MONASH UNIVERSITY
Caulfield campus

Assignment-1

Submitted by

Anand Prakash – 29829178

Shrey Chhaiya – 29978009

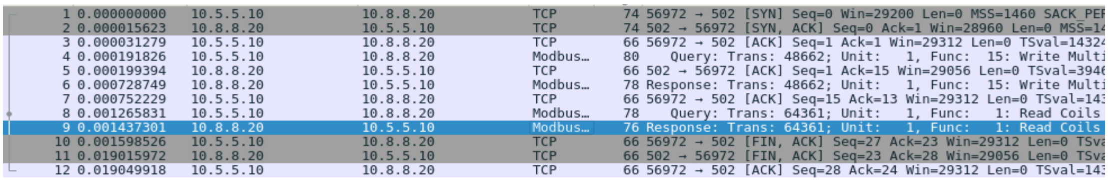
1. INTRODUCTION

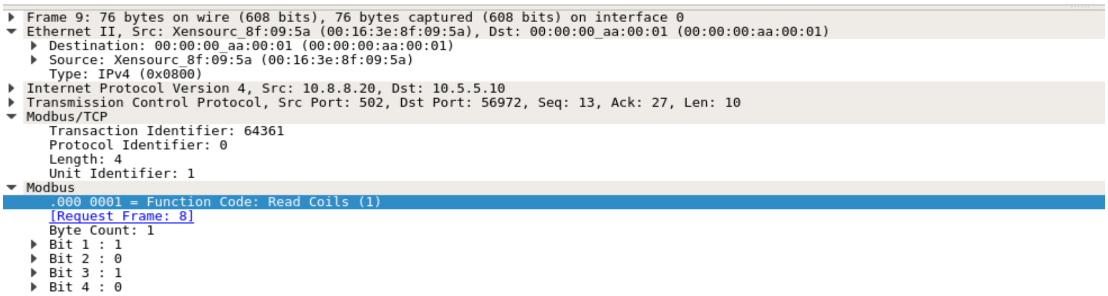
This report is about TCP communication using Modbus. The original Modbus protocol is not designed to be secure. The security is provided by using a more secure Modbus protocol. Python files are used for implementing Confidentiality, Integrity and Authentication to the original Modbus protocol. In the second part, VPN and Firewall have been implemented for a factory on a come emulator file. Containers are used in both the part. All these attempts are for creating a secure protocol for communication. (Wikimedia Foundation, 2019)

2 FIRST TASK: SECURING MODBUS PROTOCOL

2.1 SECURITY ISSUES OF MODBUS PROTOCOL

Modbus is an application layer protocol which uses the port 502 of TCP/IP stack for communication. In its security analysis, it was noticed that it transfers all the messages in clear text over the transmission media. It can be seen in the Wireshark file below that the contents of the Read Coils request are in plain text. This shows that there is lack of Confidentiality in the Modbus protocol. Also, there is no Authentication done by this protocol and there are no Integrity checks involved in its working. Modbus rely on the lower level protocols for preserving the integrity of the messages.

