传输层安全 (TLS) 实验

1 概述

如今，越来越多的数据通过 Internet 传输。但是，当数据在不受保护的公共网络上传输时，这些数据甚至可以被其他人读取或修改。担心通信安全性的应用程序需要对数据加密并检测数据是否被篡改。密码学解决方案可用于实现此目标，但密码算法很多，即使对于同一算法，也可以使用许多参数。为了方便不同的应用程序相互通信，应用程序需要遵循共同的标准。TLS （传输层安全性）就是这样的一个标准。如今，大多数 Web 服务器都使用基于 TLS 构建的 HTTPS 。

本实验的目的是帮助学生了解 TLS 的工作原理以及如何在编程中使用 TLS 。该实验指导学生实现一对 TLS 客户端和服务器程序。在此基础上，学生将进行一系列实验，以便了解 TLS 协议的安全原理。学生还将实现一个简单的 HTTPS 代理程序，以了解当受信任的 CA 受到攻击时，对安全产生的影响。该实验涵盖以下主题：

- 公钥基础设施（PKI）
- 传输层安全（TLS）
- TLS 编程
- HTTPS 代理
- X.509 证书
- 中间人攻击

先决条件 该实验依赖于 PKI 实验。在进行本实验之前，学生应先进行 PKI 实验。

阅读材料 PKI 和 TLS 的详细介绍可以参见：

- SEED 书, *Computer & Internet Security: A Hands-on Approach*, 3rd Edition, by Wenliang Du. 详情请见 `https://www.handsonsecurity.net`.

实验环境 本实验在我们预先构建好的 Ubuntu 20.04 VM（可以从我们的 SEED 网站当中下载）当中测试可行。

2 实验环境

在本实验中，我们使用三台计算机，一台用于客户端，一台用于服务器，另一台用于代理。我们将使用容器表示这些计算机。它们的 IP 地址在下面列出：

```
client: 10.9.0.5
server: 10.9.0.43
proxy:  10.9.0.143
```

容器设置及其命令 请从实验的网站下载 Labsetup.zip 文件到你的 VM 中，解压它，进入 Labsetup 文件夹，然后用 docker-compose.yml 文件安装实验环境。对这个文件及其包含的所有 Dockerfile 文件中的内容的详细解释都可以在链接到本实验网站的用户手册 [1] 中找到。如果这是您第一次使用容器设置 SEED 实验环境，那么阅读用户手册非常重要。

在下面，我们列出了一些与 Docker 和 Compose 相关的常用命令。由于我们将非常频繁地使用这些命令，因此我们在 .bashrc 文件（在我们提供的 SEED Ubuntu 20.04 虚拟机中）中为它们创建了别名。

```
$ docker-compose build  # 建立容器镜像
$ docker-compose up     # 启动容器
$ docker-compose down   # 关闭容器

// 上述 Compose 命令的别名
$ dcbuild       # docker-compose build 的别名
$ dcup          # docker-compose up 的别名
$ dcdown        # docker-compose down 的别名
```

所有容器都在后台运行。要在容器上运行命令，我们通常需要获得容器里的 Shell 。首先需要使用 docker ps 命令找出容器的 ID ，然后使用 docker exec 在该容器上启动 Shell 。我们已经在 .bashrc 文件中为这两个命令创建了别名。

```
$ dockps        // docker ps --format "{{.ID}}  {{.Names}}" 的别名
$ docksh <id>   // docker exec -it <id> /bin/bash 的别名

// 下面的例子展示了如何在主机 C 内部得到 Shell
$ dockps
b1004832e275  hostA-10.9.0.5
0af4ea7a3e2e  hostB-10.9.0.6
9652715c8e0a  hostC-10.9.0.7

$ docksh 96
root@9652715c8e0a:/#

// 注: 如果一条 docker 命令需要容器 ID, 你不需要
```

[1] 如果你在部署容器的过程中发现从官方源下载容器镜像非常慢，可以参考手册中的说明使用当地的镜像服务器

```
//      输入整个 ID 字符串。只要它们在所有容器当中
//      是独一无二的，那只输入前几个字符就足够了。
```

