CS631
Assignment 3

Gajender Sharma

October 2021

1 Need of Encryption in SCADA Protocols

SCADA MODBUS is an application layer messaging protocol, positioned at level 7 of the OSI model. It provides client/server communication between devices connected to different types of buses or networks. It does not implement security mechanism as Modbus is a very old protocol with 30+ years of existence used to exchange messages between electronic devices when there was no concern of security so no security measures were taken as a necessity but time has changed alot since then every year we hear Cyber attacks on ICS devices on national scale so it has become necessary to implement security/encryption functionality to such protocols with help of many IT devices and mechanism.The easiest attack to use against Modbus is to simply sniff the traffic on a network, find the Modbus devices and then issue harmful commands to the Modbus devices.

Modbus/TCP has no security or encryption features, so it is easy to use Wireshark to gather information from packets of data that are on your network to and from a Modbus port on a device.

Also any evil entity can see the data of modbus message clearly in textual format as there is no encryption in modbus and can gain access to modbus device,So now in this assignment I have implemented security with the help of modbus proxy and RSA cryptography algorithm to encrypt messages between Modbus Client and Modbus server.

2 Types of Encryption

2.1 Symmetric Key Encryption

Symmetric encryption is a type of encryption where only one key (a secret key) is used to both encrypt and decrypt electronic information. The entities communicating via symmetric encryption must exchange the key so that it can be used in the decryption process. This encryption method differs from asymmetric encryption where a pair of keys, one public and one private, is used to encrypt and decrypt messages.

2.2 Asymmetric Key Encryption

Asymmetric Encryption uses two distinct, yet related keys. One key, the Public Key, is used for encryption which is sent to the Client for encrypting the messages and the other, the Private Key, is for decryption which can be decrypted at the proxy side. As implied in the name, the Private Key is intended to be private so that only the authenticated recipient(Modbus Proxy) can decrypt the message.

3 How Communication is implemented in Assignment

In usual communication, Modbus Client is directly connected to Modbus Server but here we have created a bridge between client and server through a Proxy and implemented encryption and decryption with the help of modbus-proxy.

I have used both symmetric and asymmetric key encryption in assignment, asymmetric key encryption is used in Modbus proxy to share the symmetric key(encryption key) generated by the client, asymmetric key encryption generates Public and Private keys and send public key to Client for encryption and uses its Private key for decryption.Symmetric key is generated by Client which is mainly used for the encryption and decryption of modbus messages which will happen between the client and server.

Python Code which is used for encryption security in modbus

[file:modbus-proxy.py]

```
    ----KEYS EXCHANGE BETWEEN CLIENT AND MODBUS PROXY-----
publicKey, privateKey = rsa.newkeys(512)
client.writer.write(publicKey.save_pkcs1())
```

In above code , we are generating public key and private key pair from RSA cryptography algorithm where 512 is number of bits to store n in (n=p*q) and sending this public key to Client over connection established.

[file:modbus-client.py]

```
modbusproxy_public_key = rsa.PublicKey.load_pkcs1(sock.recv(1024))
EncryptionKey = Fernet.generate_key()
enc_EncryptionKey = rsa.encrypt(EncryptionKey,
    modbusproxy_public_key)
sock.sendall(enc_EncryptionKey)
fernet = Fernet(EncryptionKey)
```

1. In first line, we are receiving the modbus proxy public key and storing it.
2. second line, we are generating a symmetric key which will be mainly used for encryption.
3. third line, we are encrypting the symmetric key with public key of modbus proxy.
4. fourth line, sending this encrypted symmetric key to modbus proxy.
5. creating fernet instance of encryption key which will be used for encryption and decryption.

[file:modbus-proxy.py]

```
enc_encryption_Key= await client.reader.read(1024)
encryption_Key = rsa.decrypt(enc_encryption_Key, privateKey)
client.fernet = Fernet(encryption_Key)
```

Here we are awaiting on Client to receive the encrypted Encryption key.When you call await, the function you're in gets suspended while whatever you asked to wait on happens, and then when it's finished, the event loop will wake the function up again and resume it from the await call, passing any result out.

we are using private key of the modbus proxy to decrypt the encryption key as client encrypted the key using public key of modbus proxy and creating a fernet instance of the encryption key which is used for encryption and decryption of messages.

Lets see how the client will send the encrypted message

[file:modbus-client.py]

```
message = tcp.write_multiple_coils(slave_id=1,starting_address=1,
    values=[1, 0, 1, 0,0,1,0])
response = tcp.send_message(fernet.encrypt(message), sock)
```

Here we are writing to multiple coils with parameters **slave_id = 1** represents the number of devices connected **starting_address** represents the starting address of coil to start writing and **values** represents the quantity and values to write from starting address.

second line above, will encrypt the modbus ADU with symmetric encryption key before sending it to the modbus proxy.