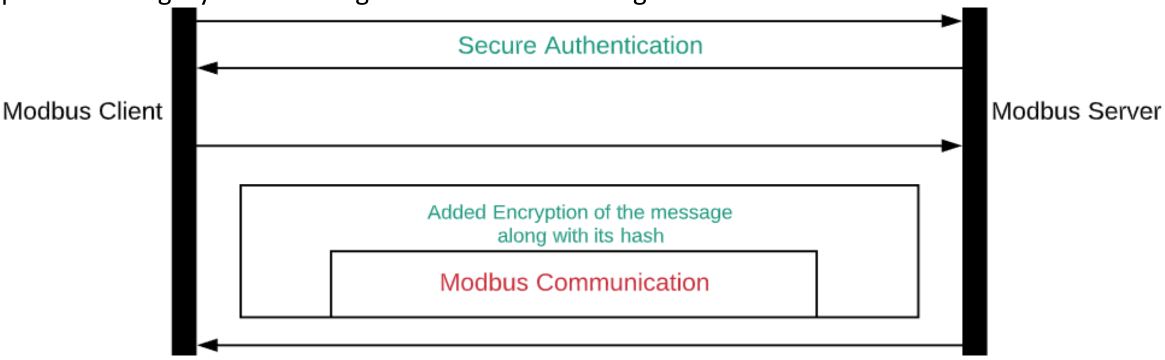




An easy attack to perform on this protocol is to sniff the traffic on the network and identify all the devices that are using Modbus protocol. After this, issue the harmful commands on those devices. Since there is no encryption on the messages sent by Modbus, it is very easy to see the contents of the packets captured by Wireshark. First, you have to look for the IP address of the BMS and the Modbus device that is receiving the packets. After this, check for the Function Code of the request. When you have all this data, identifying the Modbus device and its control command options by finding its Modbus Register Map is easy. Once these identifications are done, there is no limit to what can be done to the device. You can just start issuing commands as if you are the BMS. (Cyberbit, 2019)

2.2 SECURE VERSION OF THE MODBUS PROTOCOL

The protocol that we designed used RSA Signature and Verification from both the Modbus Client and Modbus Server side for the Authentication of both sides. The Client sends the RSA Public Key along with RSA Signature to the Server and vice-versa. The Server then verifies Client's Signature using the Client's Public Key and the stored message on the server whose hash was signed by the Client. Similarly, Client verifies Server's Signature using the Server's Public Key and the message stored on the Client device whose hash was signed by the Server. After the Authentication, the normal Modbus Communication is done but by encrypting all the messages along with their hash values before sending. Diffie–Hellman Session Key is used for the encryption and decryption of the interchanged messages. The encryption provides Confidentiality and the received hash value of the message can be compared by generating new hash value of the received message and if both of them are same, it provides Integrity. SHA-256 algorithm is used for the generation of hash values.



The secure principles that this upgraded Modbus protocol achieves are Confidentiality, Integrity and Authentication and thus enhances the security of the Modbus by not disclosing any information if its traffic is captured by Wireshark. So, after applying this enhanced protocol, any unauthorized user will not be able to read the message, if the user changes the messages then it can be identified by comparing the hash values.

