Ciências ULisboa

Aplicações Distribuídas 2019/20

Guião de apoio 7
SSL/TLS

---

1. Introdução ao tema

Uma das formas mais simples de concretizar comunicação segura em aplicações é utilizar o protocolo SSL/TLS (TLS é a versão padronizada do SSL – doravante vamos usar SSL apenas). Este protocolo funciona sobre TCP/IP e faz uso de várias técnicas criptográficas para proteger a autenticação, a integridade e a confidencialidade dos dados transmitidos. A Figura 1 ilustra o protocolo.

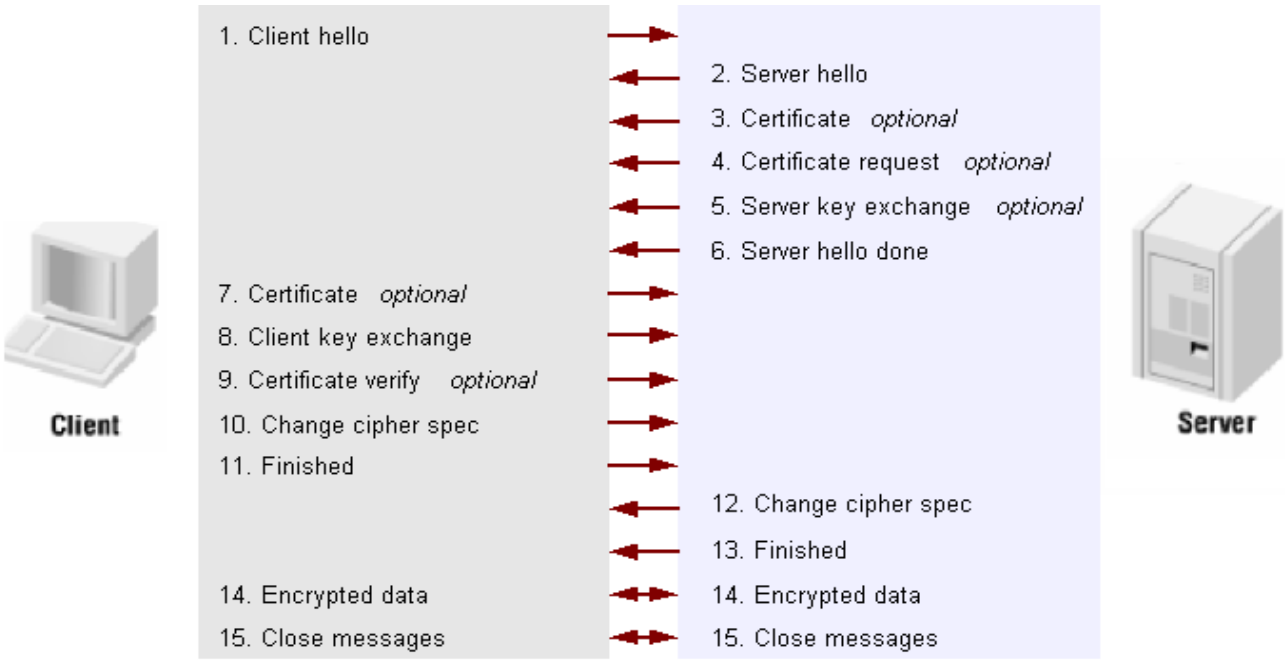

Figura 1. O protocolo SSL/TLS.

Não vamos discutir os detalhes do protocolo neste documento, mas recomendamos os alunos a reverem os conceitos relacionados com o protocolo nos slides da última aula TP. Neste guião vamo-nos centrar na concretização de comunicação segura numa aplicação distribuída em que clientes e servidores comunicam por TCP.

2. Geração de certificados e chaves

Antes de concretizarmos/modificarmos a aplicação segura, precisamos gerar as chaves para as partes envolvidas no sistema. Isto pode ser feito através da ferramenta OpenSSL [1].

Nos comandos a seguir, apresentamos os quatro comandos para criar um ficheiro com a chave privada de um servidor (*server.key*) e um certificado de chave pública auto-assinado (*server.crt*).

1) Crie uma diretoria para os exemplos deste guião.
2) Criação de uma chave privada RSA de 2048 bits (cifrada por uma password) para o servidor
   a. Crie uma diretoria para o servidor com o nome **servidor**.
   b. Nessa diretoria execute o seguinte comando:
      openssl genrsa -des3 -out server.orig.key 2048
   c. O seguinte comando modifica a chave privada RSA para um formato de texto em claro:
      openssl rsa -in server.orig.key -out server.key
3) Criação de uma requisição para assinatura de um certificado de chave pública:
   openssl req -new -key server.key -out server.csr
4) Criação de um certificado de chave pública (auto-assinado) no formato x509:
   openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt
   Caso desejássemos ter o certificado de chave pública assinado por uma CA (cuja chave privada estaria em `ca.key`), o 4º passo seria:
   openssl x509 -req -days 365 -in server.csr -signkey ca.key -out server.crt

Os ficheiros gerados devem estar disponíveis para a aplicação, que deverá receber o *path* para acedê-los, conforme será mostrado na próxima secção.