如果你在设置实验环境时遇到问题，可以尝试从手册的 “Miscellaneous Problems” 部分中寻找解决方案。

文件目录 在虚拟机中进行代码编辑比在容器中更为方便，因为我们可以使用我们喜欢的编辑器。为了使虚拟机和容器共享文件，我们使用 Docker `volumes` 在虚拟机和容器之间创建了一个共享文件夹。如果你查看 Docker Compose 文件，就会发现我们已经在某些容器中添加了以下条目。它表示将主机（即 VM）上的 `./volumes` 文件夹挂载到容器内的 `/volumes` 文件夹。我们在虚拟机上将代码写入 `./volumes` 文件夹，就可以在容器内使用它们。

```
volumes:
      - ./volumes:/volumes
```

3 任务 1: TLS 客户端

在此任务中，我们将逐步构建一个简单的 TLS 客户端程序。通过该过程，学生将了解 TLS 编程中的基本要素和安全注意事项。我们将在 client 容器上运行此客户端程序。

3.1 任务 1.a: TLS 握手

在客户端和服务器安全通信之前，首先需要设置几项内容，包括使用哪种加密算法和密钥、哪种 MAC 算法、哪种密钥交换算法等。这些密码学参数需要由客户端和服务器协商，这是 TLS 握手协议的主要目的。在此任务中，我们专注于 TLS 握手协议。以下示例代码启动了一个与 TLS 服务器之间的 TLS 握手（服务器名称需要指定为第一个命令行参数）。

Listing 1: `handshake.py` (在 `Labsetup/volumes` 目录下)

```
#!/usr/bin/env python3

import socket, ssl, sys, pprint

hostname = sys.argv[1]
port = 443
cadir = '/etc/ssl/certs'

# Set up the TLS context
context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)
context.load_verify_locations(capath=cadir)
context.verify_mode = ssl.CERT_REQUIRED
context.check_hostname = True

# Create TCP connection
```

```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((hostname, port))
input("After making TCP connection. Press any key to continue ...")

# Add the TLS
ssock = context.wrap_socket(sock, server_hostname=hostname,
                            do_handshake_on_connect=False)
ssock.do_handshake()   # Start the handshake
pprint.pprint(ssock.getpeercert())
input("After handshake. Press any key to continue ...")

# Close the TLS Connection
ssock.shutdown(socket.SHUT_RDWR)
ssock.close()
```

任务。 使用上面的代码与真实的基于 HTTPS 的 Web 服务器进行通信。可能需要添加其他代码才能完成任务。学生可以在网络上找到 Python 的 SSL 模块手册。请报告以下内容:

- 客户端和服务器之间使用的加密算法是什么?
- 请在程序中输出服务器证书。
- 说明 `/etc/ssl/certs` 的作用。
- 使用 Wireshark 在程序执行期间捕获网络流量，并解释你的观察结果。请说明哪个步骤触发 TCP 握手，以及哪个步骤触发 TLS 握手。解释 TLS 握手和 TCP 握手之间的关系。

3.2 任务 1.b: CA 的证书

在上一个任务中，我们使用 `/etc/ssl/certs` 文件夹中的证书来验证服务器的证书。在此任务中，我们将创建自己的证书文件夹，并将相应的证书放在该文件夹中用于验证。

请创建一个名为 `certs` 的文件夹，并将客户端程序中的 `cadir` 行更改为以下内容。运行客户端程序并报告您的观察结果。

```
cadir = './certs'
```

您需要将相应的 CA 证书放入 `certs` 文件夹中。请使用您的客户端程序查找需要哪些 CA 证书来验证 `www.example.com` 服务器的证书，然后将证书从 `/etc/ssl/certs` 复制到您自己的文件夹中。再次运行您的客户端程序。如果您正确完成了所有操作，您的客户端程序应该可以与服务器对话。

