UNIVERSITATEA POLITEHNICA BUCUREȘTI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

PROIECT DE CERCETARE

Implementarea si verificarea comparativă a firewall-urilor pentru protocoale seriale industriale

Ioana - Laura Popescu

Coordonator științific:

Prof. dr. ing. Dumitru Cristian Trancă

BUCUREŞTI

2019

UNIVERSITY POLITEHNICA OF BUCHAREST
FACULTY OF AUTOMATIC CONTROL AND COMPUTERS
COMPUTER SCIENCE DEPARTMENT

RESEARCH PROJECT

Comparative implementation and verification of serial-protocol dedicated firewalls

Ioana - Laura Popescu

Thesis advisor:

Prof. dr. ing. Dumitru Cristian Trancă

BUCHAREST

2019

TABLE OF CONTENTS

1 INTRODUCTION

1.1 Common Security Challenges

In the last 2 to 3 decades, the industrial environment has changed significantly, by the fact that network control systems have become almost inevitable in modern Industrial IoT Systems. [1]

However, along with the growth of the network and standard protocols usage, the number of vulnerabilities has grown as well.

According to [1], before adopting „Internet Technology” approach, most of the IioT systems used to operate in isolation with the „outside world” so there was absolutely no concern regarding the vulnerabilities which were faced only by the enterprise information systems for decades.

Consequences of previously mentioned threats can be, of course, much more severe in Inustrial IoT than in IT enterprise.

For example:

- Most that could happen when an attack has occured in IT enterprise is loss of data or expose of confidentiality. In IIoT, undefined behaviour can cause loss of lives.
- By the nature of these systems, and the fact that they use protocols like ModBus enabled anyone who could interface ModBus to communicate, and therefore sending commands or read sensor data.

Furthermore, in [1], because of the fact that in the past, all the IIoT operations were happening in isolation, authentication and encryption have been overlooked.

Most Industrial Control Systems are basically made of sensors/actuators, along with control devices (PLC’s) and many of them are monitored by HMI (Human Machine Interfaces), which are, most probably, Internet enabled [1].

As a prerequisite, any industrial control system that involves Control Level Equipment can be considered as part of IIoT category.

Because of this, we can confirm that attacks can vary when targetting different parts of the industrial system. Having this said, the severity of theire respective outcomes will vary as well. We will discuss possible attacks and a small categorization below.

2 PROTOCOLS OF INTEREST

2.1 Modbus over TCP

Modbus is a serial standard communications protocol commonly used in connecting industrial electronic devices [8].

It is an application layer positioned at level 7 of the OSI model, that provides client/server communication between devices connected via different types of networks. [9]

Modbus is a request/reply protocol and offers services specified by function codes. It is based on the client/server paradigm and it usually operates with one master and several slaves. These function codes are elements of request/reply PDUs.[9]

It's implementation uses:

