CSCI468/968 Advanced Network Security Programming Project

2020/02/21

Project Description

In this project, our goal is to build a secure proxy tool which enable the end users to communicate with the remote server in a secure manner. The secure proxy can be used in many situations in the modern network applications. For example, one could use the secure proxy to protect his network privacy by first connecting to the proxy server, and then the server will in turn try to connect to the real network host for the user. In this way, the user' s real identity is protected since the network traffic observed only points to the proxy server instead of the user himself. Another real-world application is that when the branch office wants to contact to the headquarters which locates in the different geographic locations, the secure proxy server can be used to receive the traffic from the branch office and forward to the headquarters backend server.

In the above applications, the network communication between the proxy server and the headquarters backend server or the server which the end users try to access is out of the proxy server' s concern. The proxy server' s main task is to ensure that the network traffic between the end user and the proxy server is securely protected. By securely protected, here we specifically mean that

1. network traffic is encrypted to resist against passive eavesdropping attack;

2. The integrity of the traffic is ensured to resist against the active insertion attack;

3. Assume the proxy server provides services to many users, thus user identification login procedure is required;

4. Authenticated key exchange protocol should be deployed to ensure the secret keys used by the cryptographic algorithms can be agreed securely with forward security property guaranteed.

Task 1

Write a simple proxy server tool to achieve the following functionalities.

(1). The program includes client part and the server part.

(2). The client part runs on the end user' s computer, and it is responsible to collect the user' s outgoing traffic and forward to the

remote proxy server.

(3). Socks5 protocol is used to forward the network traffic, which works on the transport layer. All the network traffic using TCP or UPD protocol can be forwarded by the client program. Thus, implementing the Socks5 protocol is one of the main jobs of task 1.

(4). The server part runs on the proxy server' s computer, and it is responsible for receiving the traffic from the client program. It should be able to analysis the socks5 protocol, extract the real network address the user wants to visit from the network packets, and then access the corresponding web server. After receiving the reply from the web server, the proxy will then forward it back to the client side.

(5). Use Wireshark to capture the traffic to show that the unencrypted message can be easily discovered.

Running example:

Before starting the program, we should let the OS to forward all the traffic by using socks5 protocol to the local address that the client is listening (Can be easily set in Windows, OSX and linux).

Suppose the client program is named "client.go" and listens to the local address "127.0.0.1: 8488". The proxy server can be simulated on the same local machine by using the local address "127.0.0.1:8489". Then we can start the client program by running:

```
go run client.go -c 127.0.0.1:8488 -s 127.0.0.1:8489
```

The server program is named “server.go”, and can be started as

go run server.go :8489

To test the program running correctly, you should be able to visit websites as usual without any problem. On the two opened terminals, you should be able to see the log information popping up as you visit the website.

Hint: Please refer to https://www.ietf.org/rfc/rfc1928.txt

for the details of socks5 protocol

Task 2

Based on the program written in Task1, we will add the confidentiality and integrity protection to the secure proxy tool, so that we can resist against the passive eavesdropping attackers. We only consider using symmetric key cryptography to achieve the goal in this task.

(1). Client and the server side both share a common password pw with 8 characters. pw can be feed into the program through the command line when starting both the client and the server program.

(2). Use PKDF2 to derive the secret K for the symmetric key cryptography.

(3). Client-side program performs the following symmetric key encryption on all the local traffic that was collected in task 1 by the socks5 protocol. Provide the end users the option to choose one of

them.

a). AES-GCM (authenticated encryption)

b). Chacha20Poly1305 (authenticated encryption)

Hint:

a). Transfer user password to the secret key of symmetric key cryptography needs special attention so to resist against offline brute force attack. bcrypt is one of the professional algorithms to achieve this task. If you use go, then bcrypt has already been included in the crypt package, which should be called directly.

b). The authenticated encryption algorithm is a kind of symmetric key encryption algorithm with integrity property embedded. For example, when encrypting the message M with length n, it will generate the ciphertext C and an authenticated tag t. C and t will be sent to the receiver, and the receiver will first perform the integrity check to see whether the ciphertext C has been modified or not. If not, then decrypt C to retrieve M. The whole authenticated encryption process should be wrapped inside one API so that to hide the details from the caller.

Task 3

In task 2, we achieved the basic secure channel between two parties in a very basic setting, but it is not secure in many other scenarios. Especially the derive key will be remained to be the same each time. In this task, we try to improve over the solution in Task 2 in the

following two ways:

1. Both ends still share a password, but instead we apply a PAKE protocol to set up the secure channel.

2. Both ends know the public key of the other side. Take advantage of the AKE protocols learned in the lecture to establish the secure channel.

Task 4

Since the proxy server may provide service to many end users, thus it is required for the proxy server to manage the various user accounts locally. In this task, we need to add the identification protocol to the VPN service. Again, we assume the public key of the proxy server is known to the client in advance. Here we take advantage of the FIDO2 protocol (WebAuthn standard) for the identification. You can take advantage of the WebAuthn library for this task (https://webauthn.io/). Local biometric authentication is required to unlock the private key according to the FIDO2 specification. You are required to use your local fingerprint authentication device (on your laptop if you have), or you will be provided with a Yubikey.

Requirement:

All cryptographic algorithms can be called from the library

You are free to choose any programming languages; however, it is

recommended to use Golang.

While the algorithms can be called from the library, it is not allowed to copy any part of the higher functionalities from any other sources unless specifically specified.

Four tasks should be submitted independently. Each submission should include the source code and one report in word file. The report should address the protocol design details, explanation of the source code, the running procedure (screenshot) and other necessary results.

The submission deadlines are as follows:

Task 1: 3/13 (week 5), before 22:00

Task 2: 3/27 (week 7), before 22:00

Task 3: 4/17 (week 10), before 22:00

Task 4: 5/15 (week 14), before 22:00