

Segurança e Auditoria de Sistemas

Atividade 1

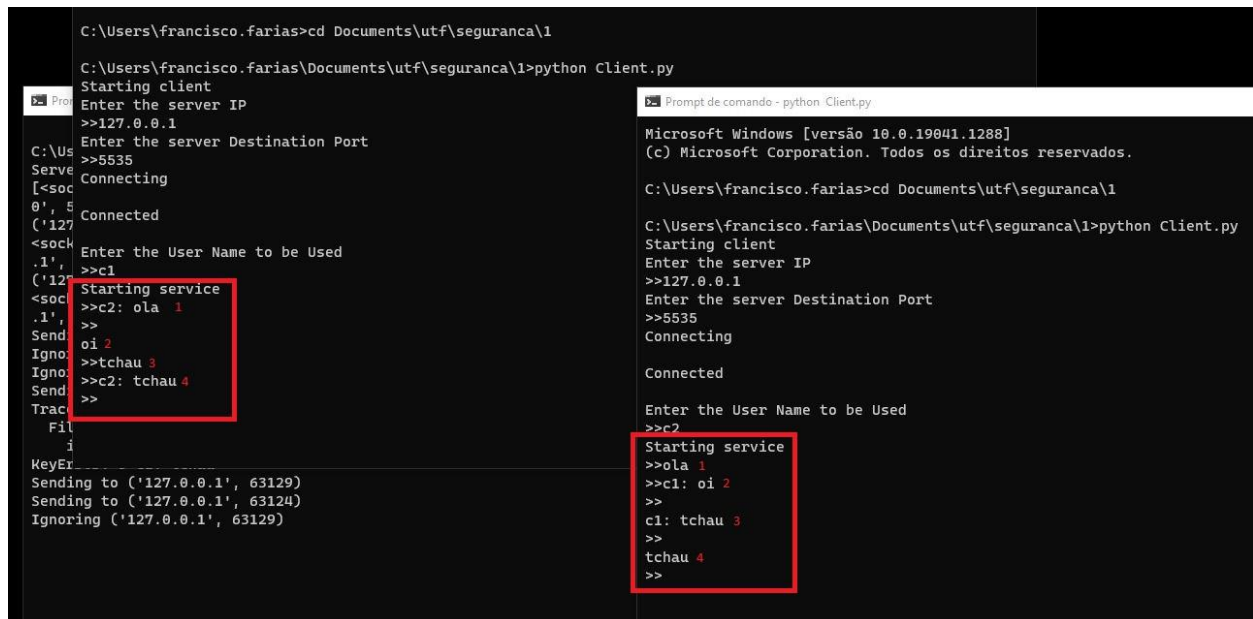
EC38D/C81 - Eng. de Software - UTFPR-CP - 2021/2

Francisco de Carvalho Farias Junior - RA: 1581660

fariasjunior96@gmail.com

A atividade

Foram fornecidos dois scripts em Python, um para o cliente e outro para o servidor, que implementam uma sala de chat peer-to-peer. Mais especificamente, cada cliente conectado a uma instância do servidor consegue enviar mensagens que são transmitidas a todos os outros clientes também conectados àquela instância. Para se conectar basta informar o IP e a porta que o servidor utiliza.



```
C:\Users\francisco.farias>cd Documents\utf\seguranca\1
C:\Users\francisco.farias\Documents\utf\seguranca\1>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
>>c1
Starting service
>>c2: ola 1
>>
>>oi 2
>>tchau 3
>>c2: tchau 4
>>
Sending to ('127.0.0.1', 63129)
Sending to ('127.0.0.1', 63124)
Ignoring ('127.0.0.1', 63129)

Microsoft Windows [versão 10.0.19041.1288]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\francisco.farias>cd Documents\utf\seguranca\1
C:\Users\francisco.farias\Documents\utf\seguranca\1>python Server.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
>>c2
Starting service
>>ola 1
>>c1: oi 2
>>
>>c1: tchau 3
>>
>>tchau 4
>>
```

Figura 1: Troca de mensagens básica entre cliente e servidor

Na implementação atual as mensagens são transmitidas em plaintext. Dessa forma, qualquer pessoa que conseguir capturar pacotes enviados entre dois nós dessa aplicação consegue de maneira trivial ler o conteúdo das mensagens trafegadas.

