Virtual Private Network (VPN) Lab

Final Submission Due: March 31 by 11:59 PM

1 Overview

A Virtual Private Network (VPN) is used for creating a private scope of computer communications or providing a secure extension of a private network into an insecure network such as the Internet. VPN is a widely used security technology and can be built upon IPSec or Secure Socket Layer (SSL). These are two fundamentally different approaches for building VPNs. This lab focuses on the SSL-based VPNs, often referred to as SSL VPNs.

The learning objective of this lab is for students to master the network and security technologies underlying SSL VPNs. The design and implementation of SSL VPNs exemplify a number of security principles and technologies, including crypto, integrity, authentication, key management, key exchange, and Public-Key Infrastructure (PKI). To achieve this goal, students will implement a simple SSL VPN for `Lubuntu`.

2 Lab Tasks

In this lab, you will implement a simple VPN for `Linux` called `MiniVPN`.

2.1 Task 1: Create a Host-to-Host Tunnel using TUN/TAP

The enabling technology for the TLS/SSL VPNs is TUN/TAP, which is now widely implemented in modern operating systems. TUN and TAP are virtual network kernel drivers; they implement network devices that are supported entirely in software. TAP (as in network tap) simulates an Ethernet device and it operates with layer-2 packets such as Ethernet frames; TUN (as in network TUNnel) simulates a network layer device and it operates with layer-3 packets such as IP packets. With TUN/TAP, you can create virtual network interfaces.

A user-space program is usually attached to the TUN/TAP virtual network interface. Packets sent by an operating system via a TUN/TAP network interface are delivered to the user-space program. On the other hand, packets sent by the program via a TUN/TAP network interface are injected into the operating system network stack; to the operating system, it appears that the packets came from an external source through the virtual network interface.

When a program is attached to a TUN/TAP interface, the IP packets that the computer sends to this interface will be piped into the program; on the other hand, the IP packets that the program sends to the interface will be piped into the computer, as if they came from the outside through this virtual network interface. The program can use the standard `read()` and `write()` system calls to receive packets from or send packets to the virtual interface.

Davide Brini has written a tutorial article on how to use TUN/TAP to create a tunnel between two machines. The URL of the tutorial is:

`http://backreference.org/2010/03/26/tuntap-interface-tutorial`

The tutorial provides a program called `simpletun`, which connects two computers using the TUN tunneling technique. Read this tutorial and familiarize yourself with the setup instructions.

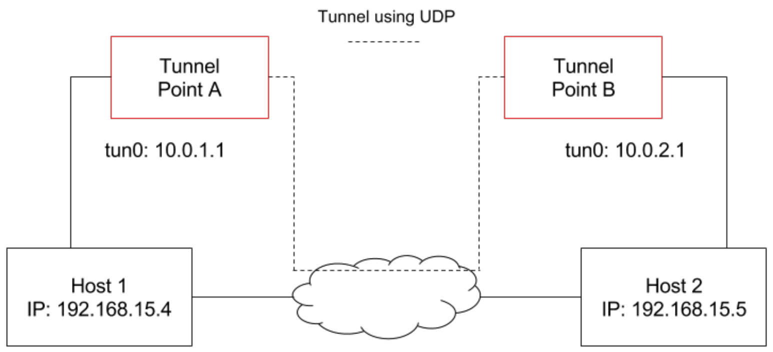

Figure 1: Host-to-Host Tunnel

A version of `simpletun` that has been modified for this lab has been provided on the resources section of piazza. Once the file is downloaded/copied into your virtual machine, compile it using the following command:

```
$ gcc -o simpletun simpletun.c
```

Creating Host-to-Host Tunnel. The following procedure shows how to create a host-to-host tunnel using the `simpletun` program. The `simpletun` program can run as both a client and a server. When it is running with the `-s` flag, it acts as a server; when it is running with the `-c` flag, it acts as a client.

1. **Launch two virtual machines.** For this task, you will launch two virtual machines on the same host machine. The following assumes the IP addresses for the two machines are `192.168.15.4`, and `192.168.15.5`, respectively. See the configuration in Figure 1. Make sure you have your virtual machines set up correctly on the network and have the port forwarding rule set.

