

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:

```

(c) Microsoft Corporation. Todos os direitos reservados.
C:\Users\fabio>cd Downloads
C:\Users\fabio\Downloads>python Client.py
Starting client
Enter the server IP
>>127.0.0.1
Enter the server Destination Port
>>5535
Connecting
Connected
Enter the User Name to be Used
>>fb
Starting service
>>teste
>>soph: teste back
>>

('127.0.0.1', 56656)
<socket.socket fd=384, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0, laddr=('127.0.0.1', 5535), raddr=('127.0.0.1', 56656)>
('127.0.0.1', 52214)
<socket.socket fd=372, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0, laddr=('127.0.0.1', 5535), raddr=('127.0.0.1', 52214)>
Ignoring ('127.0.0.1', 56656)
Sending to ('127.0.0.1', 52214)
Sending to ('127.0.0.1', 56656)
Ignoring ('127.0.0.1', 52214)

(c) Microsoft Corporation. Todos os direitos reservados.
C:\Users\fabio>cd Downloads
C:\Users\fabio\Downloads>python Client.py
Starting client
Enter the server IP
>>127.0.0.1
Enter the server Destination Port
>>5535
Connecting
Connected
Enter the User Name to be Used
>>soph
Starting service
>>fb: teste
>>
>>teste back
>>
  
```

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:

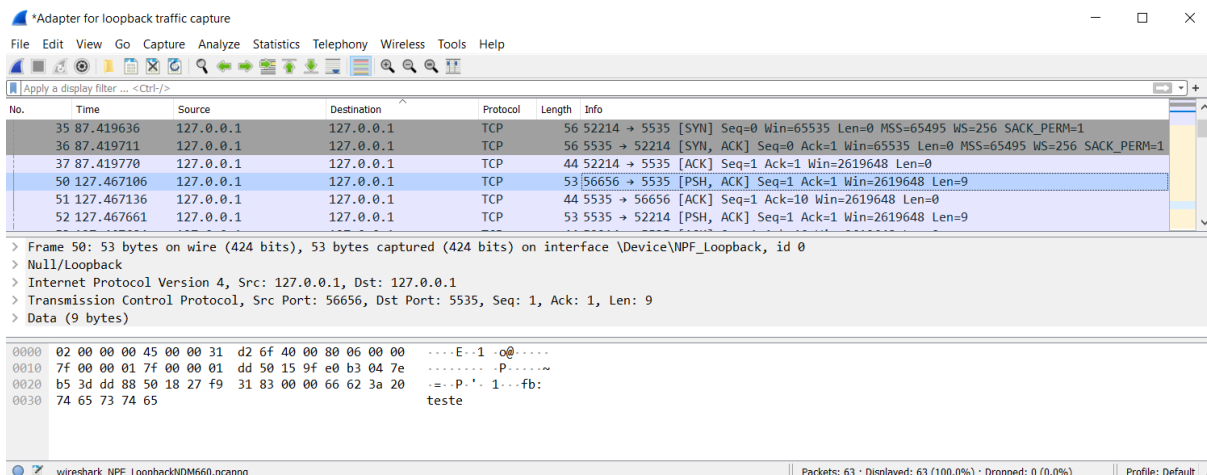


Figura 2

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

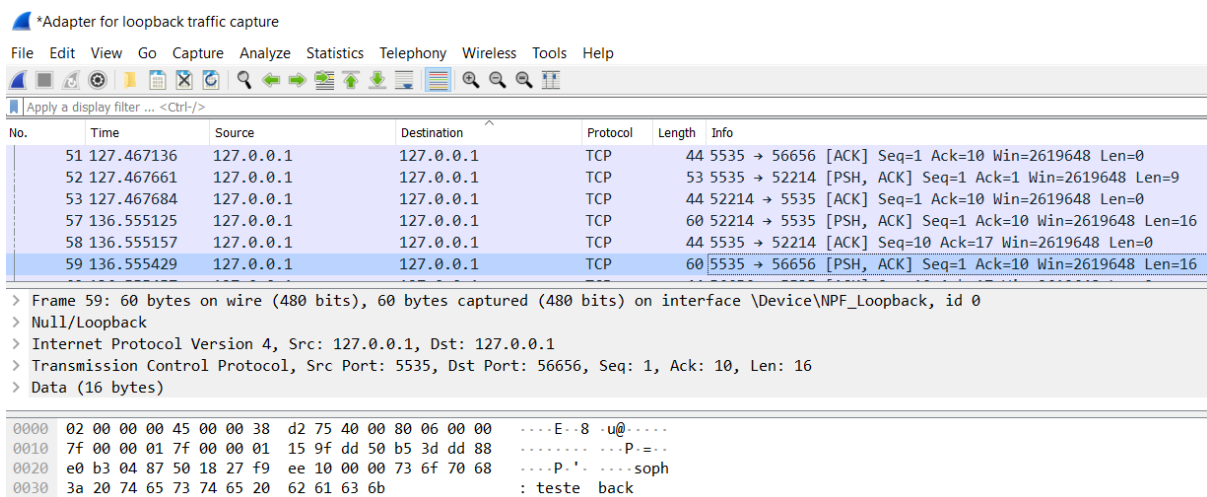


Figura 3

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

```
C:\WINDOWS\system32\cmd.exe - pyth...
C:\WINDOWS\system32\cmd.exe - pyth...
C:\WINDOWS\system32\cmd.exe - python Server.py

C:\Users\fabio\Downloads>python Client.py
MOKRW-bDse15vXu8Nf6B0r1fgdeyJ5N73eIM3sIJV8=
Starting client
Enter the server IP
>127.0.0.1
Enter the server Destination Port
>5535
Connecting
Connected
Enter the User Name to be Used
>soph
Starting service
>teste
>fb: teste back
>

C:\Users\fabio\Downloads>python Client.py
MOKRW-bDse15vXu8Nf6B0r1fgdeyJ5N73eIM3sIJV8=
Starting client
Enter the server IP
>127.0.0.1
Enter the server Destination Port
>5535
Connecting
Connected
Enter the User Name to be Used
>fb
Starting service
>soph: teste
>
>teste back
>

C:\Users\fabio>cd Downloads
C:\Users\fabio\Downloads>python Server.py
Server started on port 5535
[<socket.socket fd=384, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0, laddr=('0.0.0.0', 5535)>]
('127.0.0.1', 54347)
<socket.socket fd=400, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0, laddr=('127.0.0.1', 5535), raddr=('127.0.0.1', 54347)>
('127.0.0.1', 51446)
<socket.socket fd=296, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0, laddr=('127.0.0.1', 5535), raddr=('127.0.0.1', 51446)>
Sending to ('127.0.0.1', 54347)
Ignoring ('127.0.0.1', 51446)
Ignoring ('127.0.0.1', 54347)
Sending to ('127.0.0.1', 51446)
```

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:

Adapter for loopback traffic capture

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl+>

No.	Time	Source	Destination	Protocol	Length	Info
4	3.864019	127.0.0.1	127.0.0.1	TCP	56	54347 → 5535 [SYN, Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1]
5	3.864085	127.0.0.1	127.0.0.1	TCP	56	5535 → 54347 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
6	3.864143	127.0.0.1	127.0.0.1	TCP	44	54347 → 5535 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
10	18.671864	127.0.0.1	127.0.0.1	TCP	56	51446 → 5535 [SYN, Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1]
11	18.671949	127.0.0.1	127.0.0.1	TCP	56	5535 → 51446 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
12	18.672003	127.0.0.1	127.0.0.1	TCP	44	51446 → 5535 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
25	57.383185	127.0.0.1	127.0.0.1	TCP	144	51446 → 5535 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=100
26	57.383226	127.0.0.1	127.0.0.1	TCP	44	5535 → 51446 [ACK] Seq=1 Ack=101 Win=2619648 Len=0
27	57.383446	127.0.0.1	127.0.0.1	TCP	144	5535 → 54347 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=100
28	57.383472	127.0.0.1	127.0.0.1	TCP	44	54347 → 5535 [ACK] Seq=1 Ack=101 Win=2619648 Len=0
32	65.935814	127.0.0.1	127.0.0.1	TCP	144	54347 → 5535 [PSH, ACK] Seq=1 Ack=101 Win=2619648 Len=100
33	65.935850	127.0.0.1	127.0.0.1	TCP	44	5535 → 54347 [ACK] Seq=101 Ack=101 Win=2619648 Len=0
34	65.936286	127.0.0.1	127.0.0.1	TCP	144	5535 → 51446 [PSH, ACK] Seq=1 Ack=101 Win=2619648 Len=100
35	65.936307	127.0.0.1	127.0.0.1	TCP	44	51446 → 5535 [ACK] Seq=101 Ack=101 Win=2619648 Len=0

> Frame 25: 144 bytes on wire (1152 bits), 144 bytes captured (1152 bits) on interface \Device\NPF_{Loopback, id 0

> Null/Loopback

> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

> Transmission Control Protocol, Src Port: 51446, Dst Port: 5535, Seq: 1, Ack: 1, Len: 100

> Data (100 bytes)

```
0000 02 00 00 00 45 00 00 8c 9d 2a 40 00 80 06 00 00 ....E...*@....
0010 7f 00 00 01 7f 00 00 01 c8 f6 15 9f b2 74 dd 2a .....t.*
0020 00 14 9a e9 50 18 27 f9 77 af 00 00 67 41 41 41 ...p.' w...gAAA
0030 41 41 42 67 2d 4b 31 59 78 39 39 45 6f 56 4a 74 AABg-KlY x99EoVjt
0040 47 67 70 4d 53 66 45 4a 4f 55 53 5a 49 5a 36 42 GgpMSfEj OUSZIZ6B
0050 6c 52 62 57 70 50 31 54 64 4e 75 45 48 44 65 56 lRbWpP1T dNuEHDeV
0060 4d 55 45 6c 7a 2d 6c 42 71 4e 65 74 72 6f 65 38 MUElZ-lB qNetroe8
0070 44 38 31 72 6c 49 46 67 67 76 33 50 33 75 34 7a D81r1IFg gv3P3u4z
0080 45 56 67 6c 33 6a 34 42 6b 44 42 64 56 41 3d 3d EVg13j4B kD8dVA==
```

