Virtual Private Network (VPN) Project

1 Overview

A Virtual Private Network (VPN) is used for secure extension of a private network over an insecure network, such as the Internet. You can think of a VPN as a secure link, *tunnel*, established over a public infrastructure. There are many different protocols that can be used for secure VPNs. In this project, we will use Secure Socket Layer/Transport Layer Security (SSL/TLS), for simplicity just referred to as "SSL".

The learning objective of this project is for students to master the network and security technologies underlying SSL VPNs, and to gain insight into the general principles for VPNs. The design and implementation of SSL VPNs exemplify a number of security principles and technologies, including cryptography, authentication, key management, key exchange, and public-key infrastructures.

2 Description of the SSL VPN

The SSL VPN consists of two parts: a client and a server. The clients connects to the server over SSL. The client and the server then perform *mutual authentication*, by exchanging certificates with each other. The server authenticates the client by veryfing the client's certificate, and vice versa. We use SSL for the authentication, which means that the authentication is carried out as part of the regular ClientHello/ServerHello handshake in SSL.

After the client and the server have authenticated each other, the next step is to set up the tunnel, that is, the secure channel for communication. As a matter of fact, it is part of the SSL protocol already to establish a secure TCP connection between client and server. But we do not want to use a secure TCP connection for our VPN; we want a secure tunnel over UDP instead[1].

The tunnel is secured by encrypting it with symmetric encryption (AES in CBC mode). Therefore, the next thing the server does after authentication is to generate an AES key, and an initialization vector. The key and the vector are sent over the SSL connection to the client. When both the client and the server have the key and the initialization vector, they can start sending AES-encrypted UDP datagrams to each other.

In the client and server, the tunnel will be visible as a *tunnel interface*; a virtual network interface with the name "TP-tun0". When the client wants to send a packet over the tunnel, for instance, it does that by directing the packet out onto the tunnel interface. The server will receive the packet on its corresponding tunnel interface. Hence, in this way, the tunnel can be though of as a direct link that connects the two tunnel interfaces on the client and the server to each other.

3 Project Environment

You will implement this project on a pre-built 64-bit virtual machine (VM). Please refer to the project module on the course web in Canvas for download instructions. You need to install Oracle VirtualBox 64-bit version on your computer so that you can import the VM. For best experience, you are encouraged to have a 64-bit OS installed in your computer. During the import, if VirtualBox complains about the network adapter, go into the menu to change network settings and pick what is available in your OS. Make sure that the first thing you do after the import is to take a snapshot of the VM so that you can return back to this fresh state anytime you feel like you need to start all over again. The credentials for the VM are as follows:

[1]There are several reasons why we want to use UDP for the tunnel instead of TCP; one of the main reasons is that we want to be able to run TCP over the secure tunnel. If we were to use TCP for the secure tunnel, it would mean that we would run TCP over TCP. This could have negative implications for performance.

```
Username: cdev
Password: IK2206
```

In the VM, you will find source code pre-installed for the VPN. In fact, the code implements a fully functional VPN, called "TP", but *without security*. So your job is to add security to TP.

3.1 Networking

TP's server and client applications have to run in separate environments, therefore you need to have one VM for each. The server VM and the client VM must have an IP connectivity via either of the following approaches.

3.1.1 One host and two guests

In this approach, you launch both the server VM and the client VM on your host machine and configure the network settings to use the *Internal Network* feature of VirtualBox. Each VM then needs to have two network adapters; one is for you to establish an SSH session (see Section 3.2) and the other for connecting to the custom internal network.

Navigate to the network settings in the server VM configuration and make sure that the already enabled adapter is attached to host-only adapter. Then click the Adapter 2 tab, enable it, attach it to internal network, and pick a name for your custom network. Repeat the same steps for the client VM, and when you are attaching the second adapter, select the custom internal network you named earlier from the drop-down menu.

The next step is to configure the new adapter in the server VM. Run the command *ip link show* in a terminal window to see the interface name (starting with "enp") of the newly attached adapter. Assign an IP address to the adapter and bring it up using the command "sudo ifconfig *enpxxx ipaddress/prefixlength* up". Repeat the same steps for the client VM, making sure that the server and the client VMs are in the same subnet on the internal network. Verify your setup by pinging the server VM's IP from your client VM.

3.1.2 One guest per each host

In this approach, one student in the group hosts the server VM on his/her computer, and the other student hosts the client VM on another computer. The student computers are configured to share the Internet access with the VMs so that they can communicate with each other over the Internet. One network adapter per VM will be sufficient.