- TCP/IP over Ethernet
- Asynchronous serial transmission via different media

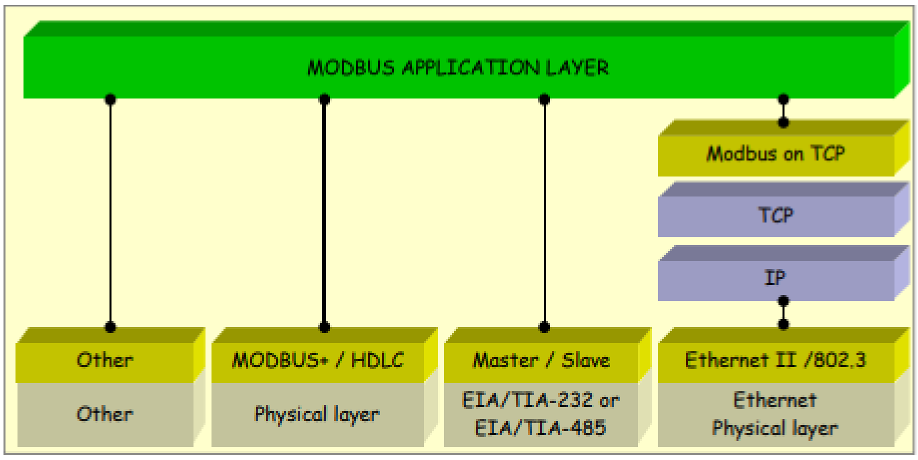


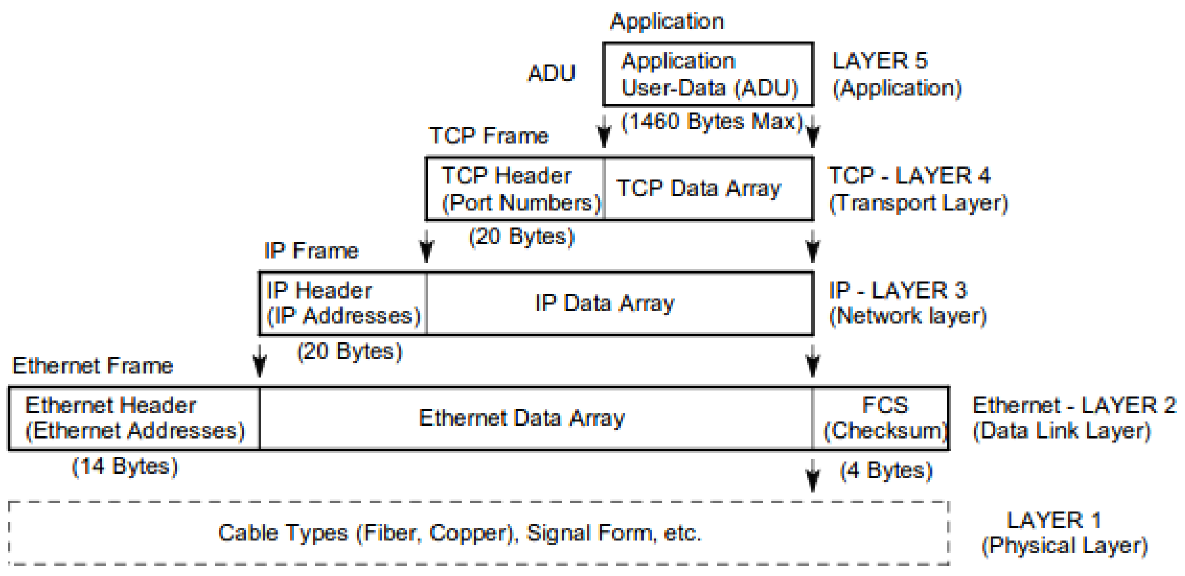
Figure 1 - Modbus communication stack 1 [9]

2.1.1 Modbus TCP

Is a version of Modbus which is encapsulated and transmitted through TCP, and while being encapsulated in TCP, it does not need a checksum field, since this is take care of from the lower levels of the OSI model. [10]

2.1.2 Modbus over TCP

Simply the Modbus protocol encapsulated in the TCP frame and keeps calculating the checksum at application layer, although the lower layers have already implemented this feature. [10]



Application
ADU
Application User-Data (ADU)
(1460 Bytes Max)
LAYER 5 (Application)
TCP Frame
TCP Header (Port Numbers)
TCP Data Array
(20 Bytes)
TCP - LAYER 4 (Transport Layer)
IP Frame
IP Header (IP Addresses)
IP Data Array
(20 Bytes)
IP - LAYER 3 (Network layer)
Ethernet Frame
Ethernet Header (Ethernet Addresses)
Ethernet Data Array
FCS (Checksum)
(14 Bytes)
(4 Bytes)
Ethernet - LAYER 2 (Data Link Layer)
Cable Types (Fiber, Copper), Signal Form, etc.
LAYER 1 (Physical Layer)

Illustration 1: Figure 11 - Construction of a TCP/Modbus Data Packet

Aș it can be seen from Figure 11 [16], the client application forms it’s request then passes it’s data down to the lower layers which add their own control information to the packet în the form of protocol headers.

After all of the actions mentioned, the packet will reach the physical layer, and passes through the network, untill it reaches it’s desctination. When that happend, the packet will be decoded, at each portion by the appropiate layer, when traveling up, untill it reaches the destination application. When that happens, the server/slave application will respond accordingly, and the response will be encoded once again and go through the same process back to source application.

Although each layer only comunicates with the layer above or below it, this can easily be visualised that each layer belonging to one conversation participant, will be a discussing „partner” for the layer with the same number belonging to the other conversation party envolved. These protocols that belong to the same layer, even on different nodes, are called *Peer Entities [16].*

The TCP/IP protocol provides all necessary resources for 2 devices to communicate with each other over a Local Area Network or Global Area Network.