3. Uso de SSL/TLS no Python

Para transformar uma socket TCP numa socket SSL basta utilizar a função **wrap_socket** do módulo **ssl** do *Python* [2]. Essa função recebe como parâmetros o socket TCP previamente criado e já ligado, bem como um conjunto de informações relativas ao SSL (e.g., versão do protocolo, caminhos para ficheiros com chaves e certificados).

A seguir apresentamos trechos de códigos para concretizar clientes e servidores num cenário de autenticação unilateral. Apenas o cliente verifica a identidade do servidor que comunica com qualquer cliente.

Cliente

```
import ssl

...
sock.connect(...)
ssl_sock = ssl.wrap_socket(sock,
     ssl_version=ssl.PROTOCOL_TLSv1, cert_reqs=ssl.CERT_REQUIRED,
     ca_certs="server.crt")
...
```

Servidor

```
import ssl

...
(conn_sock,addr) = sock.accept()
ssl_sock = ssl.wrap_socket(conn_sock, server_side=True,
     ssl_version=ssl.PROTOCOL_TLSv1,
     keyfile="server.key", certfile="server.crt")
...
```

Note que o cliente deve ter acesso ao certificado do servidor para verificar o certificado que este envia (uma vez que é auto-assinado), enquanto o servidor deve ter acesso a sua chave privada (para provar sua identidade) e seu certificado (para ser enviado aos clientes).

Os próximos dois trechos de código apresentam código similar, mas agora para autenticação mútua entre cliente e servidor.

Cliente

```
import ssl

...
sock.connect(...)
ssl_sock = ssl.wrap_socket(sock,
     ssl_version=ssl.PROTOCOL_TLSv1, cert_reqs=ssl.CERT_REQUIRED,
     ca_certs="server.crt", keyfile="client.key",
     certfile="client.crt")
...
```

Servidor

```
import ssl

...
(conn_sock,addr) = sock.accept()
ssl_sock = ssl.wrap_socket(conn_sock,
     server_side=True, ssl_version=ssl.PROTOCOL_TLSv1,
     cert_reqs=ssl.CERT_REQUIRED, ca_certs="client.crt",
     keyfile="server.key", certfile="server.crt")
...
```

Nestes trechos de código os dois lados requerem três informações: *ca_certs* (certificado contendo a chave publica usada para verificar a assinatura de um certificado), *keyfile* (chave privada usada para provar identidade) e *certfile* (o certificado a ser enviado no protocolo).

---

2. Exercícios fundamentais

1. Copie os programas cliente e servidor apresentados a seguir para a sua área e execute-os no computador do laboratório. Note que este programa consiste na versão final do programa construído no primeiro guião (e depois usado em guiões posteriores).

Cliente (continua na próxima página)

```
import sys, socket as s

if len(sys.argv) > 1:
    HOST = sys.argv[1]
    PORT = int(sys.argv[2])
else:
    HOST = '127.0.0.1'
    PORT = 9999

while True:
```

```
    msg = input('Mensagem: ');

    if msg == 'EXIT':
        exit()

    sock = s.socket(s.AF_INET, s.SOCK_STREAM)
    sock.connect((HOST, PORT))

    sock.sendall(msg.encode())
    resposta = sock.recv(1024)

    print ('Recebi: %s' % resposta.decode())
    sock.close()
```

Servidor

```
import sys, socket as s

HOST = ''

if len(sys.argv) > 1:
        PORT = int(sys.argv[1])
else:
        PORT = 9999

sock = s.socket(s.AF_INET, s.SOCK_STREAM)
sock.setsockopt(s.SOL_SOCKET, s.SO_REUSEADDR, 1)

sock.bind((HOST, PORT))
sock.listen(1)

list = []

while True:
        (conn_sock, addr) = sock.accept()
        try:
                tmp = conn_sock.recv(1024)
                msg = tmp.decode()
                resp = 'Ack'

                if msg == 'LIST':
                        resp = str(list)
                elif msg == 'CLEAR':
                        list = []
                        resp = "Lista apagada"
                else:
                        list.append(msg)

                conn_sock.sendall(resp.encode())

                print ('list= %s' % list)
                conn_sock.close()
        except:
                print ('socket fechado!')
                conn_sock.close()

sock.close()
```

2. Utilize o OpenSSL para criar uma chave e um certificado auto-assinado no servidor.

3. Modifique os programas anteriores para usarem SSL para comunicação, de tal forma que o cliente verifique a identidade do servidor (autenticação unilateral).

4. Utilize o OpenSSL para criar uma chave e um certificado auto-assinado para o cliente.

5. Modifique os programas anteriores para usarem SSL para comunicação com autenticação mútua.

6. Crie uma chave e certificado para uma entidade chamada "gestor" e use a chave dessa entidade para assinar novos certificados para o cliente e o servidor. Modifique os programas cliente e servidor para utilizarem esse novo modelo de certificação. Consegue identificar a vantagem de usar esse modelo?

---

3. Bibliografia e outro material de apoio

https://www.openssl.org/docs/apps/openssl.html
http://carlo-hamalainen.net/blog/2013/1/24/python-ssl-socket-echo-test-with-self-signed-certificate

4. Referências

[1] https://www.openssl.org/
[2] https://docs.python.org/3/library/ssl.html