Segurança e Privacidade de Dados

Trabalho 1

Mestrado em Engenharia Informática – Engenharia de Dados

2023 / 2024

1190384 – André Teixeira

1190818 – Luís Pinto

1231925 – João Henriques

Resumo

Este documento foi realizado no âmbito da unidade curricular de Segurança e Privacidade de Dados no âmbito de explicar uma ferramenta que permite a troca de mensagens encriptadas entre o servidor e um cliente.

Inicialmente é efetuada uma introdução ao tema, passando pelo processo de implementação onde se explicam os passos e decisões tomadas, convergindo numa conclusão.

Palavras-chave:

Encriptação, Chat, Chaves Públicas, Chaves Privadas, Certificados, TOR

ÍNDICE

ÍNDICE DE FIGURAS

1. Introdução

Este relatório foi realizado no âmbito da unidade curricular de Segurança e Privacidade de Dados, incluída no primeiro ano do mestrado em Engenharia Informática, ramo de Engenharia de Dados, do Instituto Superior de Engenharia do Porto e apresenta os resultados do desenvolvimento de uma aplicação de chat seguro.

O nosso objetivo principal foi criar uma plataforma que permitisse a troca de mensagens de forma segura, mesmo em ambientes considerados inseguros, fazendo uso de técnicas avançadas de criptografia. Assim, procuramos criar uma solução que garantisse a confidencialidade e integridade das comunicações, através de RSA e de chaves assimétricas. Para além disso, foi utilizado o protocolo TOR para garantir o anonimato e segurança. Este relatório descreve os objetivos do projeto, a metodologia adotada e os resultados alcançados, destacando a importância da segurança da informação num mundo cada vez mais conectado.

2. Implementação

Irá ser explicado o processo de implementação através da explicação de *snipets* do código criado para o problema exposto.

Começamos por fazer os *imports* das bibliotecas *python* utilizadas para o projeto.

```
1  import sys
2  from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout, QPushButton, QLabel, QLineEdit, QTextEdit, QMessageBox
3  from PyQt5.QtCore import Qt
4  from PyQt5.QtGui import QIcon
5  import socks
6  import socket
7  import threading
8  import rsa
```

Figura 1 - Imports utilizados para o projeto

Para tornar a aplicação mais *user friendly* foi construído um *frontend* simples através da biblioteca *PyQt5*.

```
class ChatWindow(QWidget):
```

Classe que monta a janela do *frontend*, que contém as seguintes funções:

- __init__(self, parente=None):

```
def __init__(self, parent=None):
    super(ChatWindow, self).__init__(parent)
    self.setWindowTitle("ISEP Chat")
    self.resize(800, 600)

    self.setup_ui()

    self.public_key, self.private_key = rsa.newkeys(1024)
    self.public_partner = None
```

Figura 2- Função __init___

Esta função permite a inicialização do programa e da janela onde poderemos interagir enviando e recebendo mensagens, são definidos parâmetros como o tamanho da janela, o título e ainda são criadas as *public* e *private keys* que foram encriptadas com *rsa*, essenciais para uma comunicação segura entre diferentes *hosts*.

- setup_ui(self):

```python
def setup_ui(self):
    layout = QVBoxLayout()

    # Texto de saída
    self.output_text = QTextEdit()
    self.output_text.setReadOnly(True)
    self.output_text.setStyleSheet("QTextEdit { background-color: #f0f0f0; color: #333; border: 1px solid #ccc; }")
    layout.addWidget(self.output_text)

    # Caixa de entrada de texto
    self.input_text = QLineEdit()
    self.input_text.setFixedHeight(60)
    self.input_text.setPlaceholderText("Digite a mensagem aqui")
    self.input_text.setStyleSheet("QLineEdit { background-color: #fff; color: #333; border: 1px solid #ccc; }")
    self.input_text.returnPressed.connect(self.send_message)  # Conectar a tecla Enter à função send_message
    layout.addWidget(self.input_text)

    # Botão de enviar mensagem
    self.send_button = QPushButton("Enviar")
    self.send_button.setStyleSheet("QPushButton { background-color: #06bd33; color: #fff; border: none; padding: 8px 20px; }"
                                   "QPushButton:hover { background-color: #038222; }")
    self.send_button.clicked.connect(self.send_message)
    layout.addWidget(self.send_button)

    # Botão para iniciar o servidor
    self.host_button = QPushButton("Iniciar Servidor")
    self.host_button.setStyleSheet("QPushButton { background-color: #0465ba; color: #fff; border: none; padding: 8px 20px; }"
                                   "QPushButton:hover { background-color: #014682; }")
    self.host_button.clicked.connect(self.run_server)
    layout.addWidget(self.host_button)

    # Botão para conectar como cliente
    self.connect_button = QPushButton("Conectar como Cliente")
    self.connect_button.setStyleSheet("QPushButton { background-color: #c4a704; color: #fff; border: none; padding: 8px 20px; }"
                                      "QPushButton:hover { background-color: #8a7503; }")
    self.connect_button.clicked.connect(self.run_client)
    layout.addWidget(self.connect_button)

    self.setLayout(layout)
```