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:

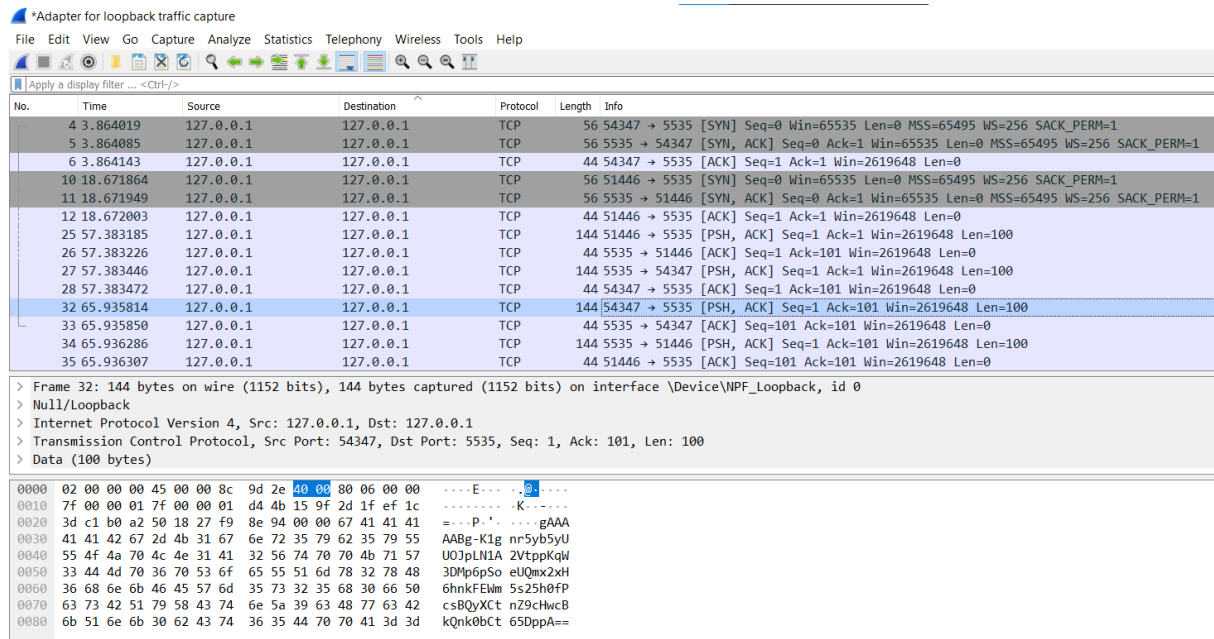


Figura 6

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”:

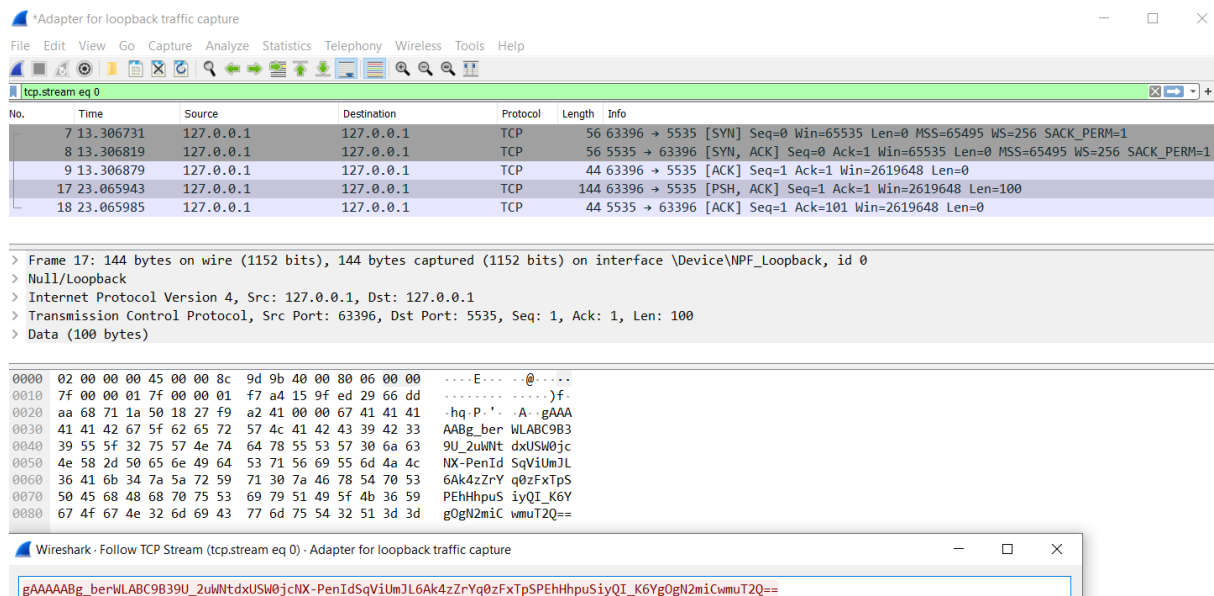


Figura 7

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.