2. **Tunnel Point A:** use Tunnel Point A as the server side of the tunnel. Point A is on machine `192.168.15.4` (see Figure 1). It should be noted that the client/server concept is only meaningful when establishing the connection between the two ends. Once the tunnel is established, there is no difference between client and server; they are simply two ends of a tunnel. Run the following command (the `-d` flag asks the program to print out the debugging information):

   ```
   On Machine 192.168.15.4:
   # sudo ./simpletun -i tun0 -s -d
   ```

   After the above step, your virtual machine will now have multiple network interfaces, one is its own Ethernet card interface, and the other is the virtual network interface called `tun0`. This new interface is not yet configured, so you need to configure it by assigning an IP address.

   It should be noted that the above command will block and wait for connections, so, we need to find another window to configure the `tun0` interface. To do this open up another SSH session to the virtual machine. Run the following commands (the first command will assign an IP address to the interface `"tun0"`, and the second command will bring up the interface):

   ```
   On Machine 192.168.15.4:
   # sudo ip addr add 10.0.1.1/24 dev tun0
   ```

```
# sudo ifconfig tun0 up
```

3. **Tunnel Point B:** you use Tunnel Point B as the client side of the tunnel. Point B is on machine `192.168.15.5` (see Figure 1). You run the following command on this machine (The first command will connect to the server program running on `192.168.15.4`, which is the machine that runs the Tunnel Point A. This command will block as well, so you need to find another window for the second and the third commands):

```
On Machine 192.168.15.5:
# sudo ./simpletun -i tun0 -c 192.168.15.4 -d
# sudo ip addr add 10.0.2.1/24 dev tun0
# sudo ifconfig tun0 up
```

4. **Routing Path:** After the above two steps, the tunnel will be established. Before you can use the tunnel, you need to set up the routing path on both machines to direct the intended outgoing traffic through the tunnel. The following routing table entry directs all the packets to the `10.0.2.0/24` network (`10.0.1.0/24` network for the second command) through the interface `tun0`, from where the packet will be hauled through the tunnel.

```
On Machine 192.168.15.4:
# sudo route add -net 10.0.2.0 netmask 255.255.255.0 dev tun0

On Machine 192.168.15.5:
# sudo route add -net 10.0.1.0 netmask 255.255.255.0 dev tun0
```

5. **Using the Tunnel:** Now you can access `10.0.2.1` from `192.168.15.4` (and similarly access `10.0.1.1` from `192.168.15.5`). You can test the tunnel using `ping` and `ssh`:

```
On Machine 192.168.15.4:
$ ping 10.0.2.1
$ ssh 10.0.2.1

On Machine 192.168.15.5:
$ ping 10.0.1.1
$ ssh 10.0.1.1
```

UDP Tunnel: The connection used in the `simpletun` program is a TCP connection, but our VPN tunnel needs to use UDP. Therefore you need to modify `simpletun` and turn the TCP tunnel into a UDP tunnel. **Question 2.1:** Think about why it is better to use UDP in the tunnel, instead of TCP write your answer in your lab report.

If the connection works, then you know you've configured your tunnel correctly. The tunnel is not secure since `simpletun` sends network traffic down the tunnel without encryption or authentication. The next task adds security to the tunnel.

2.2 Task 2: Create a Virtual Private Network (VPN)

At this point, you have learned how to create a network tunnel. Using this tunnel any machine behind the `10.0.1.0/24` network can be connected by any machine in `10.0.2.0/24` network and vice-versa, where the machines `192.168.15.4` and `192.168.15.5` act as the gateways for those networks. (In this lab we skip the part of configuring actual machines to connect to the gateways so that it can completed with less number of VMs.) Now, if you can secure this tunnel, you will essentially get a VPN. This is what you are going to achieve in this task. To secure this tunnel, you need to achieve two goals, confidentiality and integrity. The confidentiality is achieved using encryption, i.e., the contents that go through the tunnel is encrypted. Modern VPN software usually supports a number of different encryption algorithms. For the `MiniVPN` in this lab, you only need to support a single encryption algorithm.