2.3 IMPLEMENTATION OF THE PROTOCOL

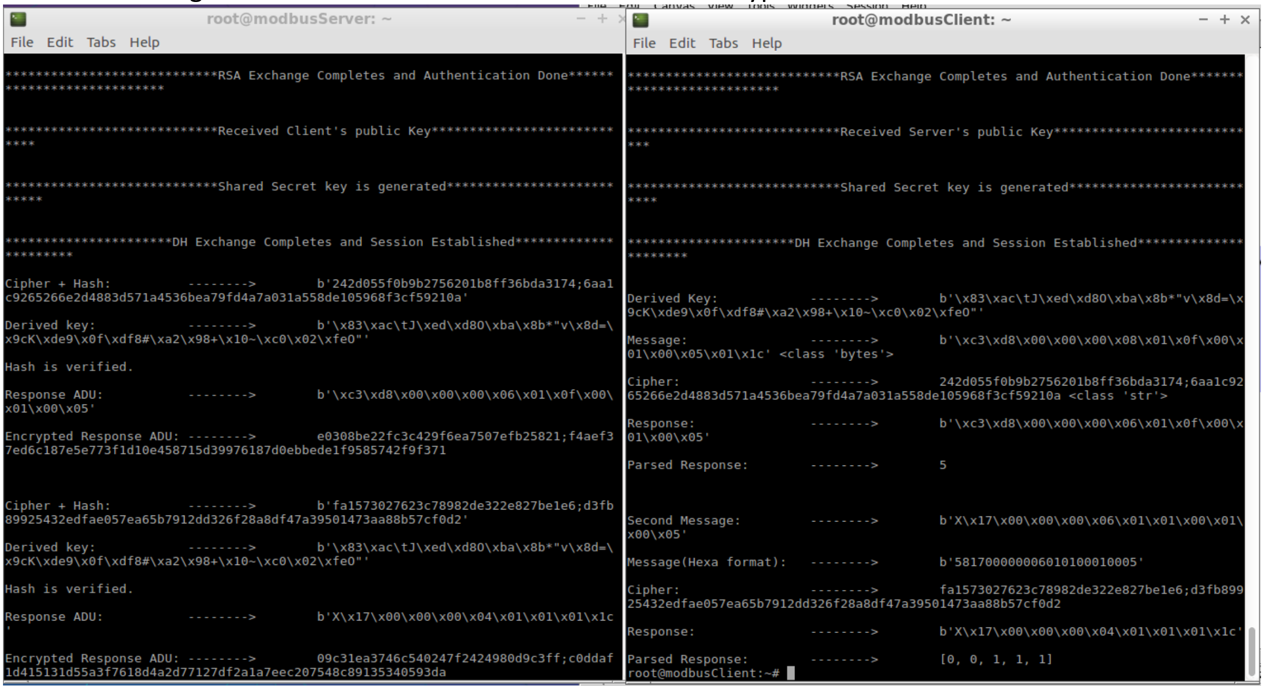
```
digest = SHA256.new()

def signData(msg, private_key):
    digest.update(msg)
    private_key = RSA.importKey(private_key)
    signer = PKCS1_v1_5.new(private_key)
    sig = signer.sign(digest)
    return b64encode(sig)

def verifySign(msg, publicKey, sig):
    publicKey = RSA.importKey(publicKey)
    digest.update(msg)
    verifier = PKCS1_v1_5.new(publicKey)
    try:
        verifier.verify(digest, b64decode(sig))
        return True
    except:
        return False

def getHash(msg):
    return str(h.sha256(msg).hexdigest())

def verifyHash(hashToVerify,hashGenerated):
   # print("verify hashToVerify: ",hashToVerify)
   # print("verify hashGenerated: ",hashGenerated)
    if hashToVerify in hashGenerated:
        return True
    else:
        print("Integrity Exception")
        return False
        raise ValidateError("Cannot Verify received messa
ge digest with the digest received")
```

```
def aes_cbc_encrypt(msg, obtained_key, iv,hashGenerated):
    #print("plain text before digest: ",message)
    leng = 16 - (len(msg) % 16)
    msg += bytes([leng])*leng
    aes = AES.new(obtained_key, AES.MODE_CBC, iv)
    enc = aes.encrypt(msg)
    return (enc.hex()+";"+hashGenerated)


def aes_decrypt(enc, obtained_key, iv):
    enc = enc.decode('utf-8')
    enc1 = enc.split(";")
    enc = enc1[0]
    digest = enc1[1]
    enc = bytes.fromhex(enc)
    dec = AES.new(obtained_key, AES.MODE_CBC, iv)
    plain_text = dec.decrypt(enc)
    plain_text = plain_text[:-plain_text[-1]]
    return (plain_text,digest)


def hkdf_function(session_key):
    salt = b'alienware'
    info = b'hello world'
    backend = default_backend()
    hkdf = HKDF(algorithm=hashes.SHA256(),length=32, salt
=salt, info=info, backend=backend)
    obtained_key=hkdf.derive(session_key)
    return obtained_key
```