[file:modbus-proxy.py]

```
async def _read(self):
    if type(self).__name__ =='Client':
        Encrypt_data = await self.reader.read(1024)
        self.log.info(" Encrypted data received from client :
        %r", Encrypt_data)
        reply = self.fernet.decrypt(Encrypt_data)
    else:
        header = await self.reader.readexactly(6)
        size = int.from_bytes(header[4:], "big")
        reply = header + await self.reader.readexactly(size)
    self.log.info("Data received from modbus device : %r",
        reply)
    return reply
```

In above **_read()** function of modbus proxy, it job is to receive the modbus message from either side of the bridge i.e client or server.Here I have changed the functionality of code to differentiate whether receiving message is from client or server as by ***if type(self).__name__ =='Client'***

If the message is received from the client, modbus proxy first has to decrypt the encrypted message with the Encryption key and then forward it to the

modbus device.

If the message is received from the modbus device, then it can simply read the ADU of modbus data packet without any decryption.

[file:modbus-proxy.py]

```
async def _write(self, data):
    if type(self).__name__ =='Client':
        self.log.info("Data which is to be sent to Client
        before encryption : %r", data)
        encrypt_data = self.fernet.encrypt(data)
        self.log.info("Data is encrypted, now sending to client
        : %r", encrypt_data)
        self.writer.write(encrypt_data)
    else:
        self.log.info("Data sending to modbus device : %r",
        data)
        self.writer.write(data)

    await self.writer.drain()
```

In above **_write()** function of modbus proxy, it job is to send the modbus message to either side of the bridge i.e client or server.Here I have changed the functionality of code to differentiate whether receiving message is from client or server as by ***if type(self).__name__ ==’Client’***

If the message is to be send to the client, modbus proxy first has to encrypt the modbus data packet with the Encryption key and then forward it to the modbus Client.

If the message is to be send to the modbus device, then it can simply write the ADU of modbus data packet without any encryption.

