CST em Análise e Desenvolvimento de Sistemas — Prof. Emerson Ribeiro de Mello
SEG786203 – Segurança da Informação — mello@ifsc.edu.br

Laboratório 3: certificados digitais

02/10/2025

Nesse laboratório serão demonstrados os passos para subir um servidor *web* Nginx e configurado com certificados digitais autoassinados para HTTPS. No segundo exemplo será feito uso de certificados digitais assinados por uma Autoridade Certificadora (AC). No terceiro exemplo é demonstrado como trabalhar com certificados digitais em projetos Java, usando o Java Keytool e *keystores*.

1 Criando certificados digitais autoassinados

Os passos a seguir mostram como gerar um certificado digital autoassinado e configurar o servidor *web* Nginx para usar HTTPS. Faremos uso de um contêiner Docker para subir o servidor *web* Nginx. Iremos fazer o mapeamento das portas HTTP (80) e HTTPS (443) do contêiner para as portas 8080 e 8443 do hospedeiro, respectivamente.

1. Criação da chave privada do servidor

```
# Criação de diretórios para armazenar os certificados e chaves do servidor
mkdir -p ~/primeiro/ssl/{certs,private} && cd ~/primeiro

# Criação da chave privada do servidor
openssl genrsa -out ssl/private/server.key 2048
```

2. Criação do certificado autoassinado do servidor

```
# Será necessário informar o FQDN (Fully Qualified Domain Name) do servidor no campo CN.
# O campo CN é o nome comum do servidor, que deve ser o mesmo nome usado na URL.
# Exemplo: servidor.com.br

# Neste laboratório o FQDN será localhost

openssl req -x509 -new -key ssl/private/server.key -out ssl/certs/server.crt \
-days 365 -subj "/C=BR/ST=SC/L=Sao Jose/O=IFSC/OU=SEG/CN=localhost"

# Exibindo o conteúdo do certificado
openssl x509 -in ssl/certs/server.crt -text -noout | more
```

1.1 Uso de certificados digitais em um servidor web Nginx

Crie a seguinte estrutura de diretórios e arquivos no seu diretório de trabalho.

```
primeiro
|-- Dockerfile
|-- html
|   `-- index.html
|-- nginx.conf
`-- ssl
    |-- certs
    `-- private
```

O arquivo `Dockerfile` (veja Listagem 1) contém as instruções para criar a imagem do Nginx. O arquivo `nginx.conf` (veja Listagem 2) contém a configuração do servidor Nginx com suporte a HTTP e HTTPS. O arquivo `index.html` (coloque qualquer conteúdo nesse arquivo) é a página inicial que será servida. O diretório `ssl` contém o certificado digitai e a chave privada do servidor.

Listagem 1: Conteúdo do arquivo `Dockerfile`

```
FROM nginx:alpine

COPY ssl /etc/nginx/
COPY nginx.conf /etc/nginx/conf.d/default.conf
```

Listagem 2: Configuração do Nginx

```
server {
    listen 80 default_server;
    listen [::]:80 default_server;
    listen 443 ssl default_server;
    listen [::]:443 ssl default_server;

    server_name localhost;

    ssl_certificate      /etc/nginx/certs/server.crt;
    ssl_certificate_key  /etc/nginx/private/server.key;

    location / {
        root /usr/share/nginx/html;
        index  index.html index.htm;
    }
}
```

Agora é necessário criar a imagem do Nginx com a configuração acima e executar o contêiner.

```
# Criar a imagem do Nginx
docker build -t nginx-ssl .

# Executar o contêiner do Nginx fazendo o mapeamento das portas 80 e 443 do contêiner
docker run -it -p 8080:80 -p 8443:443 -v $(pwd)/html:/usr/share/nginx/html --rm nginx-ssl
```

Para acessar o servidor Nginx, via HTTP, abra o navegador e acesse http://localhost:8080 ou https://localhost:8443, para acessar o servidor via HTTPS. Acessando via HTTPS o navegador irá exibir um aviso de que a conexão não é segura, pois o certificado é autoassinado.

