Transport Layer Security

Networked Systems (H) 2023-2024 – Laboratory Exercise 3
Dr Colin Perkins, School of Computing Science, University of Glasgow

1 Introduction

The laboratory exercises for Networked Systems (H) will introduce you to network programming in C using the Berkeley Sockets API, and help you understand the operation and structure of the network. The exercises will help you practice C programming, building on the Systems Programming (H) course, and introduce network programming in C. Other exercises will illustrate key points in the operation of the network. The laboratory exercises are intended to complement the material covered in the lectures. Some expand on the lectures to give you broader experience in a particular subject. Others exercises cover material, such as network programming in C, that's better taught by doing than by lecturing.

This exercise explores the use and deployment of the Transport Layer Security protocol. **This is a formative exercise, and is not assessed.**

2 Formative Exercise 3: Transport Layer Security

Lecture 3 discussed secure communications and introduced the Transport Layer Security Protocol (TLS). There have been numerous different versions of TLS used over the years and each version supports a number of different encryption and key exchange algorithms. There is also a rich ecosystem of different Certificate Authorities (CAs) that verify the authenticity of websites using TLS. In this exercise, you will conduct measurements of this ecosystem to understand how TLS is used in practice.

The following Python program shows how to connect to port 443 (HTTPS) on a web server, negotiate TLS within that TCP connection, then retrieve details of the TLS version, cipher suite (i.e., the encryption algorithm used), and the peer certificate (i.e., the digital signature, signed by a Certificate Authority, verifying the identity of the site). The connection is then closed without requesting to retrieve any web page.

```
import socket # Import Berkeley sockets library
import ssl    # Import TLS library

hostname = "www.csperkins.org"
context  = ssl.create_default_context()

# Create a TCP connection to the server:
with socket.create_connection((hostname, 443)) as sock:
  # Negotiate TLS within that TCP connection:
  with context.wrap_socket(sock, server_hostname=hostname) as ssock:
    # Retrieve details about the ngotiated TLS parameters:
    tls_version = ssock.version()
    tls_cipher  = ssock.cipher()
    tls_certificate = ssock.getpeercert()
    print(f"TLS version: {tls_version}")
    print(f"TLS ciphers: {tls_cipher}")
    print(tls_certificate)
```

You will note that the TLS library in Python is in the package `ssl`. SSL (Secure Sockets Layer) is the name used for TLS versions prior to TLS 1.0. Python kept the SSL name, but updated the package to support current TLS standards.

Running the above program will produce output like the following:

```
TLS version: TLSv1.3
TLS ciphers: ('TLS_AES_256_GCM_SHA384', 'TLSv1.3', 256)
{'OCSP': ('http://r3.o.lencr.org',),
 'caIssuers': ('http://r3.i.lencr.org/',),
 'issuer': ((('countryName', 'US'),),
            (('organizationName', "Let's Encrypt"),),
            (('commonName', 'R3'),)),
 'notAfter': 'Mar  8 10:09:14 2024 GMT',
 'notBefore': 'Dec  9 10:09:15 2023 GMT',
 'serialNumber': '04247A754C803232E681A4141766A01D91A3',
 'subject': ((('commonName', 'www.csperkins.org'),),),
 'subjectAltName': (('DNS', 'www.csperkins.org'),),
 'version': 3}
```

It can be seen that the server is running TLS v1.3. It has negotiated the `TLS_AES_256_GCM_SHA384` encryption and authentication algorithm (the Advanced Encryption Standard in Galois Counter Mode with 256 bit keys and SHA384 authentication). And the identity of the site is verified by the "Let's Encrypt" certificate authority with a certificate that's valid until 8th March 2024 at 10:09:14 GMT.

Re-run the Python code yourself, demonstrating that you can replicate the above output. Then run it with several different host names, to check that it works more broadly.