The integrity goal ensures that nobody can tamper with the traffic in the tunnel or launch a replay attack. Integrity can be achieved using various methods. In this lab, you only need to support the Message Authentication Code (MAC) method. The AES encryption algorithm and the HMAC-SHA256 algorithm are both implemented in the `OpenSSL` library. There is plenty of online documentation on how to use the `OpenSSL`'s crypto libraries. You should not try to implement your own AES encryption or HMAC-SHA256 algorithm.
Question 2.2: Think why it is not recommended to implement your own algorithm; and write that in your lab report.

Both encryption and MAC need a secret key. Although the keys can be different for encryption and MAC, for the sake of simplicity, assume that the same key is used. This key has to be agreed upon by both sides of the VPN. For this task, assume that the key is already provided. Agreeing upon the key will be implemented in the next task.

For encryption, the client and the server also need to agree upon an Initial Vector (IV). For security purpose, you should not hard-code the IV in your code. The IV should be randomly generated for each VPN tunnel. Agreeing upon the IV will also be implemented in the next task.

NOTE: Your `MiniVPN` must use a UDP connection for data transmission. A TCP connection can be used for starting the VPN connections and key exchange.

2.3 Task 3: Authentication and Key Exchange

Before a VPN is established, the VPN client must authenticate the VPN server, making sure that the server is not a fraudulent one. On the other hand, the VPN server must authenticate the client (i.e. user), making sure that the user has the permission to create such a VPN tunnel. After the authentication is done, the client and the server will agree upon a session key for the VPN tunnel. This session key is only known to the client and the server. The process of deriving this session key is called key exchange.

Step 1: Authenticating VPN Server A typical way to authenticate the server is to use public-key certificates. The VPN server needs to first get a public-key certificate from a Certificate Authority (CA), such as Verisign. When the client makes the connection to the VPN server, the server will use the certificate to prove it is the intended server. The `HTTPS` protocol in the Web uses a similar way to authenticate web servers, ensuring that you are talking to an intended web server, not a fake one.

In this lab, `MiniVPN` should use such a method to authenticate the VPN server. You can implement an authentication protocol (such as SSL) from scratch, using the crypto libraries in `OpenSSL` to verify certificates or you can use the `OpenSSL`'s SSL functions to directly make an SSL connection between the client and the server. When using `OpenSSL`'s functions the verification of certificates will be automatically carried out by the SSL functions. Guidelines on making such a connection can be found in the guidelines section.

Before the authentication, both client and server need to set up their public/private keys and certificates properly, so they can verify each other's public-key certificate.

Step 2: Authenticating VPN Client (i.e. User) There are two common ways to authenticate the user. One is using the public-key certificates, namely, users need to get their own public-key certificates. When they try to create a VPN with the server, they need to send their certificates to the server, which will verify whether they have permissions for such a VPN. `OpenSSL`'s SSL functions also support this option if you specify that the client authentication is required.

Since users usually do not have their public-key certificates, a more common way to authenticate users is to use the traditional user name and password approach. Namely, after the client and the server have established a secure TCP connection between themselves, the server can ask the client to type the user name and the password, and the server then decides whether to allow the user to proceed depending on whether the user name and password matches with the information in the server's user database.

In this lab, you can pick either of them to implement.

Step 3: Key Exchange If you use `OpenSSL`'s SSL functions, after the authentication, a secure channel will be automatically established (by the `OpenSSL` functions). However, we are not going to use this TCP connection for our tunnel, because our VPN tunnel uses UDP. Therefore, we will treat this TCP connection as the control channel between the client and the server. Over this control channel, the client and the server will agree upon a session key for the data channel (i.e. the VPN tunnel). They can also use the control channel for other functionalities, such as updating the session key, exchanging the Initial Vector (IV), terminating the VPN tunnel, etc.

At the end of this step, you should be able to use the session key to secure the tunnel. In other words, you should be able to test Task 4 and Task 5 together.

Step 4: Securing the Tunnel Using the session key, the client and the server can secure the VPN tunnel between them. Two objectives need to be accomplished. First, the data going through the tunnel must be encrypted, so no eavesdropper can learn the data in the tunnel. Second, the integrity of the data in the tunnel must be preserved. If anybody has tampered with the data, the receiver can detect that, and can thus discard the data. These two objectives can be achieved using the symmetric-key encryption and one-way hash algorithms. An actual VPN product is able to support a variety of these algorithms. In this lab, you only need to support one encryption algorithm (with one specific encryption mode) and one one-way hash algorithm. The choice of which algorithm to use is up to you. Please make sure you document which algorithm you use and discuss why you chose to use it (giving advantages and potential disadvantages) in your analysis report.

