UNIVERSIDADE FEDERAL DE SANTA CATARINA
Centro de Ciências, Tecnologias e Saúde - CTS

Projeto Final – parte II

Curso: **Graduação em Engenharia de Computação**
Disciplina: **DEC7557 - Redes de Computadores**
Assunto: **Programação de rede usando sockets UDP e uso de técnicas de criptografia assimétrica para assinatura digital em tokens JWT**
Data e forma de entregas: parciais, conforme especificado nos requisitos
Data de Entrega: 29 junho 2023 (quinta-feira) Entrega: conforme especificado ao final do documento
Revisão: 17 jun 2023 (v.1)

		Grupos	Linguagem
1	Javali	Aicha, Gabriel Lencina, Ítalo	python
2	Pinhão	Alexandre, Eduardo, Maurício	C++
3	Bissau	Michelly, Pedro, Jesiel	
4	Noname	Carla, Gabriel Feltes, Matheus	
5	Soquetinho	Júlio, Gregory, Capistrano	

Programação de aplicações em rede usando sockets

Objetivo: compreender os princípios das aplicações de rede na prática usando bibliotecas sockets fornecidas pelas diversas linguagens de programação. Usar recursos de criptografia de chave pública para criar e verificar tokens JWT.

Atividade: usando a linguagem definida pelo grupo, criar aplicações que façam chamadas de rede usando sockets como interface básica. As aplicações devem preferencialmente serem no modo CLI e deverão fazer uso de algumas rotinas para:
- Ler parâmetros de linha de comando;
- Criar e gerenciar sockets;
- Escrever rotinas para estabelecer comunicação em rede usando sockets;
- Usar rotinas para escrever determinados resultados em arquivos;
- Usar recursos de criptografia assimétrica para assinar informação que deve compor uma estrutura de JWT;
- Incluir mecanismos básicos de transmissão com confiança através de sockets UDP.

Parte II – criação de tokens JWT assinados com criptografia de chave pública

Objetivo: enviar dados do grupo a um servidor remoto usando UDP e controlar a sequência e o recebimento das mensagens pelo servidor. Para isso, será necessário criar e assinar um token JWT (JWS) e enviar o mesmo a um serviço remoto na porta 34567/UDP com dados de payload seguindo um formato e requisitos próprios. A resposta em JWS deve ser verificada quanto ao conteúdo e assinatura.

Requisitos:
a) Deve ser escrito na linguagem de sua escolha e usando as chamadas de sockets oferecidos pelas bibliotecas inclusas na linguagem. A criação e verificação de tokens JWT pode ser através de bibliotecas disponíveis na linguagem, como PyJWT em Python.
Obs.: os tokens JWT podem ser criados e verificados de forma offline através de comandos e sítios específicos, como o programa openssl.

b) Pode ser escrito para qualquer sistema operacional.

c) Criar um par de chaves usando algoritmo de chave pública..

Criação de um par de chaves usando algoritmo de criptografia pública

Há diversos algoritmos de chave pública, sendo os três mais proeminentes: RSA, ECDSA e EdDSA Ed25519. Além do algoritmo devem ser especificados os parâmetros do mesmo, como tamanho de chave. Uma questão que se apresenta é o formato das chaves, que, conforme o programa usado e parâmetro, podem vários. Por exemplo, um mesmo formato PEM (https://en.wikipedia.org/wiki/Privacy-enhanced_Electronic_Mail) pode ser usado de forma diferente por chaves criadas pelo programa GPG e pelo SSH, mesmo que tenha sido usado o mesmo algoritmo. Logo, para este projeto, os requisitos de chaves serão:

- Algoritmo: RSA
- Tamanho de chave: 1024 bits
- Formato arquivo: PEM

A geração do par de chaves pode ser por um destes meios (recomendado o uso da letra a)

a) Usando o sítio https://cyberchef.org: seção **Chave Pública → Generate RSA Key Pair**
b) Usando o OpenSSH através do aplicativo ssh-keygen para gerar um par de chaves do tipo SSH[1].