请注意，将 CA 的证书复制到 `./certs` 文件夹是不够的。当 TLS 试图验证服务器证书时，它用颁发者的标识信息生成哈希值，将此哈希值用作文件名的一部分，然后使用该名称在 `certs` 文件夹中查找颁发者的证书。因此，我们需要使用 subject 字段生成的哈希值来重命名每个 CA 的证书，或者我们可以用哈希值创建符号链接。在以下命令中，我们使用 `openssl` 生成哈希值，然后将其用于创建符号链接。

```
$ openssl x509 -in someCA.crt -noout -subject_hash
4a6481c9

$ ln -s someCA.crt 4a6481c9.0
$ ls -l
total 4
lrwxrwxrwx 1 ... 4a6481c9.0 -> someCA.crt
-rw-r--r-- 1 ... someCA.crt
```

额外要求。 请对使用不同 CA 证书的两个不同 Web 服务器执行此任务。

3.3 任务 1.c: 主机名检查

此任务的目的是帮助学生了解客户端进行主机名检查的重要性。请使用客户端程序执行以下步骤。

- 第 1 步: 使用 dig 命令获得 www.example.com 的 IP 地址（你可能要在虚拟机中执行此命令，因为容器中没有安装 dig 命令）:

  ```
  $ dig www.example.com
  ...
  ;; ANSWER SECTION:
  www.example.com.  403   IN A  93.184.216.34
  ```

- 第 2 步: 在容器中修改 /etc/hosts 文件，将下面这一项添加到文件的末尾（其中 IP 地址是从 dig 得到的）。

  ```
  93.184.216.34   www.example2020.com
  ```

- 第 3 步: 在客户端程序中尝试把下面这一行在 True 和 False 之间切换，然后分别使用你的客户端程序连接到 www.example2020.com 。描述你观察到的现象并解释。

  ```
  context.check_hostname = False  #  True 和 False 都试试
  ```

主机名检查的重要性 请基于此实验解释主机名检查的重要性。如果客户端程序不执行主机名检查，那么可能出现的安全方面的后果是什么？请作出解释。

3.4 任务 1.d: 发送和接收数据

在此任务中，我们将数据发送到服务器并获得服务器的响应。由于我们选择使用 HTTPS 服务器，我们需要将 HTTP 请求发送到服务器，否则，服务器将无法理解我们的请求。下面的代码示例演示如何发送 HTTP 请求以及如何读取响应。

```
# 发送 HTTP 请求到服务器
request = b"GET / HTTP/1.0\r\nHost: " + \
```

```
          hostname.encode('utf-8') + b"\r\n\r\n"
ssock.sendall(request)

# 读取服务器发来的 HTTP 回复
response = ssock.recv(2048)
while response:
    pprint.pprint(response.split(b"\r\n"))
    response = ssock.recv(2048)
```

任务。 （1）请将数据发送和接收的代码添加到你的客户端程序中，并报告你的观察结果。（2）请修改 HTTP 请求，以便从 HTTPS 服务器获取所需的图像文件（无需显示图像）。

4 任务 2: TLS 服务器

在执行此任务之前，学生需要创建一个证书颁发机构（CA），并使用此 CA 的私钥为该任务创建服务器证书。另一个 SEED 实验（PKI 实验）已经介绍了如何执行这些操作，这是该实验的先决条件。在此任务中，我们假定已经创建了所有必需的证书，包括 CA 的公钥证书和私钥（`ca.crt` 和 `ca.key` ），以及服务器的公钥证书和私钥（`server.crt` 和 `server.key` ）。应该注意的是，服务器证书中使用的通用名称必须包含学生的姓氏（拼音）和当前年份。

