INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
RIO GRANDE DO NORTE

Professor: Macêdo Firmino
Disciplina: Redes de Computadores
Prática 03: Programação com Socket e Criptografia em Python

Criptografia

Criptografia diz respeito ao conjunto de princípios e técnicas utilizados na proteção digital ou mecânica de informações. Hoje, ela é uma ferramenta de segurança amplamente utilizada nos meios de comunicação e consiste basicamente na transformação de determinado dado ou informação a fim de ocultar seu real significado. Em outras palavras, a criptográfia buscam solucionar problemas universais comuns a duas situações básicas: proteger informações armazenadas e utilizadas sempre pela mesma pessoa ou entidade, e proteger informações trocadas entre duas pessoas (ou entidades) diferentes.

Ela possui quatro objetivos principais, que são os princípios de confidencialidade, integridade, autenticação e não-repúdio, ou irretratabilidade.

Entre os algoritmos de criptografia temos os de chave simétrica que foi o primeiro tipo de criptografia criado. Os algoritmos que a utilizam essa abordagem têm como característica principal o uso de uma mesma chave criptográfica para criptografar ou descriptografar uma informação, por isso o adjetivo "simétrico" dá nome a esta técnica. Exemplificando um pouco este conceito, quando um emissor cifra uma mensagem com um algoritmo de criptografia simétrico, ele utiliza uma chave, que é representada por uma senha ou um conjunto de bits para codificar os dados. O receptor então faz uso do algoritmo para descriptografar a mensagem e aplica a mesma chave que foi utilizada pelo emissor para voltar à mensagem em sua forma original. Sem a mesma, não é possível decifrar a informação recebida.

PyCryptodome

O PyCryptodome é uma biblioteca em pyhton que trata de implementações de algoritmos de criptografia. Ele possui vários recursos úteis, por exemplo, Cifras simétricas (AES, DES e 3DES), Modos tradicionais de operações para cifras simétricas (BCE, CBC e CFB), Cifras de fluxo (Salsa20 e RC4), Hashes criptográficos (SHA e MD5), Códigos de Autenticação de Mensagens (HMAC e CMAC) e Cifras assimétricas (RSA).

A seguir será apresentado um exemplo de um código que faz uso do algoritmo AES do PyCryptodome para cifrar e decifrar uma mensagem de texto.

```
from Cryto.Cipher import AES
def encrypt(key, data):
    cipher = AES.new(key, AES.MODE_ECB)
    ciphertext = cipher.encrypt(data)
    return ciphertext
def decrypt (key, ciphertext):
    cipher = AES.new(key, AES.MODE_ECB)
    msg = cipher.decrypt(ciphertext)
    return msg
key = "ifrn"
if len(key) % 16 != 0:
    key += ' ' * (16 - len(key)%16)
data = "aula de seguranca"
if len(data) % 16 != 0:
    data += ' ' * (16 - len(data)%16)
ciphertext = encrypt(key, data)
print ciphertext
if len(ciphertext) % 16 != 0:
    ciphertext += ' ' * (16 - len(ciphertext)%16)
plaintext = decrypt(key, ciphertext)
print plaintext
```

Socket em Python

Uma API (Interface de Programa Aplicativo) especifica os detalhes de como um programa aplicativo interage com o software de protocolo. A API de socket, embora tenha sido originalmente desenvolvida para o sistema operacional UNIX, esta implementada em vários sistemas operacionais modernos. Esta API permite que aplicativos mandem e recebam mensagem para outros aplicativos. Estes outros aplicativos podem estar rodando tanto na máquina local quanto numa máquina remota. Em outras palavras, um programa vê esta ferramenta como um mecanismo de entrada-saída.

Em python a API de socket foi desenvolvida e pode ser utilizada pelos diversos programas. Para utilizar um socket em Python, o socket deverá ser importado através seguinte comando:

```
import socket
```

Quando cria um socket, um aplicativo deve especificar a família de protocolos a ser usada, como também o tipo de serviço desejado (orientado à conexão ou sem conexão). Em python a função que cria um socket UDP e TCP, sobre IPv4, é mostrado abaixo:

```
udp = socket.socket(socket.AF_INET,
socket.SOCK_DGRAM)

tcp = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
```

Depois de criar o socket, o programa servidor especifica um endereço local (porta) para o socket. A funcao bind() solicita ao sistema operacional (SO) para ligar a porta ao socket.

```
tcp.bind(orig)
udp.bind(orig)
```

Depois de especificar o endereço, o programa servidor no caso de um socket TCP se faz necessário colocar o socket para esperar solicitações de clientes. Para esta função é utilizado o comando listen(). Além disso, quando receber as solicitações deverão ser aceitas (utilizando o comando accept()).

```
tcp.listen(1)
con, cliente = tcp.accept()
```

Ainda no servidor, o programa deverá especificar o valor máximo de dados que podem ser recebidos em bytes. Para isso, é utilizado a função recv().

```
msg = con.recv(TAMANHO)
```

No caso de um programa cliente TCP, depois de criar um socket ele deverá especificar o endereço de um servidor remoto para estabelecer uma conexão. Em python a função connect() conecta com um endereco de socket remoto. Para isso, se faz necessário informar o endereço IP do host de destino e a porta para qual deserá estabelecer a conexão.

```
tcp.connect(dest)
```

Uma vez que um socket foi estabelecido, os aplicativos podem transferir informações. No caso do UDP é utilizada a função sendto(). Esta função envia dados para o socket. O valor de entrada da função é a mensagem e o endereco do socket (HOST e a PORTA).

```
udp.sendto (msg, dest)
```

No caso do TCP é utilizada a função send(). Esta função envia dados para o socket previamente estabelecido. O valor de entrada desta função é a mensagem a ser enviada.

```
tcp.send (msg)
```

Depois de transferir e receber os dados, tanto o socket TCP quanto o UDP deverá ser destruído. O o comando close() do objeto socket em python tem esta função.

```
udp.close()
tcp.close()
```

A seguir será apresentado um exemplo de um programa que faz uso do socket em python para criar um servidor TCP na porta 33333, que recebe um texto e envia o mesmo texto transformando todas as letras em maiúsculas.

```
from socket import *
serverPort = 33333
serverSocket = socket(AF_INET, SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
print 'O servidor esta pronto'
while 1:
    connectionSocket, addr = serverSocket.accept()
    message = connectionSocket.recv(1024)
    messageMaiuscula = message.upper()
    connectionSocket.send(messageMaiuscula)
    connectionSocket.close()
```

A seguir é apresentado um exemplo de um cliente que utiliza socket para mandar e receber informações para o servidor mostrado anteriormente.

```
from socket import *
serverName = '127.0.0.1'
serverPort = 33333
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
message = raw_input('Entre com a sentanca em minuculas: ')
clientSocket.send(message)
modifiedMessage = clientSocket.recv(1024)
print 'Mensagem do servidor: ', modifiedMessage
clientSocket.close()
```

Atividade

Faça uma aplicação cliente-servidor (chat simples) para demonstrar a programação de socket e criptografia simétrica:

- O clietne e o servidor utilizam criptografia de chave simétrica AES e a chave foi previamente compartilhada entre os dois.
- O servidor inicializa e fica aguardando conexão.
- Um cliente lê uma linha de caracteres (dados) do teclado, criptografa e a envia para o servidor.
- O servidor recebe os dados, decriptografa e mostra no terminar.
- O servidor também envia lê uma linha de caracteres (dados) do teclado, criptografa e a envia para o cliente.
- O cliente recebe os dados, decriptografa e apresenta em sua tela.
- Rode o Wireshark e veja o funcionamento do seu programa na rede.