2 Certificados digitais assinados por uma Autoridade Certificadora

Uma vantagem de usar certificados assinados por uma AC é que, diferente dos certificados autoassinados, o navegador não exibirá um aviso de segurança ao acessar o servidor via HTTPS, se a AC estiver na lista de autoridades confiáveis do navegador.

1. Criação da chave privada da Autoridade Certificadora (AC)

```
# Criação de diretórios para armazenar os certificados e chaves da AC
mkdir -p ~/ca && cd ~/ca

# Criação da chave privada da Autoridade Certificadora (AC).
# Forneça uma senha com pelo menos 4 caracteres
openssl genrsa -aes256 -out ca.key 4096
```

2. Criação do certificado autoassinado da Autoridade Certificadora (AC)

```
# Criação do certificado autoassinado da Autoridade Certificadora (AC)
openssl req -new -x509 -days 365 -key ca.key -out ca.crt \
-subj "/C=BR/ST=SC/L=Sao Jose/O=IFSC SJE/CN=Minha-CA-ADS-SEG"
```

3. Vá para o diretório `primeiro` do exercício anterior. Iremos criar um pedido de certificado (CSR, *Certificate Signing Request*) do servidor para ser assinado pela AC.

```
cd ~/primeiro

# É necessário informar o FQDN (Fully Qualified Domain Name) do servidor no campo CN
# Exemplo: servidor.com.br
# Neste laboratório o FQDN será localhost
openssl req -new -key ssl/private/server.key -out server.csr \
-subj "/C=BR/ST=SC/L=Sao Jose/O=ADS/OU=SEG/CN=localhost"
```

4. Assinatura do CSR com a chave privada da AC e criação do certificado do servidor

   Aqui é onde a AC assina o pedido de certificado (CSR) do servidor com a chave privada da AC. O certificado do servidor é criado e assinado pela AC, sendo este válido por 365 dias.

```
# Assinatura do CSR com a chave privada da AC
openssl x509 -req -extfile <(printf "subjectAltName=DNS:localhost") -days 365 \
-in server.csr -CA ~/ca/ca.crt -CAkey ~/ca/ca.key -CAcreateserial -out ssl/certs/server.crt

# Exibir o conteúdo do certificado.
# Atente-se para o campo Issuer, que deve ser o mesmo do certificado da AC
openssl x509 -in ssl/certs/server.crt -text | more
```

5. Para verificar se o certificado do servidor foi assinado pela AC. O comando deve retornar `OK` se tudo estiver correto.

```
openssl verify -CAfile ~/ca/ca.crt ssl/certs/server.crt
```

6. Reconstrua a imagem do Nginx e execute o contêiner novamente, conforme os passos da seção anterior.

```
# Criar a imagem do Nginx
docker build -t nginx-ssl .
# Executar o contêiner do Nginx fazendo o mapeamento das portas 80 e 443 do contêiner
docker run -it -p 8080:80 -p 8443:443 -v $(pwd)/html:/usr/share/nginx/html --rm nginx-ssl
```

7. Importe o certificado da AC (ca.crt) no navegador.

8. Acesse o servidor Nginx via HTTPS em https://localhost:8443. O navegador não deve exibir qualquer aviso de segurança.

3 Java Keytool e keystores

O Java Keytool é uma ferramenta, fornecida com o JDK, para gerenciar chaves e certificados digitais em um *keystore*. O *keystore* é um repositório de chaves e certificados digitais que pode ser usado em aplicações Java. Servidores de aplicação Java, como Apache TomCat, Jetty, WildFly, entre outros, usam *keystores* para armazenar chaves privadas e certificados digitais para uso em conexões seguras (HTTPS).