我们将使用 server 容器来运行此 TLS 服务器程序。确保你设置了正确的 DNS 映射，使 TLS 服务器的域名指向 server 容器的 IP 地址。

4.1 任务 2.a. 实现一个简单的 TLS 服务器

在此任务中，我们将实现一个简单的 TLS 服务器。我们使用任务 1 中的客户端程序来测试该服务器程序。下面提供了示例服务器代码。

Listing 2: `server.py` (在 `Labsetup/volumes` 文件夹里)

```
#!/usr/bin/env python3

import socket
import ssl

html = """
HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n
<!DOCTYPE html><html><body><h1>Hello, world!</h1></body></html>
"""

SERVER_CERT    = './certs/server.crt'
SERVER_PRIVATE = './certs/server.key'


context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
```

```
context.load_cert_chain(SERVER_CERT, SERVER_PRIVATE)

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM, 0)
sock.bind(('0.0.0.0', 443))
sock.listen(5)

while True:
    newsock, fromaddr = sock.accept()
    ssock = context.wrap_socket(newsock, server_side=True)

    data = ssock.recv(1024)              # Read data over TLS
    ssock.sendall(html.encode('utf-8'))  # Send data over TLS

    ssock.shutdown(socket.SHUT_RDWR)     # Close the TLS connection
    ssock.close()
```

测试 我们将使用任务 1 中开发的客户端程序来测试该服务器程序。在任务 1 中，客户端程序从 `/etc/ssl/certs` 文件夹加载受信任的证书。在此任务中，CA 由我们创建，其证书并未存储在该文件夹中。我们不建议学生将此 CA 添加到该文件夹，因为这会影响整个系统。学生可以将 CA 的证书存储在 `./certs` 文件夹中，然后按照任务 1 中的说明设置文件夹。请分别使用 `/etc/ssl/certs` 文件夹和 `./certs` 文件夹测试您的程序，描述你观察到的现象并解释原因。

4.2 任务 2.b. 使用浏览器测试服务器程序

在此任务中，我们将使用主机 VM 上的浏览器测试 TLS 服务器程序。首先，用浏览器访问服务器，报告您从浏览器中看到的内容并解释原因。服务器监听的 443 端口是 HTTPS 的默认端口。

为了使浏览器与 TLS 服务器通信，浏览器需要验证服务器的证书。它必须使用证书颁发者 CA 的证书进行验证，但是由于此 CA 是在我们的实验中创建的，因此浏览器不在其受信任的证书列表中。我们需要向其中手动添加 CA 的证书。为此，请在地址栏中键入以下 URL，然后单击页面上的 `View Certificates` 按钮（滚动到底部）。

```
about:preferences#privacy
```

在 `Authorities` 标签中，你将看到已被 Firefox 接受的证书列表。在这里，我们可以导入我们自己的证书。选择证书文件后，请选择 “Trust this CA to identify websites” 选项。你会看到我们的证书现在在 Firefox 接受的证书列表中。

请证明您的浏览器可以成功与 TLS 服务器通信，并且可以显示服务器返回的内容。

4.3 任务 2.c. 有多个名字的证书

许多网站都有不同的 URL，例如，`www.example.com` ，`www.example.org` ，`example.com` 都指向同一 Web 服务器。由于大多数 TLS 客户端程序都实施了主机名匹配策略，因此证书中的公用名必须与服务器的主机名匹配，否则 TLS 客户端将拒绝与服务器通信。

为了使证书具有多个名称，X.509 规范定义了一个叫 Subject Alternative Name （SAN）的扩展。使用 SAN 扩展，可以在证书的 `subjectAltName` 字段中指定多个主机名。

要使用该字段生成证书签名请求，我们可以使用配置文件，并将所有必要的信息放入该文件中（PKI 实验显示了如何在命令行中执行所有操作）。以下配置文件给出了一个示例。它指定 subject 字段的内容，并在扩展中添加 `subjectAltName` 字段。该字段指定多个备用名称，包括通配符名称 `*.bank32.com` 。应该注意的是，该字段还必须包括 common name 字段里的名字，否则 common name 字段里的名字将不会被接受。

Listing 3: `server_openssl.cnf`

```
[ req ]
prompt             = no
distinguished_name = req_distinguished_name
req_extensions     = req_ext

[ req_distinguished_name ]
C  = US
ST = New York
L  = Syracuse
O  = XYZ LTD.
CN = www.bank32.com

[ req_ext ]
subjectAltName = @alt_names

[alt_names]
DNS.1   = www.bank32.com
DNS.2   = www.example.com
DNS.3   = *.bank32.com
```