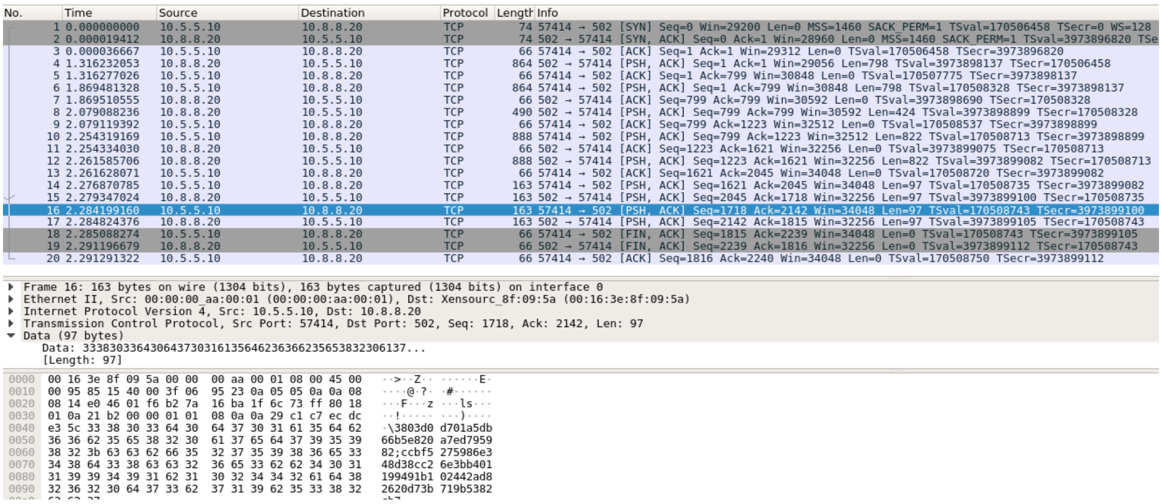
At the client side, we changed the code in our main client file but at the server side all changes are made in __init__.py file because the AbstractHandler is inside this file. So we added above functions in AbstractHandler so that handle function can use them in order to get request and send back replies. So we have created the signData and verifySign functions for generating signatures and verifying them at both the sides. The getHash function generates the hash value of a message and verifyHash function compares the newly generated hash value and the received hash value of the message for checking the integrity of the messages. The aes_cbc_encrypt function encrypts the messages along with their hash values and the aes_decrypt function separates the hash value from the cipher text and the decrypts the cipher text. The hkdf_function derives the shared key of desired size from the actual Diffie–Hellman session key (You can notice that salt value is used in this function).

In order to run our client and server communication, we need to configure eth0.cfg file to set IP address, gateway address and DNS server addresses. After configuration we ran the core emulator file to get communication between client and server so that we can verify and check our code.

For the validation, all the steps were printed in some way and as you can see below, all the messages that are exchanged between the client and the server are encrypted.



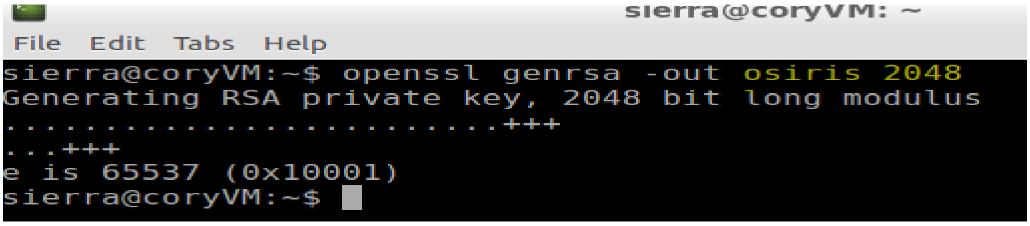
As per the previous Wireshark pcap file shown on the first page, thr protocol was insecure. An attacker can sniff the data from the Modbus protocol and read the messages. But as you can see in the wireshark pcap screenshot below, communication is now secure and all the Modbus protocol is now secure TCP protocol. So data inside the packets are encypted and can't be read by the attackers.



3 SECOND TASK: VPN & FIREWALL

3.1 IPSEC VPN

Here, IPsec VPN is used to provide security in Modbus to satisfy authentication and encryption. Modbus data sends over IP protocol. Here, IPsec VPN generates tunnel between Osiris and Bast gateway routers. (Juniper Networks, 2019)
Strongswan Configurations:
Conn node will specify left and right connections & subnets, which is basically start & end point of tunnel. Key exchange protocol = ikev2 (2nd generation) for more stable and reliable communication. We are creating tunnel and using public key authentication. For that we are generating RSA public & private key certificates and adding it into files. RSA generation is done by OpenSSL command in terminal and copied them into router's strongswan configuration files.