NOTE: As part of securing the channel, not only will you need to handle confidentiality (via encryption) and integrity (via HMAC), you will need to be able to protect against man-in-the-middle attacks. Consider a malicious user in the middle between each end of your VPN. What security flaws might they be able to exploit and how do you prevent against them, for example listening to key exchange and authentication, performing replay attacks, etc. Your VPN must be able to handle these types of attacks. Make sure you discuss, in your analysis report, what attacks your VPN prevents and what you did to prevent them.

Step 5: Break the Tunnel Break the VPN tunnel from the client side. The server needs to be informed, so it can release the corresponding resources. The same should happen when the connection is broken from the client side.
Question 2.3: Why is it important for the server to release resources when a connection is broken?

2.4 Task 6: Supporting Multiple VPN Tunnels

In the real world, one VPN server often supports multiple VPN tunnels. Namely, the VPN server allows more than one client to connect to it simultaneously; each client has its own VPN tunnel with the server, and the session keys used in different tunnels should be different. For this task your VPN server should be able to support multiple clients. You cannot assume that there is only one tunnel and one session key.

When a packet arrives at the VPN server through a VPN tunnel, the server needs to figure out from which VPN tunnel the packet came from. Without this information, the server cannot know which decryption key (and IV) should be used to decrypt the packet; using a wrong key is going to cause the packet to be dropped, because the HMAC will not match. Take a look at the IPSec protocol and think about how IPSec can support multiple tunnels and use a similar idea.

(BONUS) Dynamic Reconfiguration and Additional Features For additional security (and bonus points on the lab) you can implement some commands at the client side, for example:

- Change the session key on the client end, and inform the server to make the similar change.
- Change the IV on the client end, and inform the server to make the similar change.

For additional bonus, you are encouraged to implement other features for your `MiniVPN`. In your analysis report explain what features you added as well as why you chose to add them (include advantages and disadvantages of the features). Remember that while adding additional features to the VPN can improve usability, whatever features you add to your implementation, you need to ensure that the security is not compromised.

2.5 Grading

- End-to-end encrypted UDP VPN (140 points)
- Multiple VPN connections (25 points)
- Bonus Features (10 points)
- Writing quality (15 points)
- Answers to the three questions (15 points, 5×3)

3 Guidelines

3.1 Create certificates

In order to use `OpenSSL` to create certificates, you have to have a configuration file. The configuration file usually has an extension `.cnf`. It is used by three `OpenSSL` commands: `ca`, `req` and `x509`. The manual page of it can be found using a Google Search. You can also get a copy of the configuration file from `/usr/lib/ssl/openssl.cnf`. After copying this file into your current directory, you need to create several sub-directories as specified in the configuration file (look at the `[CA_default]` section):

```
dir             = ./demoCA          # Where everything is kept
certs           = $dir/certs        # Where the issued certs are kept
crl_dir         = $dir/crl          # Where the issued crl are kept
new_certs_dir   = $dir/newcerts     # default place for new certs.
```

```
database         = $dir/index.txt    # database index file.
serial           = $dir/serial       # The current serial number
```

For the `index.txt` file, simply create an empty file. For the `serial` file, put a single number in string format (e.g. 1000) in the file. Once you have set up the configuration file `openssl.cnf`, you can create certificates for the three parties involved, the Certificate Authority (CA), the server, and the client.

Certificate Authority (CA). For this lab, you are allowed to create your own CA, and then you can use this CA to issue certificates for servers and users. You will create a self-signed certificate for the CA. This means that this CA is totally trusted, and its certificate will serve as the root certificate. You can run the following command to generate the self-signed certificate for the CA:

```
$ openssl req -new -x509 -keyout ca.key -out ca.crt -config openssl.cnf
```

You will be prompted for information and a password. Do not lose this password, because you will have to type the passphrase each time you want to use this CA to sign another certificate. You will also be asked to fill in some information, such as the Country Name, Common Name, etc. The output of the command are stored in two files: `ca.key` and `ca.crt`. The file `ca.key` contains the CA's private key, while `ca.crt` contains the public-key certificate.