Figura 3 – Função setup_ui

Aqui são definidos todos os elementos adjacentes à interface, bem como a definição de todos os parâmetros necessários à comunicação:

- # Texto de saída
  - É aqui que definimos onde vai ser visualizado o nosso chat
- # Caixa de entrada de texto
  - É aqui que definimos onde poderemos escrever a mensagem que pretendemos enviar, bem como o que acontece quando premimos a tecla *Enter*, fazendo a conexão com o servidor enviando o conteúdo da mensagem presente na caixa de texto definida.
- # Botão de enviar mensagem

        - É aqui que definimos o botão que serve de alternativa ao premir a tecla "*Enter*", tem o mesmo comportamento anterior com a conexão ao servidor e envio do conteúdo presente na caixa de texto.
    - # Botão para iniciar o servidor
        - É aqui que definimos o botão de inicialização do nosso servidor, onde o utilizador que vai dar *host* deve clicar.
    - # Botão para conectar como cliente
        - É aqui que definimos o botão de conexão ao servidor hospedado pelo *host*, onde o utilizador que se vai conectar deve clicar.

- configure_tor_connection(self):

```
def configure_tor_connection(self):
    try:
        socks.setdefaultproxy(socks.PROXY_TYPE_SOCKS5, "127.0.0.1", 9050)
        socket.socket = socks.socksocket
        self.output_text.append("Conexão ao Tor estabelecida com sucesso.")
    except Exception as e:
        self.output_text.append("Erro ao conectar ao Tor: " + str(e))
```

Figura 4 – Função configure_tor_connection

- send_message(self):

```
def send_message(self):
    message = self.input_text.text()
    if message:  # Enviar apenas se a mensagem não estiver vazia
        self.input_text.clear()
        self.output_text.append("Eu: " + message)
        self.client.send(rsa.encrypt(message.encode(), self.public_partner))
```

Figura 5 – Função send_message

É aqui que definimos a variável *message* com o conteúdo resultante do *input* da caixa de texto, é feita a partir de uma cláusula *if* uma verificação do conteúdo da mensagem, de forma que apenas seja enviada quando tiver conteúdo. Posteriormente, é feita a limpeza do *input* através da função *clear()* e, então, passa-se o conteúdo presente na variável *message* para o *output* e é feita a encriptação do envio da mensagem que será enviada para o servidor.

- receive_message(self):

```
def receive_messages(self):
    while True:
        try:
            received_message = rsa.decrypt(self.client.recv(4096), self.private_key).decode()
            self.output_text.append("Amigo: " + received_message)
        except Exception as e:
            print("Erro ao receber mensagem:", e)
            break
```

Figura 6 – Função receive_message

Aqui é definida a função que vai tratar a receção da mensagem, com a clausula *try* procede-se a desencriptação da mensagem com a chave privada e lançada a exceção caso tal não tenha sido bem-sucedido, emitindo uma mensagem de erro.

- run_server(self):

```
def run_server(self):
    self.server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.server.bind(("localhost", 9999))
    self.server.listen()

    self.client, _ = self.server.accept()
    self.client.send(self.public_key.save_pkcs1("PEM"))
    self.public_partner = rsa.PublicKey.load_pkcs1(self.client.recv(1024))

    self.configure_tor_connection()

    threading.Thread(target=self.receive_messages).start()
```

Figura 7 – Função run_server

Esta função vai inicializar o servidor do utilizador que vai dar *host*. Este *host* vai ser a partir da sua máquina pessoal no seu *localhost*, que, posteriormente, vai ser possível dar acesso a partir do túnel criado pelo *ngrok*.

É feita a troca de chaves, essencial para a criação da conexão e troca de mensagens e inicializado o protocolo *Tor* e definida uma *thread* que vai tratar de receber as mensagens.

- run_server(self):

```
def run_client(self):
    self.client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.client.connect(("4.tcp.eu.ngrok.io", 19278))  # Alterar para o endereço do servidor desejado

    self.public_partner = rsa.PublicKey.load_pkcs1(self.client.recv(1024))
    self.client.send(self.public_key.save_pkcs1("PEM"))

    self.configure_tor_connection()

    threading.Thread(target=self.receive_messages).start()
```

Figura 8 – Função run_client

À semelhança da função do host, aqui é utilizada uma função que irá inicializar o cliente. O cliente irá se conectar ao endereço criado pelo *ngrok* por parte do *host* e que dar-lhe-á acesso ao *localhost* do servidor hospedado.

É feita a troca de chaves, essencial para a criação da conexão e troca de mensagens e inicializado o protocolo *Tor* e definida uma *thread* que vai tratar de receber as mensagens.