O que fazer com as chaves:
a) Chave privada: deve ser mantida com alto nível de proteção, apesar de que é a mesma que todos os componentes do grupo usarão para gerar os tokens JWT.

b) Chave pública: esta deve ser enviada por e-mail para gerson.camillo@ufsc.br com o nome do grupo no subject. Colar o conteúdo da chave no corpo da mensagem e usar envio sem formatação. Caso queiram enviar como anexo, devem ser seguidos os cuidados: texto; formato UTF-8; sem marcador de BOM.

Referências:
https://en.wikipedia.org/wiki/Public-key_cryptography
https://cryptobook.nakov.com/asymmetric-key-ciphers/the-rsa-cryptosystem-concepts
https://cryptobook.nakov.com/asymmetric-key-ciphers/elliptic-curve-cryptography-ecc
https://cryptobook.nakov.com/digital-signatures/ecdsa-sign-verify-messages
https://cryptobook.nakov.com/digital-signatures/eddsa-and-ed25519

d) Especificação do protocolo e do payload:

Neste quadro será especificado o protocolo de envio/recebimento de mensagens e os respectivos formatos
Requisição: mensagem UDP para endereço e porta informados em momento posterior[2] contendo o seguinte conteúdo: token JWT, especificamente um token JWS cujo payload deve ser o formato a seguir juntamente com os respectivos valores permitidos de campos. Esse token deve ser assinado pela chave privada do grupo. **Formato da mensagem**: deve ser usado formato JWT conforme documentos em anexo. Algoritmo de Assinatura: **RS256 (RSASSA-PKCS1-v1_5 using SHA-256)**[3] O payload deve ser formatado JSON contendo necessariamente os seguintes campos (respeitando a sintaxe): {

1 Um arquivo em anexo contém os passos básicos para usar comandos e chamadas para chaves e tokens JWT.
2 Especificação do endpoint remoto atualmente previsto: **gersoncamillo.seg.br:34567**
3 Referência para os algoritmos: https://www.rfc-editor.org/rfc/rfc7518.html

"group": "JAVALI" | "PINHAO" | "BISSAU" | "NONAME" | "SOQUETINHO",
"seq_number": 1|2|3|4,
"seq_max": 2-4,
"matricula": xxxxxxx,
}
Significado e semântica dos campos:
"group": nome do grupo em letras maiúsculas
"seq_number": número sequencial iniciando em 1 e que indicará o número de matrícula que está sendo enviado ao servidor
"seq_max": número indicando os componentes do grupo
"matricula": número de matrícula de cada aluno
}

Tratamento da requisição e resposta: o servidor receberá o token JWS e verificará a assinatura. Caso seja possível extrair os dados de todos os campos, será checada a sequência de recebimento contida no campo seq_max, de forma a prover ordenamento das mensagens. Então, erros, dentre os quais:
- assinatura não válida;
- formato não possível decodificar;
- campos ausentes ou com valores ausentes; ou
- número de sequência fora da ordem.

Farão com que o servidor não responda com qualquer mensagem. Se tudo correto, uma mensagem resposta JWT assinada com algoritmo HS256 será enviada de volta com as seguintes propriedades:
- Payload:
{
"id_request": f12010c0ec208fae7b0031714839f7d639d921c6a6e73ad97a1aab4cc21ba43e,
"next_number": 2,
"otp_number": 3205,
"otp_timestamp": 1687046022,
}
Significado e semântica dos campos:
"id_request": contém o hashing usando SHA256 do payload enviado na requisição
"next_number": número do próximo aluno que o sistema espera receber
"otp_number": número aleatório gerado para cada resposta
"otp_timestamp": timestamp em epoch segundos inteiros do momento da geração da resposta. Pode-se usar o seguinte sítio para converter num formato de data-hora compreensível para humanos: https://www.epochconverter.com

Observação importante: dados recebidos e reconhecidos estarão armazenados em memória da aplicação. Caso não chegue a informação do último aluno e a aplicação terminar por qualquer motivo, o protocolo de funcionamento determina que seja reiniciado o envio das mensagens. Ou seja, não são armazenadas informações parciais de mensagens em memória permanente.

