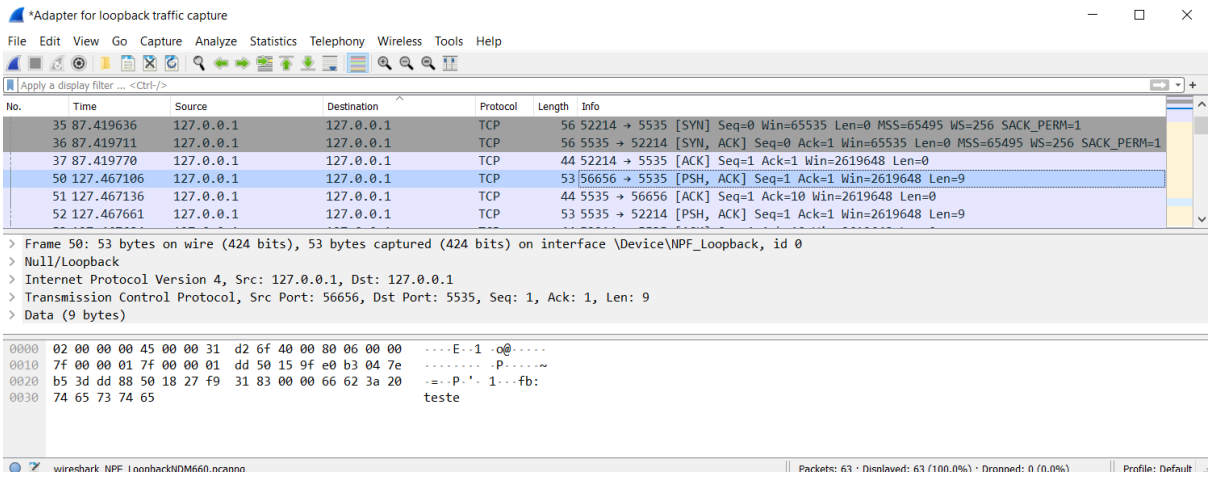

Figura 2

A figura 3 mostra através do wireshark que as mensagens do cliente 2 podem ser lidas pelo sniffer de rede pois não estão criptografadas:

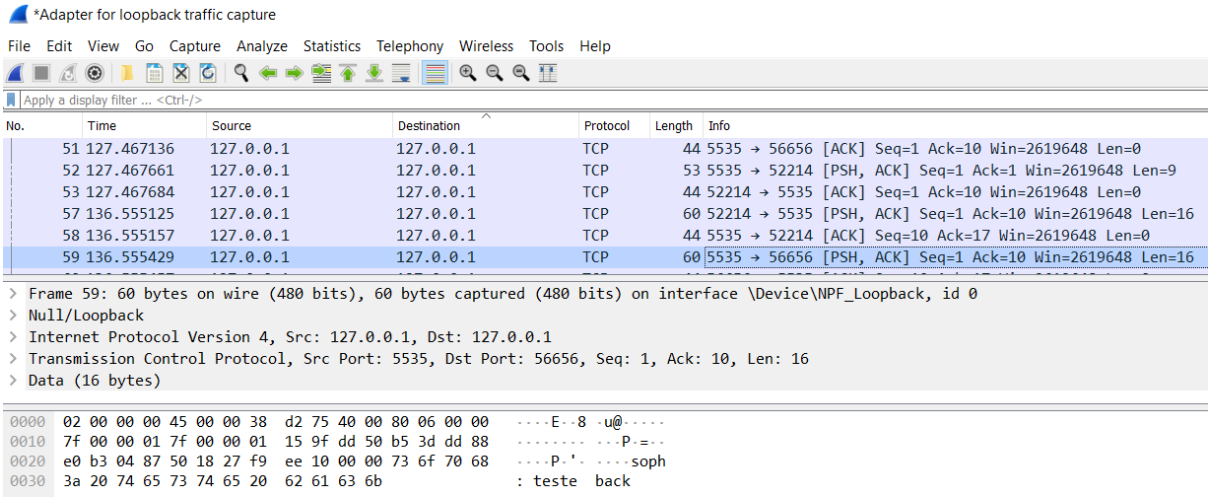

Figura 3

A figura abaixo mostra a troca de mensagem usando dois clientes já com a criptografia implementada:

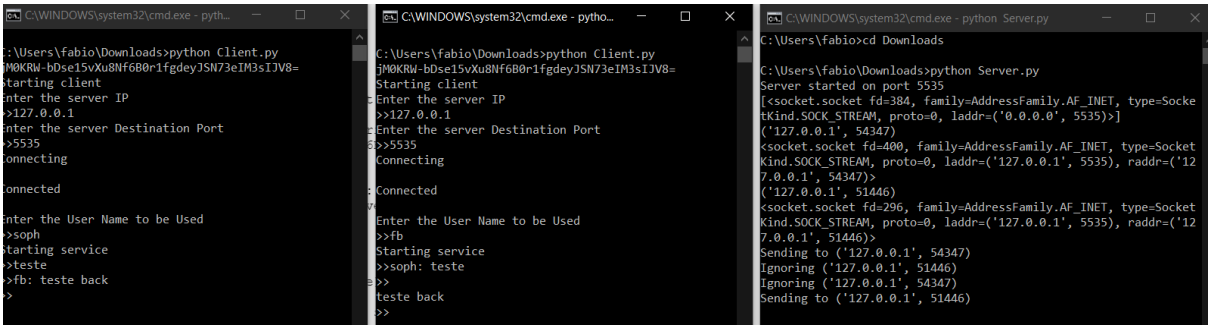

Figura 4

A figura 5 mostra através do wireshark que as mensagens do cliente 1 já não podem ser lidas pelo sniffer de rede pois estão criptografadas:

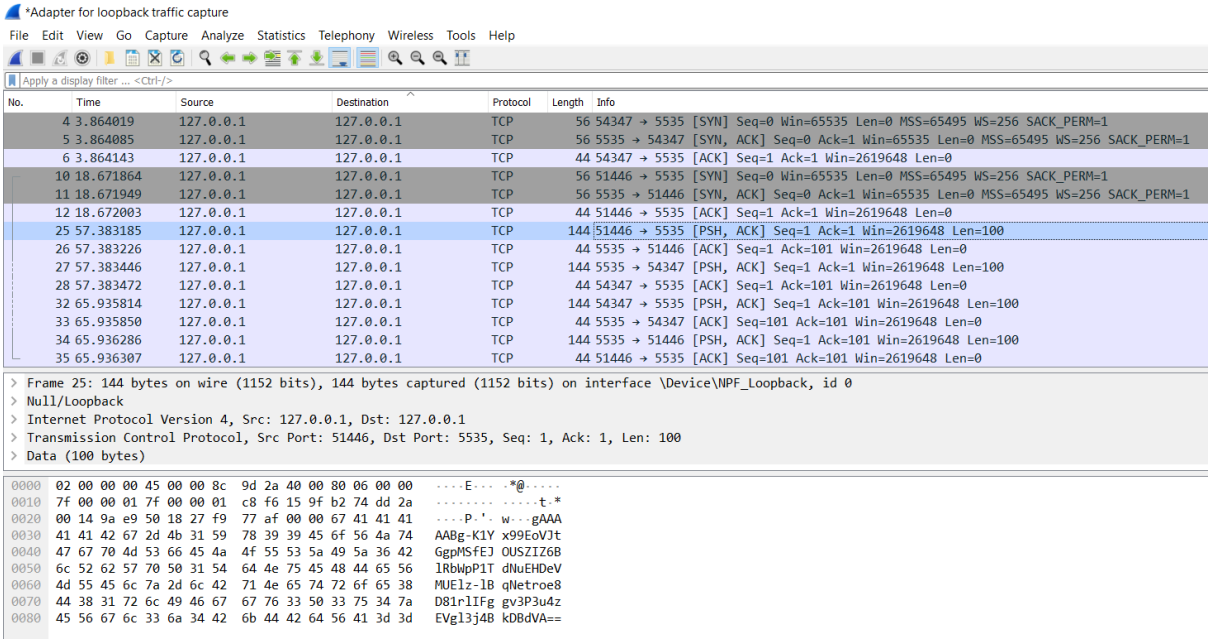

Figura 5

A figura 6 mostra através do wireshark que as mensagens do cliente 2 já não podem ser lidas pelo sniffer de rede pois estão criptografadas:

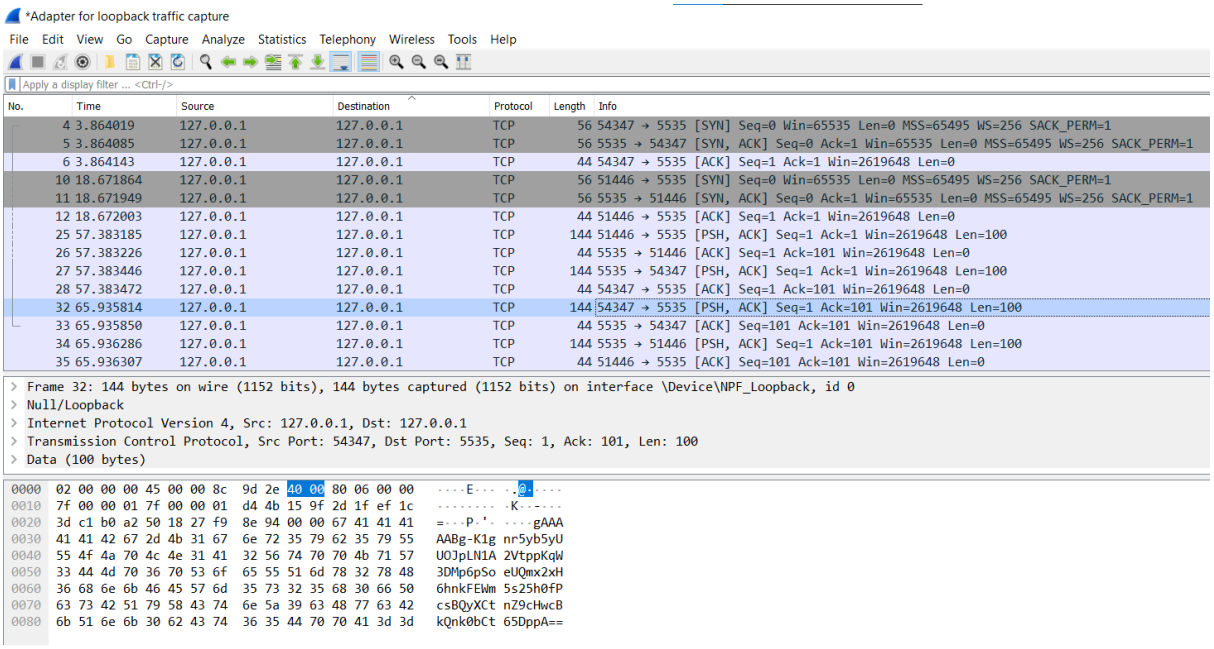

Figura 6

Por último na figura 7 o teste da execução do programa somente com um cliente e fazendo a demonstração de uso do comando “tcp.port == 5535”:

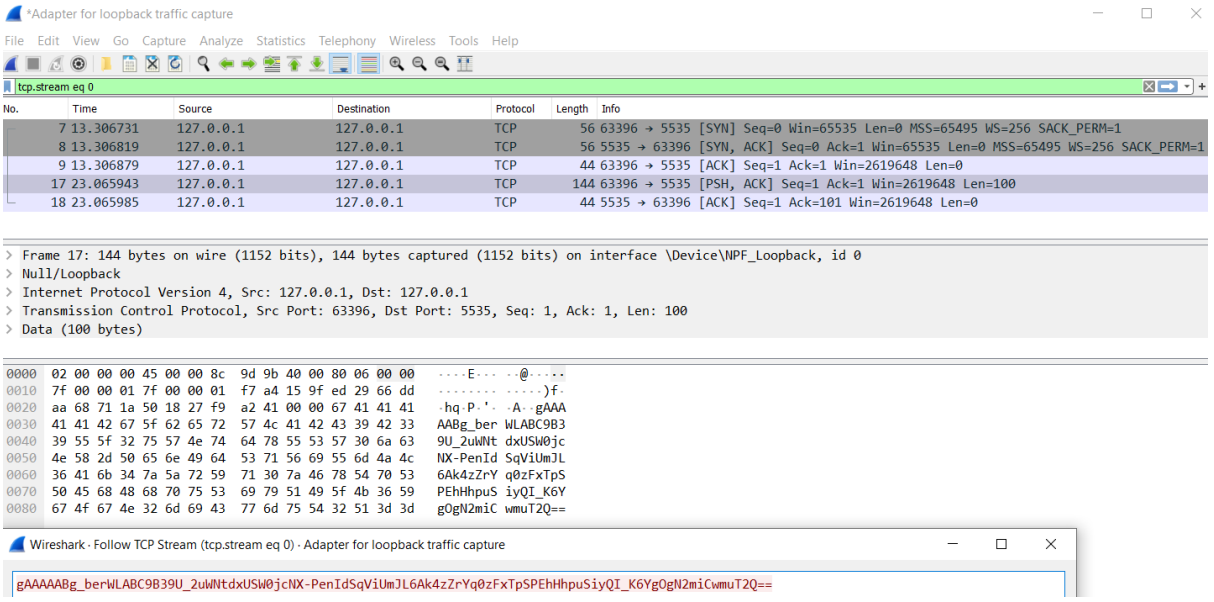

Figura 7

Conclusão

Uma vulnerabilidade observada nessa implementação é que a mesma chave é utilizada tanto para criptografar, quanto para descriptografar as mensagens, tornando o sistema menos seguro. A chave também está escrita em texto pleno no arquivo fonte do cliente, uma alternativa para contornar isso seria implementar por meio da programação um código que resgata-se a chave de um arquivo externo, isso facilitaria a atualização das chaves e adicionaria mais uma barreira de proteção. Para um uso mais adequado “Fernet.generate_key()” também poderá ser usado.

Bibliografia

FERNET. Symmetric encryption. Disponível em: https://github.com/pyca/cryptography/blob/main/docs/fernet.rst. Acesso em: 26 de julho de 2021.

GITHUB. Código Fonte Original da Aplicação. Disponível em: https://github.com/grakshith/p2p-chat-python. Acesso em: 26 de julho de 2021.