Para convergir todas as funções demonstradas anteriormente, foi criada a função *main* que é a responsável por integrar todas as funcionalidades anteriores e inicializar a aplicação.

```
if __name__ == "__main__":
    app = QApplication(sys.argv)
    # Define o ícone da janela
    app.setWindowIcon(QIcon("isep.png"))
    window = ChatWindow()
    window.show()
    sys.exit(app.exec_())
```

Figura 9 – Funçáo main

Para confirmar se as mensagens realmente estão encriptadas, utilizou-se a aplicação *WireShark*, onde foi possível verificar que o seu conteúdo estava, de facto, encriptado. Na imagem a seguir, é possível ver a interface da aplicação e no canto inferior verificar que as mensagens estão, de facto, encriptadas.

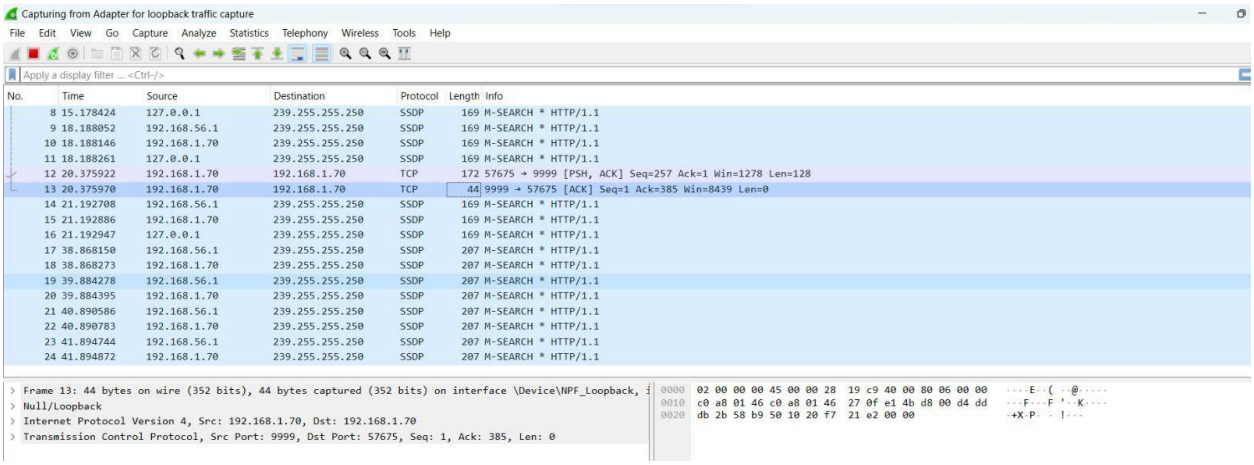

Figura 10 – Interface da aplicação WireShark

3. Conclusão

A criação de um *chat* encriptado permitiu aprofundar os conhecimentos do grupo em relação aos vários termos de encriptação, nomeadamente, chaves públicas e privadas, protocolo *Tor*, certificados, entre outros.

Ao longo deste trabalho foi possível compreender as ameaças à informação e as estratégias utilizadas para as mitigar. A necessidade de proteger a privacidade e confidencialidade dos dados é cada vez mais importante dada a crescente quantidade de dados que o utilizador partilha nas diferentes plataformas.

Percebemos também que a segurança é uma jornada continua e que ferramentas como este *chat* continuarão a evoluir para enfrentar os novos desafios e ameaças que irão aparecer.

O trabalho foi dividido de forma justa e equilibrada de forma que todos os elementos do grupo tivessem impacto no desenvolvimento e que o resultado cumprisse os requisitos pedidos no enunciado.

Referências

MoodleISEP. (n.d.). Moodle.isep.ipp.pt. Retrieved March 24, 2024, from https://moodle.isep.ipp.pt/course/view.php?id=4744

ONION SERVICES | Tor Project | Tor Browser Manual. (n.d.). Tb-Manual.torproject.org. https://tb-manual.torproject.org/onion-services/

ngrok Platform Overview | ngrok documentation. (n.d.). Ngrok.com. https://ngrok.com/docs/

Cruz, J. D. (2023, September 29). *Chaves Públicas e Privadas: Como Funcionam e Por Que São Importantes*. Medium. https://medium.com/@jonathandacruz/chaves-p%C3%BAblicas-e-privadas-como-funcionam-e-por-que-s%C3%A3o-importantes-59a6da146592

Software, C. W. I. (n.d.). *Dúvidas Terra*. Dúvidas Terra. Retrieved March 24, 2024, from https://duvidas.terra.com.br/duvidas/570/o-que-e-criptografia-de-chaves-publica-e-privada

GOLDSCHLAG, David; REED, Michael; SYVERSON, Paul. Onion Routing for Anonymous and Private Internet Connections. 1999 from: https://www.onion-router.net/Publications/CACM-1999.pdf

Sending Files Over the Internet with Python's http.server and Ngrok: A File Transfer Hack for Restricted Environments from:

https://medium.com/@jonathan.hoffman91/sending-files-over-the-internet-with-pythons-http-server-14d5446f29b4

Rede TOR e Onion Routing from: https://www.dio.me/articles/rede-tor-e-union-routing