3.1 Criar um *keystore* com o Java Keytool

O comando a seguir cria um *keystore* com uma chave privada e um certificado autoassinado que são armazenados no *keystore* com o nome `servidor.jks`.

Listagem 3: Criar um *keystore* com uma chave privada e um certificado autoassinado

```
keytool -genkey -alias servidor -dname "CN=servidor, OU=SEG, O=IFSC, L=Sao Jose, ST=SC, C=BR" \
-keyalg RSA -keysize 2048 -keystore servidor.jks -storepass minhasenha -keypass minhasenha
```

- `-genkey`: gera um par de chaves pública/privada;
- `-alias servidor`: nome do alias para a chave privada;
- `-keyalg RSA`: algoritmo de criptografia RSA;
- `-keysize 2048`: tamanho da chave RSA de 2048 bits;
- `-keystore servidor.jks`: nome do arquivo de *keystore*;
- `-storepass minhasenha`: senha do *keystore*;
- `-keypass minhasenha`: senha da chave privada;
- `-dname`: campos `(CN=servidor, OU=SEG, O=IFSC, L=Sao Jose, ST=SC, C=BR)`.

3.2 Exibir o conteúdo do *keystore*

```
# Exibir o conteúdo do \textit{keystore}
keytool -list -keystore servidor.jks -storepass minhasenha

# Exibir o conteúdo do certificado do  alias servidor
keytool -list -v -alias servidor -keystore servidor.jks -storepass minhasenha
```

3.3 Exportar o certificado do *keystore*

É possível exportar o certificado do *keystore* para um arquivo `.cer` que pode ser importado em outros *keystores* ou navegadores.

```
keytool -export -alias servidor -file servidor.crt -keystore servidor.jks -storepass minhasenha
```

3.4 Importar um certificado em um *keystore*

```
# Importar o certificado de uma Autoridade Certificadora (CA) em um \textit{keystore}
keytool -importcert -alias ca-raiz -file ca-cert.pem -keystore servidor.jks -storepass minhasenha

# Importar o certificado do servidor em um \textit{keystore}
keytool -importcert -alias servidor -file servercert.crt -keystore servidor.jks -storepass
    minhasenha
```

3.5 Remover um certificado de um *keystore*

```
# Remover o certificado do servidor do \textit{keystore}
keytool -delete -alias servidor -keystore servidor.jks -storepass minhasenha
```

4 Configurar o keytore em um projeto Java com Spring Boot

Nesta seção iremos criar um projeto Spring Boot com a biblioteca Spring Web e usar o *keystore* criado na seção anterior para configurar o servidor *web* com HTTPS. O projeto será criado com o Spring Initializr (https://start.spring.io) e será executado em um servidor embutido (Tomcat) na porta 8443.

1. Acesse o site https://start.spring.io e adicione a dependência `Spring Web`.
2. Baixe o ZIP com o projeto e descompacte o arquivo.
3. Adicione o arquivo `servidor.jks` (criado com o comando apresentado na Listagem 3) no diretório `src/main/resources` do projeto.
4. Crie o arquivo `src/main/resources/application.properties` e deixe com o seguinte conteúdo (veja Listagem 4):

Listagem 4: Configuração do *keystore* no arquivo `application.properties`

```
server.port=8443
server.ssl.key-store=classpath:servidor.jks
server.ssl.key-store-password=minhasenha
server.ssl.key-store-type=JKS
server.ssl.key-alias=servidor
server.ssl.key-password=minhasenha
```

5. Por fim, crie um controlador para testar a aplicação (veja Listagem 5). Para executar o projeto, execute o comando `gradle bootRun` e acesse a aplicação em https://localhost:8443.

Listagem 5: Classe `TestController.java`

```
package com.example.demo;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/")
public class TestController {

    @GetMapping
    public String test() {
        return "Hello World!";
    }
}
```

Documento licenciado sob Creative Commons “Atribuição 4.0 Internacional”.