我们可以使用下面的 "`openssl req`" 命令生成一对公私钥对和一个证书签名请求：

```
openssl req -newkey rsa:2048 -config ./server_openssl.cnf -batch \
            -sha256  -days 3650 -keyout server.key -out server.csr
```

出于安全考虑，默认情况下 CA 签署证书时不会将扩展字段从证书签名请求中复制到最终证书中。为了允许复制，我们需要更改 `openssl` 的配置文件。默认情况下，`openssl` 使用 `/usr/lib/ssl` 目录中的配置文件 `openssl.cnf` 。在此文件中，`copy_extensions` 选项被禁用（注释掉了）。我们不想修改系统的配置文件，所以我们将文件复制到自己的文件夹中，将其重命名为 `myopenssl.cnf` 。然后，我们从该文件中取消以下行的注释：

```
# Extension copying option: use with caution.
copy_extensions = copy
```

我们用下面的程序和证书签名请求为服务器生成证书（`server.crt`），请求中所有的扩展域都会被复制到最终的证书中。

```
openssl ca -md sha256 -days 3650 -config ./myopenssl.cnf -batch \
           -in server.csr -out server.crt \
           -cert ca.crt -keyfile ca.key
```

学生需要展示自己的服务器可以支持多个主机名，包括其各自域名中的主机名。

5 任务 3: 一个简单的 HTTPS 代理

TLS 可以抵御中间人攻击，但前提是必须保证公钥基础设施是安全的。在此任务中，我们将演示如果 PKI 受到破坏（即某些受信任的 CA 被攻破或服务器的私钥被盗），如何对 TLS 服务器进行中间人攻击。

我们将实现一个名为 `mHTTPSproxy` 的简单的 HTTPS 代理（`m` 代表 `mini` ）。代理程序仅将任务 1 和 2 中的客户端和服务器程序集成在一起。图 1 中说明了它的工作方式。我们将在 proxy 容器上运行代理。

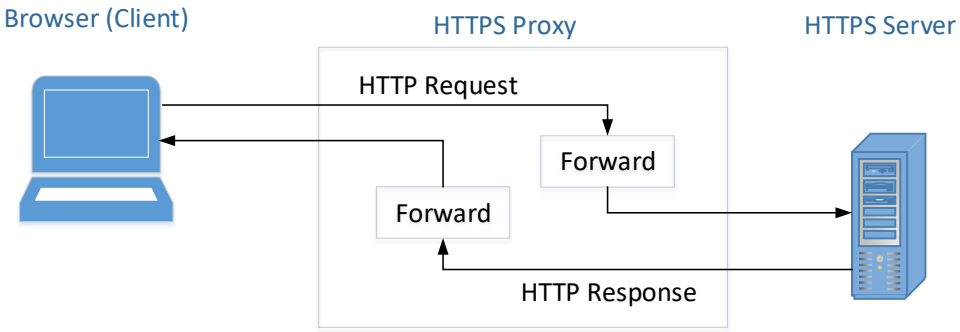

图 1: `mHTTPSproxy` 是如何工作的

代理实际上是 TLS 客户端和服务器程序的组合。对于浏览器，TLS 代理只是一个服务器程序，它从浏览器（客户端）获取 HTTP 请求，并向其返回 HTTP 响应。代理本身不会生成任何 HTTP 响应，而是将 HTTP 请求转发到实际的 Web 服务器，然后从 Web 服务器获取 HTTP 响应。对于实际的 Web 服务器，TLS 代理只是一个客户端程序。收到响应后，代理将响应转发给浏览器，即真正的客户端。因此，通过集成在前两个任务中实现的客户端和服务器程序，学生应该能够实现基本的代理工作。

