Secure Communications and Protocol Ossification

Networked Systems (H) 2021-2022 – 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.

There are a mixture of formative and summative exercises. The formative exercises will give you practice in programming networked systems in C, and allow you to gain further experience in C programming. They are not assessed. The two summative exercises will focus on security and protocol ossification, and on understanding the network topology. They are assessed and together are worth a total of 20% of the marks for this course.

This laboratory exercise reviews secure communication and protocol ossification, and supports the first assessed exercise, which will be made available on 27 January 2022. **That exercise is assessed and worth 10% of the marks for this course**.

2 Principles of Secure Communication

The Internet relies on secure communications protocols to protect user communications. Data traversing the network is encrypted to preserve its confidentiality and protect the privacy of users, and authenticated to identify the sender and demonstrate that it has not been modified in transit. The most commonly used secure communications protocol is Transport Layer Security (TLS).

Unlike the Berkeley Sockets API, that provides a cross-platform interface to TCP connections, there is no standard programming interface to TLS. Each operating system has its own secure networking API, and there are significant differences between them. Further, the most widely used TLS library for Linux, `OpenSSL`, has a complex and poorly designed programming interface that is not a good example for learning. Accordingly, this lab exercise does not involve TLS programming. Rather, it is focussed on understanding some of the properties and limitations of TLS.

Securing connection metadata: TLS operates within a TCP connection. That is, a client first connects to a server using TCP as normal. Then, once the TCP connection has been established, it uses TLS to negotiate the encryption needed to secure communications sent using that TCP connection. A consequence of this model this is that the information in the TCP and IP packet headers (the "metadata") is not encrypted or authenticated – only the data sent inside the TCP segments is protected. Review the information contained in the TCP and IP packet headers (slide 5 of Lecture 2a). Consider what an adversary that is able to snoop on the network traffic might be able to learn about your communication from looking only at those headers. Do you think that it is a significant risk that such metadata is exposed when using TLS over TCP for Internet communications? Discuss with the lecturer or with one of the demonstrators.[1]

[1] You might want to read the paper "What Can You Learn from an IP?", linked near the end of `https://irtf.org/anrw/2019/program.html` for further discussion of this topic (a video recording of the paper presentation is at `https://youtu.be/pEnT2BJvyv0` and may be more accessible than the paper).

Securing data in transit and data at rest: The nature of secure communication protocols, such as TLS, is that they protect data in transit between endpoints but not data stored within the endpoints. For example, a messaging application might use TLS to protect communications between users and the messaging server, but relies on the messaging server operator to securely store messages on the server. Think of some service or application that you use, and try to find out whether it uses TLS, or some other security protocol, to protect its communications (HTTPS, which is HTTP over TLS is a frequently used choice for protecting communications). Then, see if you can find any information about how the data stored on the servers is protected and who has access to the data. Are you convinced the service is secure? To what extent do you think it is feasible for service providers to secure data they store, so it is only available to the users of the service? How might this affect the safety and trustworthiness of cloud computing services?

3 Transport Layer Security (TLS) v1.3

TLS comprises a handshake protocol and a record protocol. The handshake protocol is used to agree encryption keys and algorithms. The record protocol is used to exchange protected blocks of information between client and server.

Ossification of the TLS Handshake: If you read the TLS specification, RFC 8446 (`https://tools.ietf.org/html/rfc8446`) or "The New Illustrated TLS Connection" (`https://tls13.ulfheim.net`) which explains the handshake protocol in detail, it becomes clear that TLS contains a lot of complexity to work around problems with middleboxes. In particular, much of the TLS v1.3 handshake protocol is spent pretending to be a TLS v1.2 handshake, with the TLS v1.3 specific parts being hidden as 'extensions' to the core protocol. This is due to *ossification* of the protocol, caused to middleboxes (firewalls, gateways, proxies, etc.) that do not correctly implement TLS version negotiation.

Read the blog post at `https://blog.cloudflare.com/why-tls-1-3-isnt-in-browsers-yet/` that describes the issue in more detail. Think about the issues raised. What hope do we have to extend and modify the design of the Internet if an essential protocol like TLS, used by almost every web browser and web server in the world, is so hard to upgrade? Does the proposal for GREASE described in the blog post make sense? What are changed in the time since that blog post was written? Discuss these issues with the lecturer or lab demonstrators.

The TLS Record Protocol: Once the TLS handshake protocol has negotiated encryption keys and other security parameters, the TLS record protocol is used to exchange the actual data. When using TCP without TLS, the TCP connection doesn't preserve message boundaries. That is, if two messages, each comprising 100-bytes, are sent over a TCP stream, then all 200 bytes will appear at the receiver in the order sent, but they will not necessarily appear as two blocks of 100 bytes. When TLS is used, the record layer ensures that message boundaries are preserved, so if a TLS sender writes two 100-bytes messages into a TLS connection operating over TCP, TLS will ensure they appear at the receiver as two 100-byte messages. Which approach do you think makes most sense, to deliver a sequence of messages, preserving message boundaries, or to deliver a sequence of bytes and to not worry about separating messages? Why?

4 QUIC

Lecture 4 will discuss QUIC, a new transport protocol that is under development within the IETF, and that aims to replace TLS and TCP. One of the key benefits of the QUIC transport protocol is that it can overlap the initial connection establishment handshake and the TLS handshake, reducing connection establishment latency to one round-trip time.

As a new transport protocol, you might expect QUIC to fit into the transport layer of the protocol stack. In the Internet, this means it should run over IP, in the same way that TCP does. QUIC does not do this. Rather, it runs above *UDP*, since experience has shown that it is difficult to deploy new transport protocols running directly over IP because firewalls tend to block traffic using unknown protocols. This is known as protocol ossification, and is a more severe version of the issues found with the TLS handshake. There is a tutorial introduction to QUIC at `https://www.youtube.com/watch?v=CtsBawwGwns`. Watch the first 30 minutes of this, which discuss these issues and how they affect the design of QUIC. Think about why protocol ossification occurs and whether it makes sense for QUIC to run over UDP. Discuss with the lecturer or demonstrators to make sure you understand the problems.[2]

5 Discussion

The goals of this exercise are to further your understanding of secure communication and TLS, and to make you think about some of the issues involved. There is no right answer to any of the questions posed. Rather, there are a series of trade-offs and what makes sense depends on what are the goals for which you are optimising. **There is a lot of material here, some of which is expected to be difficult. You are not expected to understand all the details of how TLS works.** Think about the issues raised, and **discuss them with the lecturer and/or lab demonstrators**, to help you understand different perspectives.

The first assessed exercise should be completed in parallel with this lab exercise, and will build on the knowledge gained.

[2]The last 30 minutes of the tutorial video provide a detailed description of how the QUIC protocol works, and might be of interest. The middle 30 minutes discuss how to use a particular QUIC library, and are perhaps less generally useful.