Server. Now that you have your own trusted CA, you can now ask the CA to issue a public-key certificate for the server. First, you need to create a public/private key pair for the server. The server should run the following command to generate an RSA key pair (both private and public keys). You will also be required to provide a password to protect the keys. The keys will be stored in the file `server.key`:

```
$ openssl genrsa -des3 -out server.key 1024
```

Once you have the key file, you can generates a Certificate Signing Request (CSR). The CSR will be sent to the CA, who will generate a certificate for the key (usually after ensuring that identity information in the CSR matches with the server's true identity).

```
$ openssl req -new -key server.key -out server.csr -config openssl.cnf
```

Client. The client can follow the same steps to generate an RSA key pair and a certificate signing request:

```
$ openssl genrsa -des3 -out client.key 1024
$ openssl req -new -key client.key -out client.csr -config openssl.cnf
```

Generating Certificates. The CSR file needs to have the CA's signature to form a certificate. In the real world, the CSR files are usually sent to a trusted CA for their signature. In this lab, you will use your own trusted CA to generate certificates:

```
$ openssl ca -in server.csr -out server.crt -cert ca.crt -keyfile ca.key \
             -config openssl.cnf
$ openssl ca -in client.csr -out client.crt -cert ca.crt -keyfile ca.key \
             -config openssl.cnf
```

If `OpenSSL` refuses to generate certificates, it is very likely that the names in your requests do not match with those of the CA. The matching rules are specified in the configuration file (look at the `[policy_match]` section). You can change the names of your requests to comply with the policy, or you can change the policy. The configuration file also includes another policy (called `policy_anything`), which is less restrictive. You can choose that policy by changing the following line:

```
"policy = policy_match"
```

To the line:

```
"policy = policy_anything"
```

3.2 Create a secure TCP connection using `OpenSSL`

In this lab, students need to know how to use `OpenSSL` APIs to establish a secure TCP connection. The following tutorial might be useful to you for this lab.

- `http://www.ibm.com/developerworks/linux/library/l-openssl.html`

3.3 Miscellaneous notes

Our client (or server) program is going to listen to both TCP and UDP ports, these two activities may block each other. It is better if you can `fork()` two processes, one dealing with the TCP connection, and the other dealing with UDP. These processes need to be able to communicate with each other. You can use the Inter-process call (IPC) mechanisms for the communication. The simplest IPC mechanism is unnamed pipe, which should be sufficient for us. You can learn IPC from online documents.

3.4 External libraries

Since the image is old `apt-get` commands will not work out of the box. We have provided a `sources.list` file. Replace the `/etc/apt/sources.list` with the provided file to get `apt-get` working.

4 Submission

Total points earnable: 205 points

4.1 What to Submit

Your submission directory should contain:

- All source files written by you to perform the lab tasks.
- A README text file describing how to compile and execute your source code.
- **Report (Writing quality - 15points)**: A detailed report containing an explanation of the observations. Name this report *Analysis-lab3.pdf*. Be sure to put your name in your report.

4.2 How to submit

Submit every time you have a stable version of any part of the functionality. Make sure to submit early to make sure you do not miss the deadline due to any last minute congestion.

1. Login or ssh to your mcXX machine, e.g., mc19.cs.purdue.edu
2. In the parent directory of your submission directory, type the command:

   turnin -c cs528 -p lab3 <submission-dir-name>

   where <submission-dir-name> is the name of the directory containing your files to be submitted. For example, if your program is in a directory `/homes/abc/assignment/src`, make sure you cd to the directory `/homes/abc/assignment` and type:

   turnin -c cs528 -p lab3 src
3. If you wish to, you can verify your submission by typing the following command:

   turnin -v -c cs528 -p lab3

   Do not forget the **-v** above, as otherwise your earlier submission will be erased (it is overwritten by a blank submission).

   Note that resubmitting overwrites any earlier submission and erases any record of the date/time of any such earlier submission.

   NOTE: Do not submit your virtual machine disk image. Copy the files for submission off of your virtual machine and into a directory on the mcXX machine.