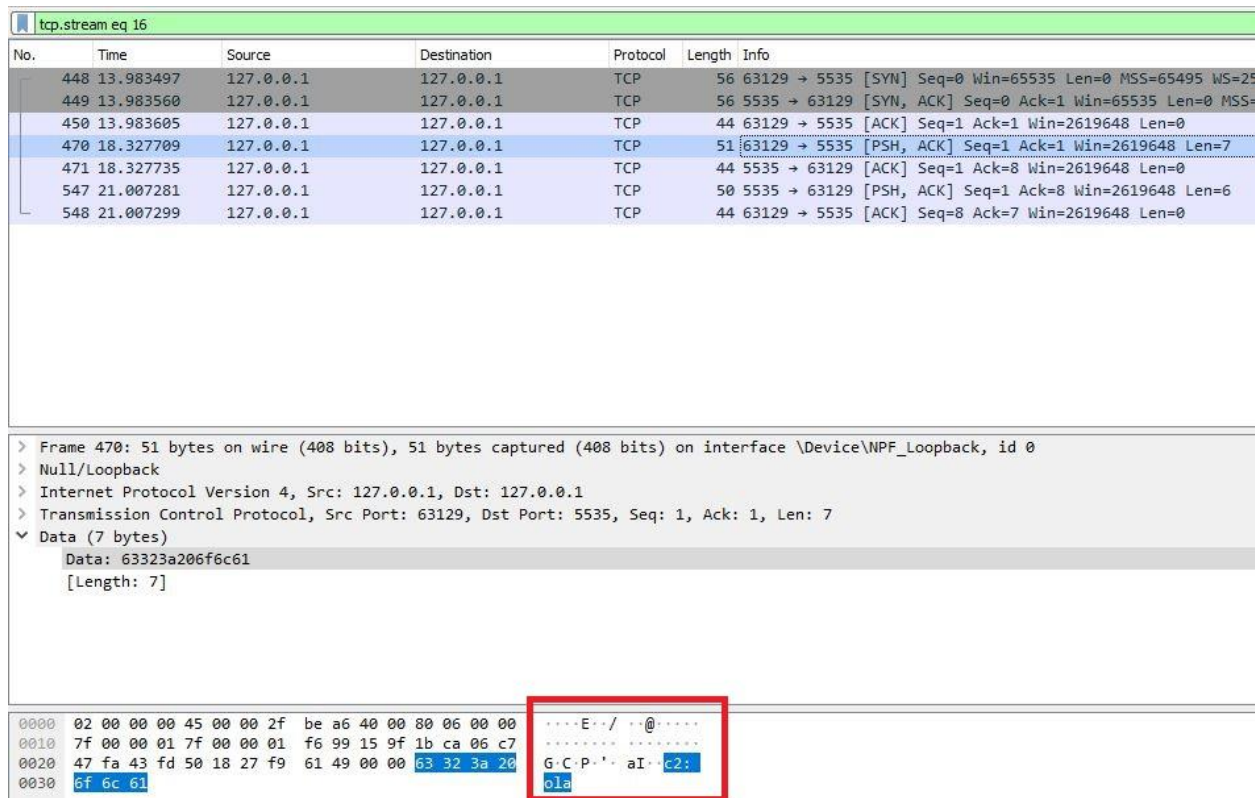


Figura 2: Inspeccionamento dos pacotes capturados durante o envio das mensagens 1 e 2, de acordo com a fig. 1. Em destaque, o envio da mensagem 1 ao servidor, com o conteúdo da mensagem em plaintext.