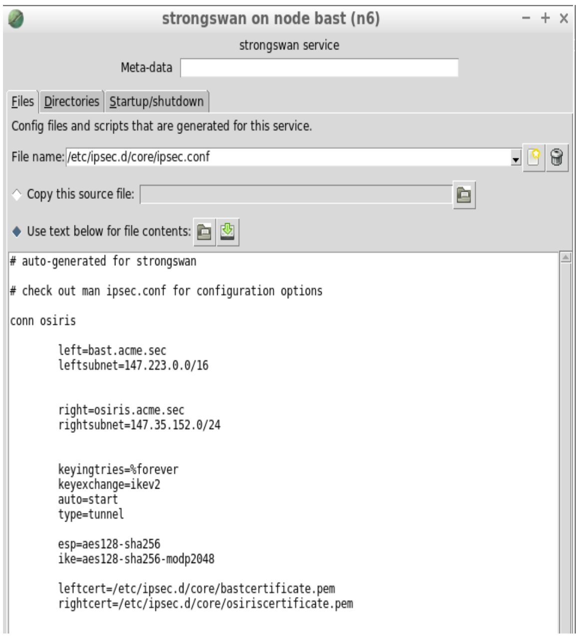


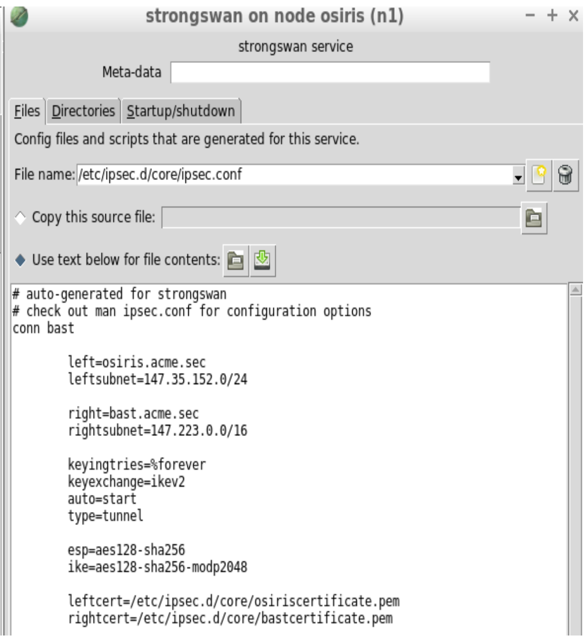
- openssl req -new -key osiris -out osiriscert
- openssl x509 -req -days 365 -in osiriscert -signkey osiris -out osiriscertificate

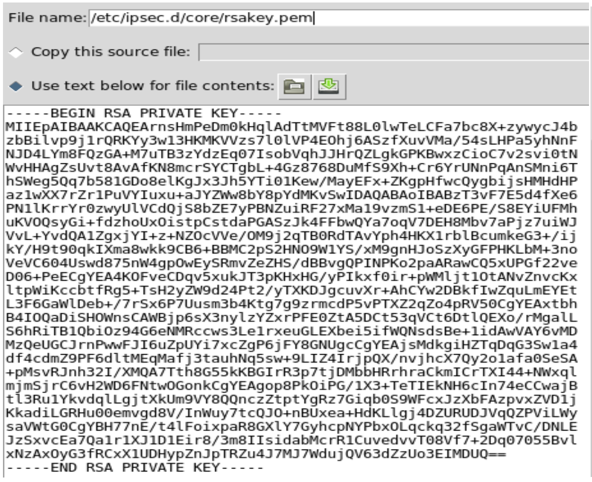
After that we specified path into ipsec.secrets file, which is our private key.

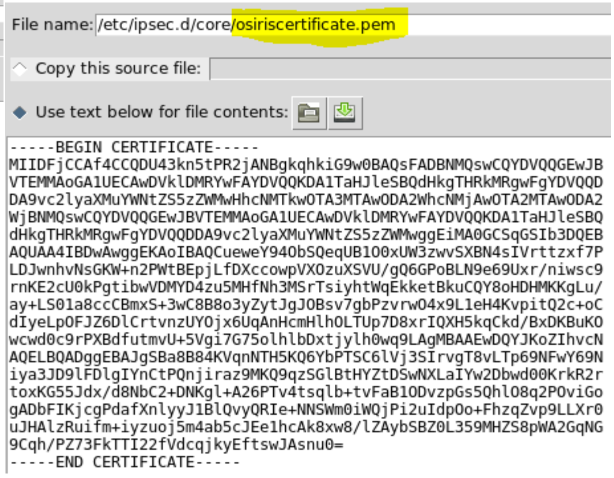
- Ipsec.secrets: ": RSA /etc/ipsec.d/core/rsakey.pem"

For, Internet Key Exchange AES-128 & HASH-256 is used for security. For Encapsulating security payload, we used AES128-SHA256-MODP2048 for authentication (RSA), data integrity (HASH) and encryption (AES) purpose. Below screenshots are showing configuration files.