The "Majestic Million" (`https://majestic.com/reports/majestic-million`) is a list that claims to measure the one million most popular websites. Download this list and extract the domain names for a few hundred of the most popular sites (filter the list to exclude any inappropriate sites). Modify the above Python program to read the list of domain names you've extracted, connect to each in turn, and retrieve and save details of the TLS version, ciphers, and certificate authority for each.

Save the output to a file, one line per site, formatted something like the following, showing site name, TLS version, cipher suite, and certificate authority:

```
google.com        TLSv1.3  TLS_AES_256_GCM_SHA384         Google Trust Services LLC
facebook.com      TLSv1.3  TLS_CHACHA20_POLY1305_SHA256   DigiCert Inc
youtube.com       TLSv1.3  TLS_AES_256_GCM_SHA384         Google Trust Services LLC
twitter.com       TLSv1.3  TLS_AES_256_GCM_SHA384         DigiCert Inc
instagram.com     TLSv1.3  TLS_CHACHA20_POLY1305_SHA256   DigiCert Inc
linkedin.com      TLSv1.2  ECDHE-RSA-AES256-GCM-SHA384    DigiCert Inc
apple.com         TLSv1.3  TLS_AES_256_GCM_SHA384         Apple Inc.
microsoft.com     TLSv1.3  TLS_AES_128_GCM_SHA256         Microsoft Corporation
wikipedia.org     TLSv1.3  TLS_AES_256_GCM_SHA384         DigiCert Inc
wordpress.org     TLSv1.3  TLS_AES_256_GCM_SHA384         Sectigo Limited
youtu.be          TLSv1.3  TLS_AES_256_GCM_SHA384         Google Trust Services LLC
play.google.com   TLSv1.3  TLS_AES_256_GCM_SHA384         Google Trust Services LLC
github.com        TLSv1.3  TLS_AES_128_GCM_SHA256         DigiCert Inc
pinterest.com     TLSv1.3  TLS_AES_128_GCM_SHA256         DigiCert Inc
maps.google.com   TLSv1.3  TLS_AES_256_GCM_SHA384         Google Trust Services LLC
vimeo.com         TLSv1.3  TLS_AES_256_GCM_SHA384         Cloudflare, Inc.
docs.google.com   TLSv1.3  TLS_AES_256_GCM_SHA384         Google Trust Services LLC
bit.ly            TLSv1.3  TLS_AES_256_GCM_SHA384         DigiCert Inc
wordpress.com     TLSv1.3  TLS_AES_256_GCM_SHA384         Sectigo Limited
amazon.com        TLSv1.2  ECDHE-RSA-AES128-GCM-SHA256    DigiCert Inc
mozilla.org       TLSv1.3  TLS_AES_256_GCM_SHA384         Let's Encrypt
...
```

Write scripts to analyse your results to answer the following questions:

1. What percentage of the sites are running TLS 1.3 and what percentage are still running some older version of TLS? What's the oldest version of TLS in use in these sites?
2. How many different cipher suites (combinations of encryption and authentication algorithms) are in use? What are the cipher suites that are used? Read up on the different combinations of encryption and authentication to try to understand why different sites make different choices.
3. The identity of each site is verified by a certificate authority. How many different certificate authorities do you see being used? What is the most popular? Look into who operates the most popular certificate authorities, what are their policies for verifying the identify of sites, and how much do they charge to do so?

Finally, try to find out what is the complete set of certificate authorities your browser trusts to verify the identity of web sites. This is often known as the set of "root certificates" that the browser trusts or the "root store". Is it possible and/or easy to find this information for your browser and operating system? What is the policy that your browser vendor uses to add/remove certificate authorities from this store? Looking at the list of trusted certificate authorities, and the range of names each is allowed to verify, what do you think about the security of the web?

3 Discussion

This is a formative exercise. It is not assessed, and you do not need to submit your code or results. The goals are (i) to give further practise in measuring and understanding the behaviour of the network; and (ii) to complement the material covered in Lecture 3. Discuss the behaviour you see with the lecturer or lab demonstrators, to ensure you understand the characteristics of TLS.