Algoritmo de assinatura da resposta: será usado o algoritmo HS256 (HMAC using SHA-256) que é um protocolo de Hashing mas parametrizado por uma chave secreta compartilhada entre as entidades.
Chave secreta: dec7557-socket-udp-with-jwt (deve ser usado o formato bytes puros em Python)

d) Receber token de resposta JWT e fazer o seguinte:
- Verificar a assinatura
- Salvar a resposta bruta em arquivo junto com o resultado da verificação da assinatura (OK ou NOT_OK)
- Enviar a mensagem do próximo aluno

Entregas

a) Arquivo de log com as mensagens recebidas e a respectiva checagem de assinatura

b) Código fonte comentado

c) Questão: quais as considerações do grupo quanto ao seguinte problema: se for recebida uma mensagem e esta tiver uma assinatura inválida, como o protocolo deveria tratar dessa questão. Faça considerações envolvendo confiabilidade e segurança.

d) Questão: como o servidor deve tratar o caso de um cliente enviar repetidamente a mesma mensagem de requisição. Deve responder criando a respectiva mensagem de resposta ou limitar o número de respostas. Neste último caso como ficaria a questão da confiabilidade deste protocolo.

Projeto Final – parte II

Parte I – scanner de rede

Atividade de escrita de um programa que execute serviço de varredura de portas para descobrir qual porta UDP o serviço de recebimento de tokens JWT estará escutando. Para fins de comparação entre comportamento de tipos de transporte, esta atividade envolverá um scanner de rede para sockets TCP e UDP.

Requisitos:

a) Deve ser escrito na linguagem de sua escolha e usando as chamadas de sockets oferecidos pelas bibliotecas inclusas na linguagem.

b) Pode ser escrito para qualquer sistema operacional.

c) O programa deve ser executado em uma rede local.

d) O objetivo é descobrir portas abertas (serviços TCP e UDP escutando por conexões) em uma máquina na mesma rede.

e) O programa deve ser, preferencialmente, executado por linha de comando, com os seguintes parâmetros:

`programa-de-varredura tcp|udp IP_do_host 80,443,2222`

`tcp|udp` : qual tipo de transporte (TCP ou UDP)

`IP_do_host` : somente endereço IPv4 para designar o host destino

`80,443,2222` : três portas para executar varredura separadas por vírgula

Obs.: valores serão definidos no teste prático.

f) Os resultados dos testes TCP e UDP devem ser salvos em arquivo, conforme apresentado no quadro.

g) Nos testes UDP, as mensagens de resposta devem ser armazenadas no mesmo arquivo do teste de portas ou em um outro, também conforme quadro a seguir.

h) Quando da realização dos respectivos testes (TCP e UDP), o endereço IPv4 público deve ser informado na documentação de explicação dos resultados. Como obter: acessando algum sítio externo que forneça essa informação, como https://ipinfo.io

Endereços e portas para testes:

TCP: 159.223.102.104 ou 2604:a880:400:d0::1745:d001 ou noctilucentis.com.br
Portas 80,2000,4000

UDP: 43.204.246.63 ou gersoncamillo.seg.br
Portas: 2000,23456,34567

Obs.: por vezes vou testar a conectividade, mas caso alguém encontre algum problema, peço que informem. O ideal seria controlar a execução via sistemas e/ou protocolos de gerenciamento de rede.

Resultados e avaliação:

a) Apresentar o código fonte comentado
b) O programa deve ser executado até o fim, sem erros
c) O programa deve detectar todas as portas UDP e TCP abertas no host destino e escrever o resultado num arquivo texto seguindo este formato:

```
tcp/80    Open
tcp/111   Closed
tcp/443   Closed
```

~~53/udp   Open|Filtered~~
~~2000/udp   Open~~
~~200/udp   Closed~~

```
53/udp   Closed|Filtered
2000/udp   Open
200/udp   Closed|Filtered
```

Obs.: considerando a proposta de uso de sockets de nível de usuário, só é possível identificar duas situações quanto a portas UDP: **aberta**, quando o servidor entender o mesmo protocolo da requisição; f**echada ou filtrada,** no caso de ausência de respostas (porta fechada , sem serviço ou porta filtrada por firewall).