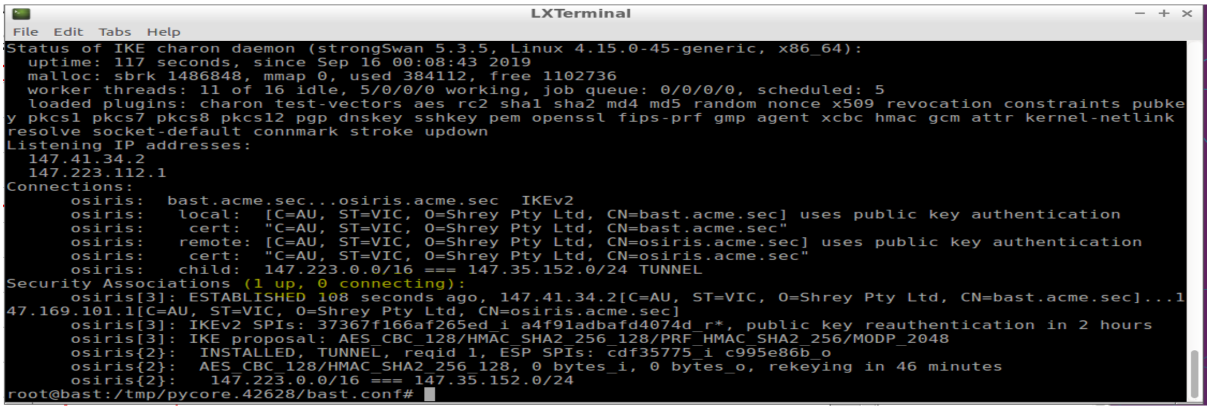






- osiriscertificate.pem, bastcertificate.pem generated and stored into strongswan.

OUTPUT



So, after configuration, we restarted IPsec connection using “IPsec restart” and checked our VPN using “IPsec statusall” command. It generated security association - 1 up with all the information regarding VPN tunnel.
After applying VPN, it solves Confidentiality, Data Integrity and Authentication which secures Modbus communication. And almost all the security issues can be satisfied using IPSEC VPNs.

3.2 Firewall Setup

Firewall is predefined security rule’s network system which controls incoming and outgoing traffic. We are using IPTABLES to set firewall rules. Modbus communication can be secured by setting up firewall rules in gateway routers to drop unauthenticated traffics. (How-To Geek, 2019)
IPTABLES:

- Iptables -F → Flush deletes all the rules which previously established.
- Iptables -P INPUT/OUTPUT/FORWARD DROP → to set default policies.
- -A → to Append rules after dropping every traffic.
- -I, -o, -s, -d → To append traffic (-Input/-output: ethernet interface) & (-source/-destination: IP addresses or networks) types to accept or remove traffic.
- -j ACCCEPT/DROP → To accept or drop traffic.

Questions & Solutions:

To connect Modbus client with Modbus server. We allowed traffic from eth0 to eth1 & from server (IP address) to client (IP address) and vice versa. Now, our client can ping to server and vice versa.

Note- As per our understanding we allowed every type of traffic to both the way. We can also specify port 502 by specifying -p tcp & -dport/-sport 502. But as we don't have checking mechanism for 502 port so we didn't specify it. But, with port 502 it's working by simply adding above mentioned line.

Any user can access acme central website: As per our understanding anyone can access website which is TCP access on port 80(HTTP). So, we only allowed incoming traffic to the acme central website & reply by specifying port address as described in screenshot.

Note: We tried two alternatives one with -m state –state NEW, RELATED, ESTABLISHED (one side without NEW) to allow only established connections. And second approach we used port to both side for dropping other traffics. We also dropped ICMP request (ping).