Essa atividade consiste em alterar os scripts implementando uma solução de criptografia que impede que terceiros que capturarem os pacotes consigam decifrar o conteúdo das mensagens sem impedir o funcionamento normal do chat.

Contexto

A criptografia é o estudo de técnicas que permitem ofuscar informação para que assim indivíduos consigam se comunicar de maneira segura, sem que partes indesejadas consigam entender a mensagem sendo transmitida.

As aplicações práticas para a criptografia são incontáveis e, especialmente durante a era da informação, o conceito de criptografia está estreitamente atrelado à ideia de segurança.

Para um webchat (o cenário usado na atividade atual) a criptografia representa uma funcionalidade extremamente importante, uma vez que ela é a diferença entre privacidade e exposição durante o seu uso.

A solução

Para a solução, foi implementada criptografia simétrica usando o algoritmo AES através da biblioteca Cryptography. A chave secreta fica pré-definida no código.

```
10 # >> libs usadas
11 import os
12 from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
13
14 # >> chave secreta
15 KEY = b'\xf2pz\x06)\x13\x8e\x9c\xd9\x94\xd8\n\x98u\xbfb\x8b\xf4*\xe4\x04e\xe5\xf9l,\x87\xf3d\x88\x99'
16
17 # >> tamanho do vetor de inicialização, em bytes
18 IV_SIZE = 16
19
20 # >> o tamanho da mensagem passada tem que ser multiplo de 128 (AES). essa função "enche linguça"
21 def fill_msg_length(msg):
22     length = len(msg)
23     fill = 128 - (length % 128)
24     return msg + ("\x00".encode() * fill)
25
26 ## >> implementação da encriptação; vetor de inicialização é passado no começo da mensagem
27 def encrypt(msg):
28     iv = os.urandom(IV_SIZE)
29     cipher = Cipher(algorithms.AES(KEY), modes.CBC(iv))
30     encryptor = cipher.encryptor()
31     encrypted_msg = encryptor.update(fill_msg_length(msg.encode())) + encryptor.finalize()
32     return iv + encrypted_msg
33
34 # >> implementação da decriptação; pega o vetor de inicialização no começo da mensagem
35 def decrypt(chunk):
36     iv = chunk[0:IV_SIZE]
37     encrypted_msg = chunk[IV_SIZE:]
38     cipher = Cipher(algorithms.AES(KEY), modes.CBC(iv))
39     decryptor = cipher.decryptor()
40     decrypted_msg = decryptor.update(encrypted_msg) + decryptor.finalize()
41     return decrypted_msg
42
```

Figura 3: Solução usada. AES com chave secreta pré-definida no código.

O vetor de inicialização tem um tamanho fixo, é gerado no momento da encriptação e é concatenado no começo da mensagem. Foi criada também uma função para concatenar bytes nulos na mensagem para que ela fique com tamanho múltiplo de 128, que é o tamanho dos blocos no algoritmo AES.

