Comunicação Segura entre Duas Threads

Felipe Schreiber Fernandes DRE: 116206990

November 2020

1 Introdução

O objetivo principal desse trabalho foi gerar uma comunicação segura entre dois sockets. Tem-se, portanto, duas threads: Uma que aguarda uma conexão ser feita (servidor) e outra que inicia comunicação (cliente).Para criar um canal seguro utilizou-se a criptografia RSA, a partir da biblioteca openSSL, gerando um par de chaves para o servidor e outro para o cliente. A chave pública será utilisada para descriptografar e a chave privada para criptografar. Essa criptografia foi escolhida por se tratar de um criptografia assimétrica, e portanto mais resiliente a ataques do tipo man in the middle. O relatório está organizado da seguinte forma: Na seção 2 explica-se quais mensagens o cliente envia e recebe, na seção 3 explica-se quais mensagens o servidor envia e recebe e por fim conclui-se na seção 4.

2 Cliente

Primeiramente ele inicia a conexão com o servidor e espera até que ela seja aceita. Em seguida lê de dois arquivos ".pem" suas respectivas chaves pública e privada. Então a seguinte sequência de mensagens ocorre:

1. Envia para o servidor uma mensagem no formato 0—chavePrivadaCliente—, que contêm sua chave privada.
2. Recebe uma mensagem no formato 0—chavePrivadaServidor— e guarda o valor dessa chave numa variável.
3. Escreve uma mensagem no formato 1—ola do cliente— e a criptografa com a chave do servidor recebida no passo anterior. Envia a mensagem criptografada.
4. Envia o tamanho da mensagem criptografada.
5. Recebe mensagem no formato 2—ola do servidor—, embora criptografada com sua chave privada.
6. Recebe o tamanho da mensagem anterior criptografada.
7. Uma vez obtida a mensagem e o seu tamanho, ela é descriptografada usando a chave pública do cliente. Então o cliente consegue ler a mensagem em texto plano.
8. Escreve uma mensagem no formato 3—pare— e a criptografa usando a chave privada do servidor. Envia a mensagem criptografada.
9. Envia o tamanho da mensagem criptografada.
10. Encerra a conexão e a thread.

3 Servidor

Primeiramente ele cria um socket, o coloca para escutar a porta 8080 e espera até que um novo socket tente se comunicar e o aceita. Em seguida lê de dois arquivos ".pem" suas respectivas chaves pública e privada. Então a seguinte sequência de mensagens ocorre:

1. Recebe uma mensagem no formato 0—chavePrivadaCliente— e guarda o valor dessa chave numa variável.
2. Envia para o cliente uma mensagem no formato 0—chavePrivadaServidor—, que contêm sua chave privada.
3. Recebe mensagem no formato 1—ola do cliente—, embora criptografada com sua chave privada.
4. Recebe o tamanho da mensagem anterior criptografada.
5. Uma vez obtida a mensagem e o seu tamanho, ela é descriptografada usando a chave pública do servidor. Então o servidor consegue ler a mensagem em texto plano.
6. Escreve uma mensagem no formato 2—ola do servidor— e a criptografa com a chave do cliente recebida. Envia a mensagem criptografada.
7. Envia o tamanho da mensagem criptografada.
8. Recebe uma mensagem no formato 3—pare—, embora criptografada com sua chave privada.
9. Recebe o tamanho da mensagem anterior criptografada.
10. Uma vez obtida a mensagem e o seu tamanho, ela é descriptografada usando a chave pública do servidor. Então o servidor consegue ler a mensagem em texto plano.
11. Se ela for "pare" encerra a conexão e a thread. Dá um release no lock MAIN, possibilitando que o programa principal execute e finalize.

4 Conclusão

Conseguiu-se criar um meio de comunicação seguro onde os endpoints não tem acesso a chave utilizada para descriptografar um do outro. Assim tem-se um canal seguro e a prova de ataques Man In The Middle, uma vez que ele só terá acesso as chaves usadas para criptografar mas não as do processo inverso. Supõe-se que antes de executar os arquivos "ServerPrivate.pem","ServerPublic.pem","ClientPrivate.pem","ClientPublic.pem" estejam presentes. A compilação pode ser feita com gcc -o < *nome_executavel* > chat.c MyEncrypt.c -lcrypto -lpthread.

5 Referências

http://www.firmcodes.com/asymmetric/