Acme remote website can be accessible from only central facility users. So, as per our understanding we allowed all 3 subnetwork's traffic from central facility towards remote facility. Only allowing website which is TCP port 80 (http).

Same as 2nd answer, all unauthorised users are now not allowed to pass through Bast gateway router to protect Acme remote facility from outside world.

Same as 3rd answer, Osiris gateway router is protecting central facility from any unauthorised users to get in.

```
# This deletes all previously set rules, so that this script can run from a clean start
iptables -F

# Set default policy: drop all packets
# This means that the firewall blocks all traffic
iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP

#For Modbus Client and Server communication
iptables -A FORWARD -i eth0 -o eth1 -s 147.236.56.100 -d 147.223.112.100 -j ACCEPT
iptables -A FORWARD -i eth1 -o eth0 -s 147.223.112.100 -d 147.236.56.100 -j ACCEPT

#For Acme Central website
iptables -A FORWARD -i eth1 -o eth0 -d 147.35.152.10 -p tcp --destination-port 80 -j ACCEPT
iptables -A FORWARD -i eth0 -o eth1 -s 147.35.152.10 -p tcp --source-port 80 -j ACCEPT

#Manage access to remote factory website acmeRfactoryweb
iptables -A FORWARD -i eth0 -o eth1 -s 147.35.152.0/24,147.100.201.0/24,147.236.56.0/24 -d
147.223.112.215 -p tcp --destination-port 80 -j ACCEPT
iptables -A FORWARD -i eth1 -o eth0 -s 147.223.112.215 -d 147.35.152.0/24,147.100.201.0/24,
147.236.56.0/24 -p tcp --source-port 80 -j ACCEPT
```

BAST

```
# This deletes all previously set rules, so that this script can run from a clean s
tart
iptables -F

# Set default policy: drop all packets
# This means that the firewall blocks all traffic
iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP

#For the communication of Modbus Server and Client
iptables -A FORWARD -i eth2 -o eth1 -d 147.236.56.100 -s 147.223.112.100 -j ACCEPT
iptables -A FORWARD -i eth1 -o eth2 -d 147.223.112.100 -s 147.236.56.100 -j ACCEPT

#For giving all users access to Acme Central website
iptables -A FORWARD -d 147.35.152.10 -p tcp --destination-port 80 -j ACCEPT
iptables -A FORWARD -s 147.35.152.10 -p tcp --source-port 80 -j ACCEPT

#For giving access of the Acme remote factory website to the central facility users
iptables -A FORWARD -d 147.223.112.215 -s 147.35.152.0/24,147.100.201.0/24,147.236.
56.0/24 -p tcp --destination-port 80 -j ACCEPT
iptables -A FORWARD -d 147.35.152.0/24,147.100.201.0/24,147.236.56.0/24 -s 147.223.
112.215 -p tcp --source-port 80 -j ACCEPT
```

OSIRIS

4 CONCLUSION

A more secured Modbus protocol has been created using RSA signature and verification for the Authentication, Diffie-Hellman key exchange for creating a session key for encryption and decryption to provide Confidentiality. The Integrity is provided by using SHA-256 hash algorithm and verifying the hash values. The VPN and Firewall implementation secured the required communications on the core emulator .imn file.

5 REFERENCES

Cyberbit. (2019). *scada-modbus-protocol-vulnerabilities*. Retrieved from www.cyberbit.com: https://www.cyberbit.com/blog/ot-security/scada-modbus-protocol-vulnerabilities

How-To Geek, L. (2019). *the-beginners-guide-to-iptables-the-linux-firewall*. Retrieved from www.howtogeek.com: https://www.howtogeek.com/177621/the-beginners-guide-to-iptables-the-linux-firewall

Juniper Networks, I. (2019). *security-ipsec-vpn-overview.html*. Retrieved from www.juniper.net: https://www.juniper.net/documentation/en_US/junos/topics/topic-map/security-ipsec-vpn-overview.html

Wikimedia Foundation, I. (2019). *Modbus*. Retrieved from en.wikipedia.org: https://en.wikipedia.org/wiki/Modbus