Navigate to the network settings in the server VM configuration and change Adapter 1's attachment from host-only adapter to *NAT*, click Advanced, then click *Port Forwarding*. Add a rule named SSH, set protocol to TCP, leave the host IP and the guest IP blank, and finally set the host port[2] and the guest port to 22. This rule will allow you to establish an SSH session with the VM from your host machine using *localhost* as the SSH session destination. Add two more rules for TCP and UDP 15000 ports, on which the server application listens on for control and data channels in the server VM. In the host machine that runs the client VM, follow the same steps and configure a forwarding rule for SSH only as the client application does not listen on any port.

Note that with this approach, the server IP parameter you pass to the client application via the run configuration has to be **the public IP address** of the computer running the server VM. If that computer does not have a public IP address (which would typically be the case on a home network, for instance), you should forward the mentioned ports in your NAT router as well.

[2]Use a different host port to avoid a conflict if an SSH server service is already running on your host machine.

3.2 Development, Compiling and Running

The source code for TP comes installed in the VM as a project in the Eclipse Oxygen IDE. However, to keep down the size of the VM image, and to make it easier to manage, the VM does not have a graphical desktop interface. It only has a command line interface. So how can you then run the Eclipse Oxygen IDE? Use the desktop environment on your own computer!

You are supposed to SSH into the VM with X11 forwarding enabled in your SSH client so that any graphical content can be forwarded from the guest VM to your computer. In order for your computer to visualize the forwarded content, you need an X server application installed. For Windows 64-bit OS, VcxSrv is highly recommended. For other Windows versions, you can use Xming. IF you have a Mac, XQuartz is the preferred X server.

To start Eclipse, type in the following in your SSH session.

```
cdev@Ubuntu:~$ cd eclipse/cpp-oxygen2/eclipse/
cdev@Ubuntu:~/eclipse/cpp-oxygen2/eclipse$ sudo ./eclipse
[sudo] password for cdev:
```

If X11 forwarding components are properly set, the Eclipse IDE should show up on your host machine.

The TP project in Eclipse is configured to compile each time you run. There exists two run configurations; TP Server and TP Client. You need to update the variables of the TP Client run configuration so that it passes your TP Server's IP address. Use the "TP Server" run configuration in your VPN server VM and the "TP Client" in your VPN client VM.

3.3 Git

The source code for TP is located in a local Git repository in the VM, and you can use the built-in support for Git in Eclipse if you want to have version control for your project. You can also add a remote Git repository, which would enable you to share the source code on several computers.

4 Project Tasks

In its current form, the TP program can establish a tunnel between the server and client by encapsulating IP datagrams in UDP packets. However, the IP datagrams carried within UDP packets are readable to any malicious interceptor. Your task is to improve the current code and secure the tunnel using the OpenSSL libraries.

4.1 Encryption, Decryption and Key Exchange

The TP program relies on the functions implemented in the *sslUtils.cpp* file for security functionalities. However, those functions do not do anything yet. Your primary task is to modify the functions to secure the tunnel. Note that you **shall not modify** any other file in the project.

In order to pass, your implementation shall provide **all** of the following,

- The confidentiality of the packets flowing through the tunnel shall be provided using symmetric cryptography.
- The keys and related material for the symmetric cryptography shall be randomly generated, then exchanged over the control channel that is secured by asymmetric cryptography.
- Both parties shall be authenticated during the handshake. Authentication shall fail if,

- The certificate presented by either party is not signed by the trusted root CA.
- The common name (CN) in the subject field of the certificate does not meet the PKI requirements, specified in Section 4.2.

4.2 PKI and Certificates

In order for the server and the client establish an SSL-VPN tunnel using the TP program, both parties need to have their own identity certificate as well as the corresponding private key. Both certificates have to be signed by the same certificate authority (CA).

OpenSSL framework is installed in the VM for you to create your own PKI infrastructure. 1) Configure basic parameters such as directory, database index file and serial number in a conf template file[3]. 2) Create a self-signed certificate for the CA using the command *openssl req* and passing your configuration file as a parameter. Output will be a root CA certificate and its corresponding private key. 3) Generate private keys for both the server and the client by running *openssl genrsa* command in their respective VMs. 4) Generate certificate signing requests (CSR) for the server and the client by executing the command *openssl req* in their respective VMs. 5) Sign the CSRs at the server VM using the *openssl ca* command. Finally, place the signed certificates of the server and the client in their respective VMs.

