UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

PEDRO HENRIQUE DA SILVA PEREIRA

TROCA DE MENSAGENS COM CONFIDENCIALIDADE, AUTENTICIDADE E INTEGRIDADE

Atividade 1 – Segurança e Auditoria de Sistemas

Cornélio Procópio
2020

1 Bibliotecas usadas e importações feitas para o código

Para a construção dos códigos foram utilizadas as bibliotecas e importações: socket, hashlib, Crypto, base64, rsa, simplecrypt, Crypto.PublicKey, Crypto.Util.randpool, Crypto.Cipher, binascii, Crypto.Hash, Crypto.Signature, encrypt, decrypt , PKCS1_OAEP, hexlify, b64encode, b64decode, SHA256 e PKCS1_v1_5.

2 Construção dos codigos

Os dois códigos foram construidos baseados na trasmissão de dados via socktes, mas a primeira parte da implementação dos códigos são variáveis de escopo global, onde foram geradas as chaves assimétricas e transformadas em strings para serem armazenadas em arquivos. Após isso foram implementadas duas estruturas uma para assinaturas com chaves privada e outra para verificar a autenticidade, ambas estruturas foram retiradas do link: https://www.it-swarm.dev/pt/python/como-criptografar-com-uma-chave-privada-rsa-em-python/806449929/ e por ultimo foi atribuida uma chava simétrica, o algorimito que ultiliza esta chave é baseado na biblioteca simplecrypt.

2.1 Server.py

Dentro do código Server.py foi atribuido o ip e a porta na qual o código iria ficar escutando até receber uma conexão, que no caso o ip: localhost e a porta: 5535. Após atribuir esses valores é gerando uma variável que recebe a quantidade máxima de dados que podem ser transmitidos, que no caso é 2048 bits ou 256 caracteres. Cria-se um socket do tipo TCP e habilita o uso do socket no ip e na porta definidas antes. Atribui um numéro máximo de clientes que podem se conectar que é 5. Após todas essas definições ele solicita um user name e

começa escutar na rede por um cliente pedindo conexão com ele, quando um cliente conecta o server recebe a chave publica do cliente, faz a leitura através do comando RSA.import da biblioteca RSA que pega a string que foi enviada pelo cliente e refaz a chave pública do cliente, depois o server envia a sua própria chave pública para o cliente.

Após espera-se que o cliente conectado envie alguma mensagem, se isso ocorre o server pega a chave simétrica envia pelo cliente decriptografa a mensagem e pega o hash que também foi enviado pelo cliente, com a estrutura verify definida no escopo global é certificada a autenticidade da chave enviada.

Se a chave for autentica o servidor envia sua chave e o hash da mesma e fica aguardando o envio de uma mensagem do cliente, quando recebe, decriptografa com a chave simetrica do cliente e armazena a mensagem, pega o hash da mensagem enviada usa a estrutura verify para certificar que a mensagem é autentica, se for autentica o server verifica a mensagem se for exit a conexão é encerrada se não o server imprime a mensagem no terminal e recebe um texto para ser enviado ao cliente.

Se a chave não for autentica o server imprime na tela “A mensagem de fulano chegou com problemas” e encerra a conexão.

2.2 Client.py

A grande diferença do código client.py para o server.py é que o client sempre inicia a comunicação, onde ele solicita a conexão, envia a chave publica primeiro e envia a mensagem e sua chave simétrica primeiro.

Mas o resto tanto no client quanto no server existe uma estrutura para enviar as chaves simétricas para garantir a confidenciálidade, uma

estrutura que garante a integridade das trocas de mensagem, e a assinatura que garante a autenticidade na troca de mensagem.