tcp.port eq 5537						
No.	Time	Source	Destination	Protocol	Length	Info
103	9.226573	127.0.0.1	127.0.0.1	TCP	56	62590 → 5537 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
104	9.226656	127.0.0.1	127.0.0.1	TCP	56	5537 → 62590 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
105	9.226701	127.0.0.1	127.0.0.1	TCP	44	62590 → 5537 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
216	20.136360	127.0.0.1	127.0.0.1	TCP	56	62596 → 5537 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
217	20.136470	127.0.0.1	127.0.0.1	TCP	56	5537 → 62596 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
218	20.136510	127.0.0.1	127.0.0.1	TCP	44	62596 → 5537 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
255	23.521936	127.0.0.1	127.0.0.1	TCP	188	62596 → 5537 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=144
256	23.521968	127.0.0.1	127.0.0.1	TCP	44	5537 → 62596 [ACK] Seq=1 Ack=145 Win=2619648 Len=0
257	23.522380	127.0.0.1	127.0.0.1	TCP	188	5537 → 62590 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=144
258	23.522442	127.0.0.1	127.0.0.1	TCP	44	62590 → 5537 [ACK] Seq=1 Ack=145 Win=2619648 Len=0
268	25.985081	127.0.0.1	127.0.0.1	TCP	188	62590 → 5537 [PSH, ACK] Seq=1 Ack=145 Win=2619648 Len=144
269	25.985120	127.0.0.1	127.0.0.1	TCP	44	5537 → 62590 [ACK] Seq=145 Ack=145 Win=2619648 Len=0
270	25.985885	127.0.0.1	127.0.0.1	TCP	188	5537 → 62596 [PSH, ACK] Seq=1 Ack=145 Win=2619648 Len=144
271	25.985921	127.0.0.1	127.0.0.1	TCP	44	62596 → 5537 [ACK] Seq=145 Ack=145 Win=2619648 Len=0


```

> Frame 255: 188 bytes on wire (1504 bits), 188 bytes captured (1504 bits) on interface \Device\NPF_{...}, id 0
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 62596, Dst Port: 5537, Seq: 1, Ack: 1, Len: 144
▼ Data (144 bytes)
  Data: 0a0682cdb5390d796b4f536b29c648388c5c6531c613ff1cfd375fed0a79056707ca281b...
  [Length: 144]
  0000 02 00 00 00 45 00 00 b8 e5 65 40 00 80 06 00 00 .....E...e@....
  0010 7f 00 00 01 7f 00 00 01 f4 84 15 a1 07 9c b8 6b .....P...S....k
  0020 f1 dd 06 91 50 18 27 f9 53 8d 00 00 0a 06 82 c0 .....P...S....k
  0030 05 39 0d 79 6b 4f 53 6b 29 c6 48 38 8c 5c 65 31 .....9.yk0Sk )-H8\el
  0040 c6 13 ff 1c fd 37 5f ed 0a 79 05 67 07 ca 28 1b .....7...y.g...
  0050 4d 34 c0 05 af fe c1 44 ab 65 c1 b9 61 24 09 a5 .....M.....D...e..a$.
  0060 f7 27 50 91 09 55 fe be 30 b0 19 ee 7f 0f cb 2f .....P...U...0...../
  0070 62 3d 09 f8 f5 49 04 58 5a 84 08 e1 5b b0 3e c5 .....b=...I-X Z...[->
  0080 88 3e 57 c0 12 11 4f 19 7e 5e 2b e0 93 92 49 4c .....>W...0...^+...IL
  0090 90 34 d1 1e d7 ce 43 45 6d 8c e1 9a 63 54 ec 17 .....4...CE m...cT...
  00a0 02 b0 27 d3 e7 27 32 98 38 49 5b df f7 d7 26 f1 .....2...8I[-&...
  00b0 2a 70 65 26 c5 f1 9d 34 94 ee 9e 48 .....pe@...4...H

```

Figura 4: A mesma sequência de mensagens ilustrada nas figuras 1 e 2, agora ofuscada durante a troca de pacotes.

Essa solução tem o problema óbvio de possuir a chave secreta embutida no código. Isso quer dizer que qualquer pessoa que tenha acesso ao cliente (especialmente no formato de script de Python) consegue facilmente extraí-la e usá-la para deciptar outras comunicações que usam essa mesma versão do cliente.

Além disso, essa solução não usa nenhum tipo de autenticação. Em situações comuns isso deixaria o sistema vulnerável a ataques CCA, onde mensagens são construídas para tentar deduzir a chave secreta baseando-se no resultado da deciptação. Porém meu entendimento é que esse ataque exige acesso à uma instância do cliente, e nesse caso fica mais fácil simplesmente pegar a chave direto do código como descrito anteriormente.

Recursos utilizados

<https://pypi.org/project/cryptography/>

<https://cryptography.io/en/latest/hazmat/primitives/symmetric-encryption/>

<https://docs.python.org/3/library/stdtypes.html#bytes>

https://en.wikipedia.org/wiki/Authenticated_encryption

https://en.wikipedia.org/wiki/Chosen-ciphertext_attack