4 Screenshots

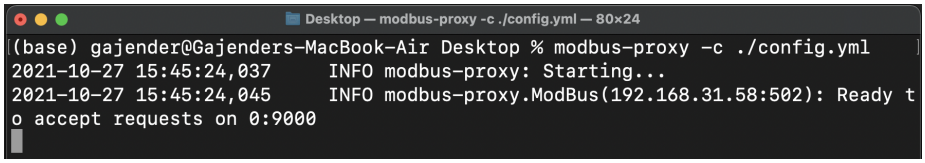

Figure 1: Modbus Proxy server listening on 0:9000 socket

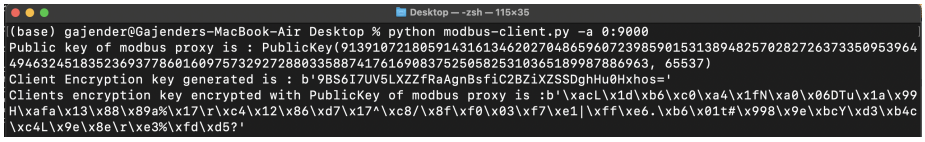

Figure 2: Keys Exchange between Client and Modbus Proxy

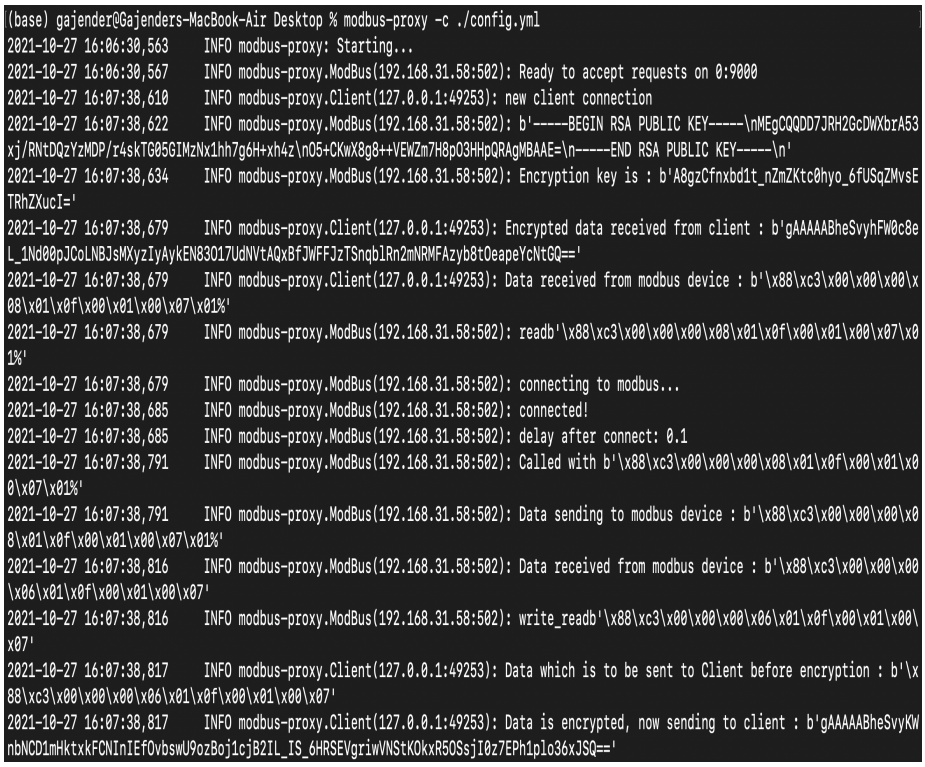

Figure 3: Logs of Bridge connection between client-modbusproxy-server

```
28  fernet = Fernet(EncryptionKey)
29
30  #creating a modbus ADU to write multiple coils with starting address 1 and values 1, 0, 1, 0,0,1,0.
31  message = tcp.write_multiple_coils(slave_id=1, starting_address=1, values=[1, 0, 1, 0,0,1,0])
32
33
34  # response recieved depends on command requested as it will return the number of coils written
35  response = tcp.send_message(fernet.encrypt(message), sock)
```

Figure 4: Requesting to write coils from client

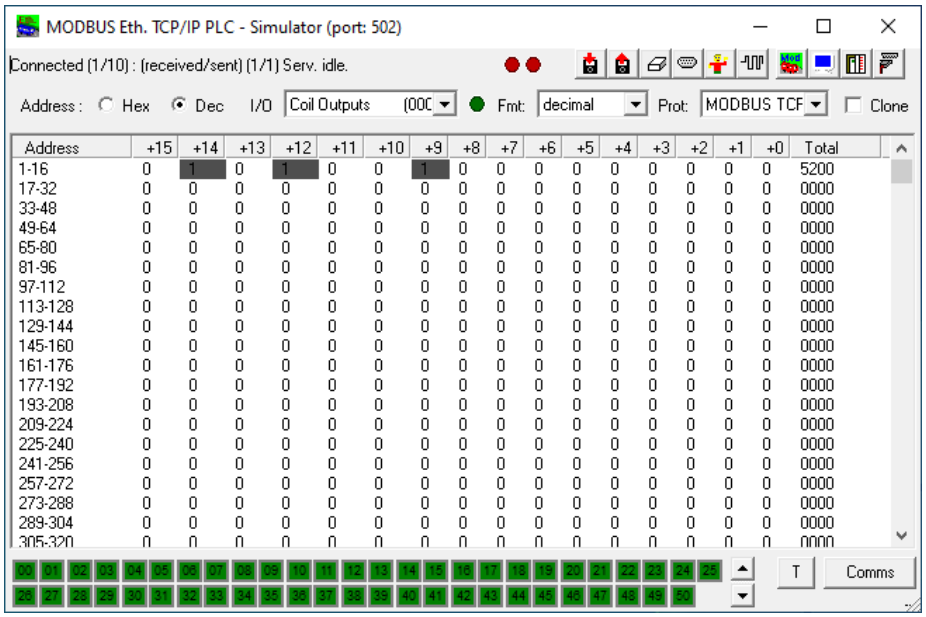

Figure 5: Writing multiple coils

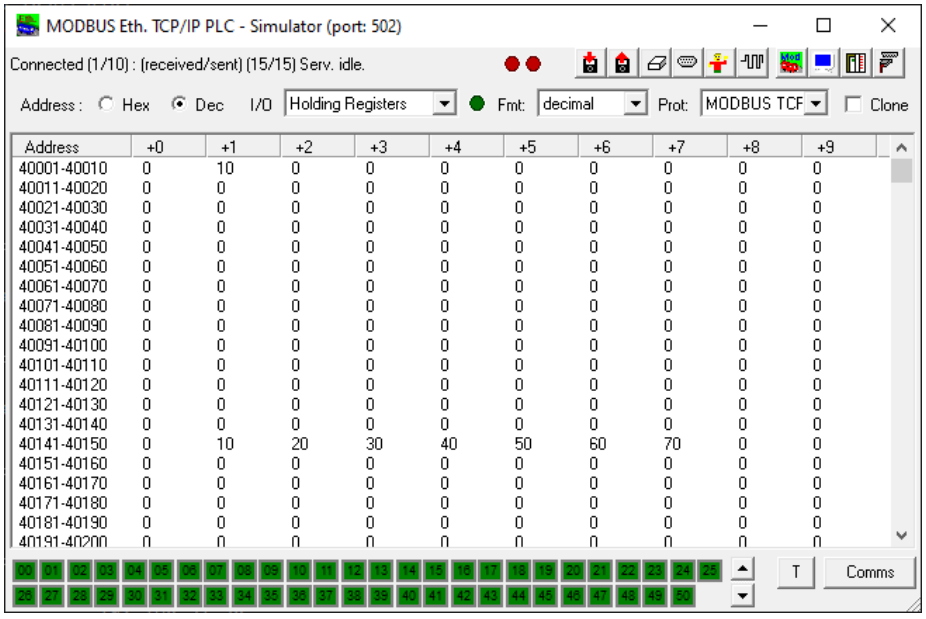

Figure 6: Writing values in Holding registers