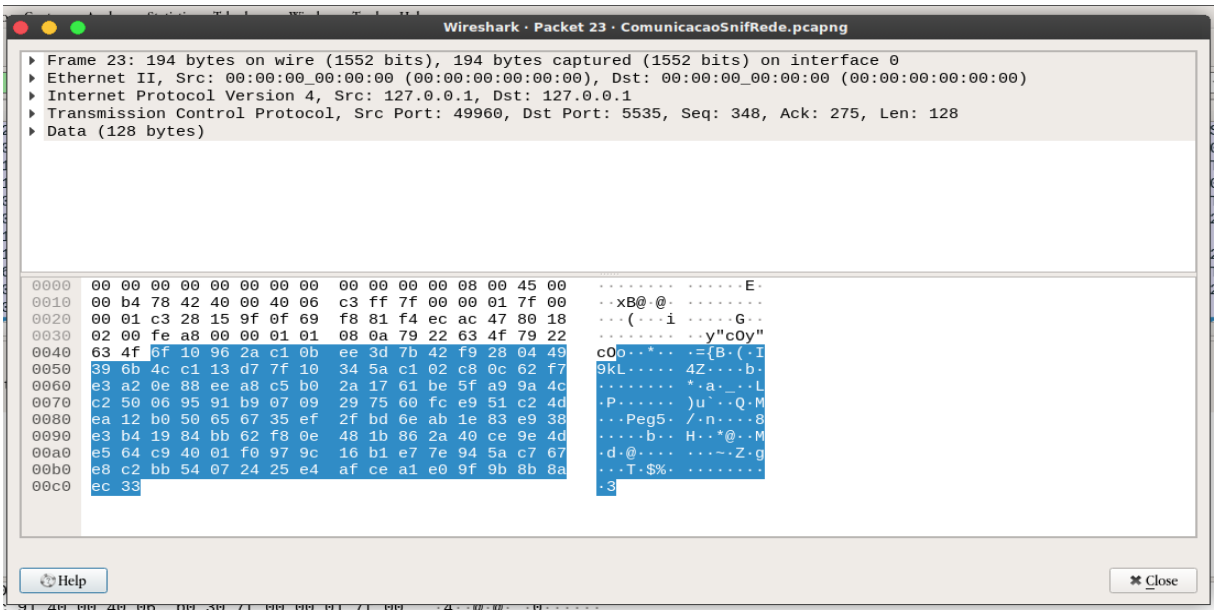

Figure 1: Leitura de um dos pacotes capturados

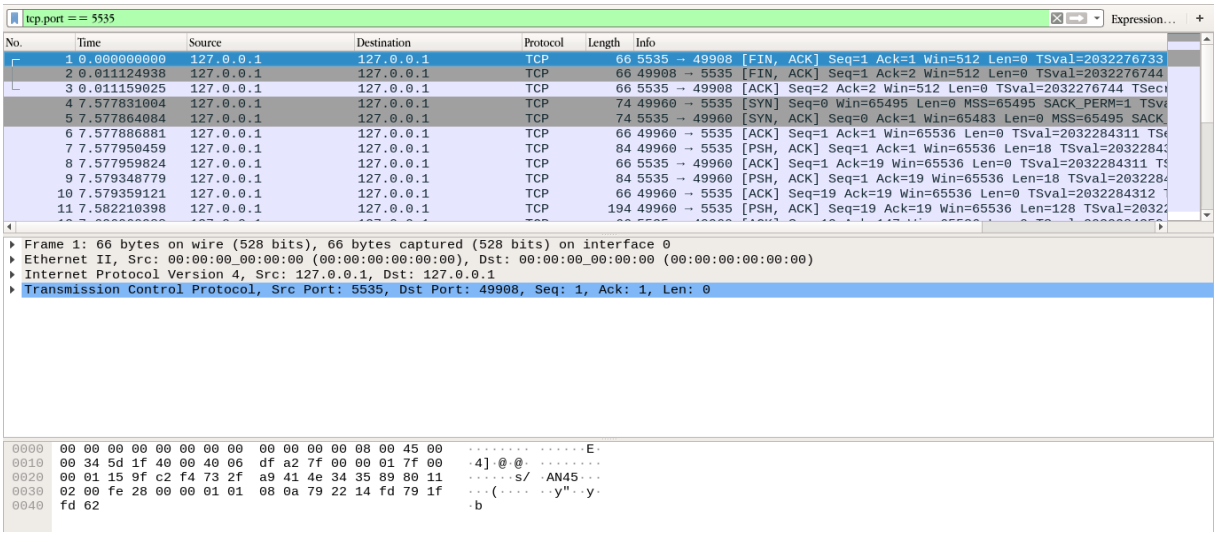

Figure 2: Pacotes capturados por Sniffer