In order to follow the steps above better, use your favorite search engine to gather information on creating your own PKI using OpenSSL and specific command syntax. There are plenty of guides and tutorials online.

Certificate verification involves two steps. The first is to validate the digital signatures. This is done by SSL as part of the intial handshake, provided that you have programmed the TP client and server to do so. The second step involves verifying the content of the certificates. For this, the following rules apply for the "common name" (CN) part of the certificate subject field:

- The CN for the CA certificate should be the string "TP CA" followed by the *KTH email addresses* of the students in the project team. For example, "TP CA student1@kth.se student2@kth.se".
- The CN for the server certificate should be the string "TP Server" followed by the *KTH email addresses* of the students in the project team. For example, "TP Server student1@kth.se student2@kth.se".
- The CN for the client certificate should be the string "TP Client" followed by the *KTH email addresses* of the students in the project team. For example, "TP Client student1@kth.se student2@kth.se".

The full paths to the identity certificate, private key and the CA root certificate need to be passed to the TP program as arguments, which is taken care of in the current run configurations. In order to comply with the run configurations, name and place each file as shown below. All the files have to be in PEM format.

```
---------In both the server and the client VM----------
/home/cdev/SSLCerts/CA/rootCA.pem
---------In the server VM----------
/home/cdev/SSLCerts/srv.pem
/home/cdev/SSLCerts/srv.key
---------In the client VM----------
/home/cdev/SSLCerts/cli.pem
/home/cdev/SSLCerts/cli.key
```

[3]You can use the template file /usr/lib/ssl/openssl.cnf

5 Main Steps

This is a short summary of the main steps in carrying out this assignment.

1. Install VirtualBox from Oracle on your computer.
2. Import the course VM image, and create two virtual machines, one for the client and one for the server.
3. Install an X server environment on your computer. Verify that you can login with SSH with X forwarding enabled on a virtual machine, and start Eclipse.
4. Create the tunnel by starting the TP client and server in their respective VMs, and verify that you can send packets between them over the tunnel.

When you have come this far, you have the project environment up an running. After the first workshop, all student groups should have reached this point. The next steps are about adding security to TP.

5. Create the three certificates: CA, client, and server.
6. Modify *sslUtils.cpp* to establish SSH contexts for the client and server, and to do certificate-based authentication.
7. Verify the certificate-based authentication.
8. Modify *sslUtils.cpp* so that the server generates a key and an initialization vector for symmetric encryption, which are passed to the client over the SSL connection.
9. Modify *sslUtils.cpp* so that the communication over UDP is encrypted and decrypted using the key generated in the previous step.
10. Verify that the communication over UDP is indeed encrypted.

6 Verification

6.1 Basic Functionality

First establish the tunnel by launching the TP program in both the server and the client VMs. Next, from the VPN client VM, start pinging the IP address assigned to the **TP-tun0 interface** of the server. Note that the tunnel interfaces will not have an IP address assigned unless the tunnel is established. Getting replies to the pings is the first step of verification. Also follow this step before starting any development to verify the IP connectivity of your deployment.

6.2 Certificates

First check that the certificate-based authentication works with the three certificates you have generated. Then try with invalid certificate data, and make sure that the client and server refuse to establish the connection. For instance, generate a certificate which is not signed by the CA, and check what happens if you use this as the client certificate, or as the server certificate. If you are using OpenSSL correctly, the certificate check should fail. You could also generate a certificate that is signed by the CA but has an invalid CN string. This certificate should also be rejected.

6.3 Secure UDP channel

Next, you need to make sure that the ping packets are not transferred in plaintext. To verify, launch Wireshark at your server VM via another SSH session by typing in *sudo wireshark*. If X11 forwarding components are properly set, Wireshark GUI should show up on your host machine. Then start capturing packets on the *enp0s3* interface of the server and filter out the IP traffic occurring between the client and the server VMs while the pinging is still in progress. Right-click one of the capture lines, click Decode-as, then choose IPv4 in the last column. This will make Wireshark interpret the payload of these UDP packets as IPv4 datagrams. If you see that the UDP packets are decoded as ICMP echo requests and responses, that means the tunnel is **not** secured and readable by any malicious interceptor.

7 Submission and Grading

You are expected to deliver the following *in a single ZIP archive file*:

- Your sslUtils.cpp file with detailed inline comments.
- CA, server and client certificates along with their private keys.

Please see the respective module in Canvas for submission module and the deadline.

History

Version	Comment	Author	Date
1.0	Initial version	Hüseyin Kayahan and Peter Sjödin	Dec 4, 2017