应该注意的是，此任务的目的是使用简单的代理来了解当 PKI 基础设施被攻陷后，中间人攻击如何工作。我们实现的并不是一个很完善的 HTTPS 代理，因为要使代理在每个 Web 服务器上都工作并不容易，需要考虑 HTTP 协议的许多方面。由于本实验的重点是 TLS ，因此学生可以选择两个不同的服务器，并演示其代理服务器可用于这些服务器。对完善的 HTTPS 代理感兴趣的学生可以从网络上寻找其他资料，例如开源的 `mitmproxy` 。

处理多个 HTTP 请求。 浏览器可能会同时向服务器发送多个 HTTP 请求，因此在从浏览器接收到 HTTP 请求之后，最好生成一个线程来处理该请求，以便代理程序可以同时处理多个请求。以下代码段显示了如何创建一个新的线程来处理每个 TLS 连接。

```
import threading

while True:
    sock_for_browser, fromaddr = sock_listen.accept()
    ssock_for_browser = context_srv.wrap_socket(sock_for_browser,
                                                server_side=True)
    x = threading.Thread(target=process_request, args=(ssock_for_browser,))
    x.start()
```

该线程将执行 process_request 函数中的代码，该函数将 HTTP 请求从浏览器转发到服务器，然后将 HTTP 响应从服务器转发到浏览器。以下提供了代码框架：

```
def process_request(ssock_for_browser):
    hostname = 'www.example.com'

    # Make a connection to the real server
    sock_for_server  = socket.create_connection((hostname, 443))
    ssock_for_server = ... # [Code omitted]: Wrap the socket using TLS

    request = ssock_for_browser.recv(2048)

    if request:
        # Forward request to server
        ssock_for_server.sendall(request)

        # Get response from server, and forward it to browser
        response = ssock_for_server.recv(2048)
        while response:
            ssock_for_browser.sendall(response) # Forward to browser
            response = ssock_for_server.recv(2048)

    ssock_for_browser.shutdown(socket.SHUT_RDWR)
    ssock_for_browser.close()
```

客户端设置。 对于此任务，由于我们将使用浏览器，因此将使用虚拟机作为客户端（受害者），而不是使用 client 容器。在实际攻击中，当受害者访问 Web 服务器（例如 www.example.com ）时，我们将发起攻击将受害者重定向到我们的代理。这通常通过 DNS 攻击、BGP 攻击或其他重定向攻击来完成。为了简化，我们不会进行此类攻击，我们只需将以下条目添加到主机 VM 上的 /etc/hosts 文件中（10.9.0.143 是 mitm-proxy 容器的 IP 地址）。

```
10.9.0.143   www.example.com
```

通过执行上述操作，我们模拟了重定向攻击的结果：受害者到 www.example.com 的流量将被重定向到攻击者机器上，你的 mHTTPSproxy 在那里等待 HTTP 请求。

任务。 学生应该实现简单的 `mHTTPSproxy`，并在代理容器上运行它。在此 MITM 攻击中，我们假设攻击者已经攻陷了受信任的 CA ，能够使用 CA 的私钥为任何域名生成伪造但有效的证书。在本实验中，浏览器已信任任务 2 中生成的 CA 证书，并且我们假定此 CA 的私钥已被攻击者盗取，因此你（攻击者）可以使用它为任何 Web 服务器伪造证书。请在以下情况下展示您的 MITM 攻击：

- 针对你自己的服务器发起中间人攻击。
- 在真实的 HTTPS 网站上发起 MITM 攻击。你可以选择一个网站，找到一个需要登录的帐户，然后使用你的 MITM 代理窃取密码。许多流行的服务器（例如 Facebook ）具有复杂的登录机制，你可以找一个具有简单登录机制的服务器。如果你使用的是真实密码，请记住在实验报告中隐藏密码。

清理。 完成此任务后，请记得从浏览器中删除 CA 的证书，并且删除在虚拟机上添加到 `/etc/hosts` 的所有条目。

6 递交

你需要提交一份带有截图的详细实验报告来描述你所做的工作和你观察到的现象。你还需要对一些有趣或令人惊讶的观察结果进行解释。请同时列出重要的代码段并附上解释。只是简单地附上代码不加以解释不会获得学分。