CS528 – lab2

PUID:0036728672

Name: Vijay Anirudh Aithi

First of All we need to change the simpletun's TCP config to UDP config:

For this follow these steps:

1) TCP data will be in Stream format, Change it into Datagram format.

```
if ( (sock_fd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
  perror("socket()");
  exit(1);
}
```

2) In TCP Implementation there will be connect, accept, listen functions.
3) We will implement Sendto() and recvfrm() in UDP because, UDP is connection less.

```
if((nread =
    recvfrom(sock_fd,buffer,BUFSIZE,0,(struct sockaddr*)&local,&local_len)) <= 0) {
        perror("Error in Receiving data\n");
        exit(1);

    }
```

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.

Answer:

TCP is a "stateful" protocol that utilizes error correction when sending data through the tunnel. UDP, also known as the "stateless" protocol, does not utilize the error correction concept and works on the number of retries.

This makes UDP faster when compared to TCP. A small noise or interception in TCP over a TCP connection would produce high latency as it would proceed to correct the dropped packets. Therefore, UDP is better when compared to TCP.

Next part we will try to secure this tunnel i.e. VPN:

1) We need to check Integrity, Confidentiality, Availability
2) For Integrity we will implement Client – Server Authentication using open ssl/tls:

   Client Side:

```
if (SSL_CTX_use_certificate_file(ctx, "./certs/client1.crt", SSL_FILETYPE_PEM) <= 0)
{
    ERR_print_errors_fp(stderr);
    exit(1);
}
if (SSL_CTX_use_PrivateKey_file(ctx, "./certs/client1.key", SSL_FILETYPE_PEM) <= 0)
{
    ERR_print_errors_fp(stderr);
    exit(1);
}

if (!SSL_CTX_check_private_key(ctx))
{
    fprintf(stderr, "Keys do not match. Please check the certificates.\n");
    exit(1);
}
else{
  printf("SSL Handshake successful.");
}
```

   Server Side:

```
if (SSL_CTX_use_certificate_file(ctx, "./certs/server1.crt", SSL_FILETYPE_PEM) <= 0)
    {
        ERR_print_errors_fp(stderr);
        exit(1);
    }
    if (SSL_CTX_use_PrivateKey_file(ctx, "./certs/server1.key", SSL_FILETYPE_PEM) <= 0)
    {
        ERR_print_errors_fp(stderr);
        exit(1);
    }

    if (!SSL_CTX_check_private_key(ctx))
    {
        fprintf(stderr, "Private key does not match the certificate public key\n");
        exit(1);
    }
```

After Authentication we will perform Key-exchange, encryption, HMAC, and Hashing

Rest is same as simpleton UDP configuration sending data and receiving data.

```c
/*****************************************************************
 * usage: Function for Encryption.                               *
 *****************************************************************/
int encryptaes(unsigned char *key, unsigned char *iv, char *buffer,int *length, int option){

    //Declarations
    int otlen = 0, templen = 0;
    int inlen = *length;
    unsigned char inputbuff[BUFSIZE];
    unsigned char outputbuff[BUFSIZE + EVP_MAX_BLOCK_LENGTH];


    //Buffers
    memcpy(inputbuff,buffer,inlen);
    EVP_CIPHER_CTX ctx;
    EVP_CIPHER_CTX_init(&ctx);
    int z = strlen(key);

    //Encryption
    EVP_CipherInit_ex(&ctx,EVP_aes_128_cbc(),NULL,key,iv,option);
    if(!EVP_CipherUpdate(&ctx,outputbuff,&otlen,inputbuff,inlen))
        return 0;
    if(!EVP_CipherFinal_ex(&ctx,outputbuff+otlen,&templen))
        return 0;
    otlen+=templen;
    EVP_CIPHER_CTX_cleanup(&ctx);

    memcpy(buffer,outputbuff,otlen);
    *length = otlen;
    return 1;
}
```

```c
/*****************************************************************
 * usage: Function for Obtain Hashing.                           *
 *****************************************************************/
void GetHash(unsigned char *key,unsigned char *buffer,int length,char *hash)
{
    HMAC_CTX msgdctx;
    unsigned char *outputhash = (char*)malloc(HMAC_LENGTH);
    int msgd_len;

    HMAC_CTX_init(&msgdctx);
    HMAC_Init_ex(&msgdctx,key,strlen(key),EVP_sha256(),NULL);
    HMAC_Update(&msgdctx,buffer,length);
    HMAC_Final(&msgdctx,outputhash,&msgd_len);
    HMAC_CTX_cleanup(&msgdctx);

    memcpy(hash,outputhash,HMAC_LENGTH);
}

/*****************************************************************
 * usage: Function for Perform Hashing.                          *
 *****************************************************************/
void PFHMAC(unsigned char *key,unsigned char *buffer,int *length)
{
    char hash[HMAC_LENGTH],inputbuff[BUFSIZE];
    int i=0,inlen=*length;
    memcpy(inputbuff,buffer,inlen);
    GetHash(key,inputbuff,inlen,hash);

    //Appending MAC to the Message

    for(i=0;i<HMAC_LENGTH;i++)
        *(buffer+inlen+i) = hash[i];
    inlen += HMAC_LENGTH;
    *length = inlen;
}
```

Question 2.2: Think why it is not recommended to implement your algorithm; and write that in your lab report.

Answer:

One of the key features of OpenSSL is that encryption and decryption of packets occur inside the VM/systems which makes it safe from MITM/Interception attacks. Let us assume we use our own AES or SHA-256 algorithms in our VPN implementation, we can face issues if we don't cover all possible test vectors that are supposed to be verified during decryption. The server/client will not be able to tell if the data has been altered and they will blindly decrypt the data. This is exactly the vulnerability that OpenSSL tries to avoid and therefore, it is recommended to use OpenSSL libraries.

QUESTION 2.3:

Why is it important for the server to release resources when a connection is broken?

ANSWER:

Servers are often equipped with limited resources such as file descriptors, memory, and PU cycles. If we think about scalability when released it ensures that all those connections do not tie with resources. Resource leaks would be a problem too, this might affect the system's security.

This is Client authentication:

```
[sudo] password for cs528user:
Successfully connected to interface tun0
Enter PEM pass phrase:
fscanf: Success
1
SERVER: Client authenticated.
```

SSH is Successful:

```
64 bytes from 10.0.1.1: icmp_req=6 ttl=64 time=1.73 ms
64 bytes from 10.0.1.1: icmp_req=7 ttl=64 time=2.36 ms
64 bytes from 10.0.1.1: icmp_req=8 ttl=64 time=1.50 ms
^C
--- 10.0.1.1 ping statistics ---
8 packets transmitted, 6 received, 25% packet loss, time 7049ms
rtt min/avg/max/mdev = 1.502/1.925/2.694/0.450 ms
[03/04/2024 18:44] cs528user@cs528vm:~/Desktop/minivpn/cs528lab2$ ssh 10.0.1.
1
cs528user@10.0.1.1's password:
Welcome to Ubuntu 12.04.2 LTS (GNU/Linux 3.5.0-37-generic i686)

 * Documentation:  https://help.ubuntu.com/

New release '14.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Mon Mar  4 17:37:21 2024 from 10.0.2.1
[03/04/2024 18:51] cs528user@cs528vm:~$
```