Para portas UDP, a identificação de porta aberta somente será positiva se o payload da requisição contiver dados/formato que o servidor UDP "entenda" (protocolo). Neste teste, o protocolo UDP será o seguinte:
Requisição: formato: um campo de bytes de tamanho limitado pelo conteúdo que deve ser o nome do grupo. Exemplo de como formatar os dados:

```
>>> bytes('soquetinho'.encode('utf-8'))
```

Resposta: dados em bytes contendo campos separados por ponto. Então, em recebendo a mensagem acima, porta aberta e servidor UDP DEC7557 sendo executado. Exemplo de uma resposta:
ACK.JAVALI.18594.2023-06-03,19:59:18
Campo 1: ACK (confirmação)
Campo 2: JAVALI (nome do grupo em case maíscula)
Campo 3: 18594 (número inteiro aleatório gerado para cada resposta)
Campo 4: 2023-06-03,19:59:18 (timestamp relativo à data-hora de envio da resposta)

Obs.:

i) Será provido posteriormente um servidor em nuvem para testes.
ii) Testes podem ser feitos criando uma conexão UDP usando o programa Netcat disponível em sistemas Linux (há versões para Windows):
`$ nc -u -l 3000` Cria um servidor UDP na porta 3000
iii) Checar se o payload da resposta se encontra no formato e contendo dados relativos à requisição, ou seja, se o nome do grupo está correto.

Obs.: o Nmap provê uma versão chamada ncat (Ncat).

d) **Escrever todos os payloads UDP em arquivo** conforme exemplo a seguir. Não é necessário para TCP, apenas se portas abertas/fechadas.
`ACK.SOQUETINHO.18757.2023-06-03,19:59:20`

e) Comentar os resultados, provendo explicações/justificativas para os resultados.

A verificação de portas abertas (scaneamento de portas) procura identificar serviços/portas que sistemas remotos tenham abertos para prover algum serviço de rede. Algumas portas já possuem serviços associados, como 80/TCP para HTTP, de forma que o total de 65536 portas está dividido em grandes faixas (uma relação completa pode ser encontrado em https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers e https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml)

a) **Portas bem conhecidas (well-known)**: números de porta na faixa de 0 a 1023 e que são conhecidas como portas de sistemas. A atribuição é para serviços de rede bem conhecidos e fica a cargo da IANA. No Unix/Linux, serviços nestas portas precisam que programas executem com direitos de administrador (root).

b) **Portas registradas**: faixa que compreende portas entre 1024 e 49151. Atribuídos pela IANA a pedido de entidades com o fim de atender determinados serviços. Normalmente o processo de bind nestas portas não requer direitos administrativos. Por exemplo, para o servidor MSSQL (banco de dados da Microsoft): 1433/tcp e 1433/ udp (ms-sql-s Microsoft-SQL-Server).

c) **Portas dinâmicas, privadas ou efêmeras (usadas por pouco tempo)**: números entre 49152 e 65535 e que são para propósitos de alocação dinâmica, como portas de origem

para conexões remotas (não são registradas pela IANA).

A identificação de portas TCP e UDP usam diferentes mecanismos, pois os algoritmos de funcionamento destes protocolos são diferentes. Em termos gerais, serviços TCP podem ser identificados através de tentativas de abertura de conexões (SYN, SYN-ACK, ACK), mas UDP funcionam de forma diferente. Um protocolo que usa UDP somente responde mensagens se o conteúdo (payload) do UDP contiver dados em formato que o respectivo protocolo entenda. Por exemplo, um servidor DNS abre a seguinte porta: 53/UDP. O programa servidor somente responderá caso a requisição contenha uma mensagem no formato do DNS, caso contrário, não haverá nenhuma resposta.

O programa Nmap (“Network Mapper”) é uma ferramenta de código aberto para diversos tipos de verificações de rede e de segurança. Seguem links com explicações detalhadas sobre os tipos de segmentos envolvidos nos transportes UDP e TCP:
Para testar portas TCP: https://nmap.org/book/scan-methods-connect-scan.html
Para testar portas UDP: https://nmap.org/book/scan-methods-udp-scan.html