However, it only guarantees that all packages are sent between the two participants, it has nothing to do with the content of those messages, such that if the devices do not understand each other it is no concern of the TCP protocol. [16]

With that being said, we may say that Modbus is an application level protocol on the *client/server* model where the client sends a request message, and the server simply responses with a response message. The main difference is, that modbus requests are oriented towards:

- read/write operations on registers, for cofiguration
- control device I/O [16]

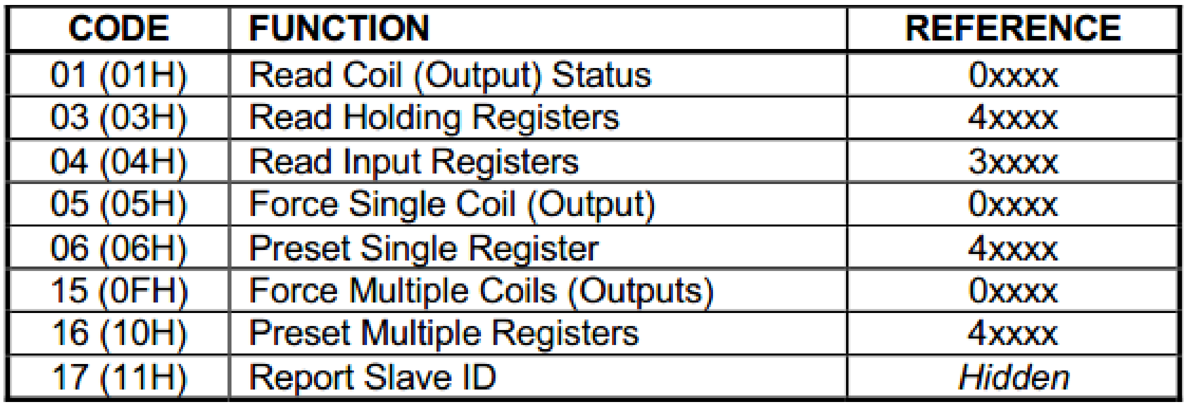
2.2 Function codes

The function code field of the PDU specifies two main informations: the kind of register group it operates on, and the kind of action the slave will take on this particular kind of register.

There are three categories of Modbus Functions codes listed below:

- Public function codes – publicly documented, unique, well defined function codes
- User-defined Function codes – can be user implemented, not necessarily unique and having 2 large ranges for user-defined function. If a user wishes to implement it’s own functionality and pudlish it, it must register with some RFC.

- Reserved Function codes – not available for public use.[9]



CODE	FUNCTION	REFERENCE
01 (01H)	Read Coil (Output) Status	0xxxx
03 (03H)	Read Holding Registers	4xxxx
04 (04H)	Read Input Registers	3xxxx
05 (05H)	Force Single Coil (Output)	0xxxx
06 (06H)	Preset Single Register	4xxxx
15 (0FH)	Force Multiple Coils (Outputs)	0xxxx
16 (10H)	Preset Multiple Registers	4xxxx
17 (11H)	Report Slave ID	*Hidden*

3 EXISTING SECURITY SOLUTIONS

1.1 Tofino 9211-ET-HN3

Honeywell’s Tofino’s Modbus Read-Only Firewall is a new Security solution from Honeywell, preconfigured to filter traffic between Experion control network and Secured System, allowing only read SIS data and respond only to time synchronisation requests. [17]

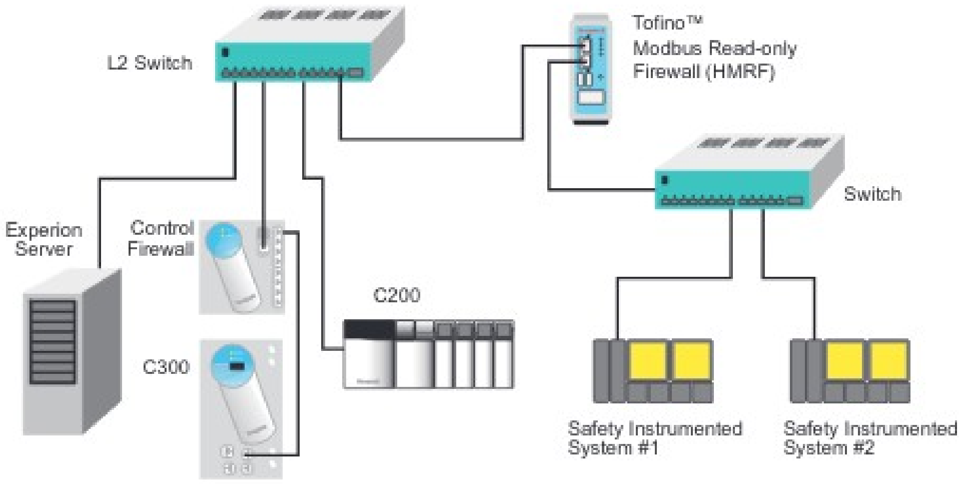

L2 Switch
Tofino™ Modbus Read-only Firewall (HMRF)
Switch
Experion Server
Control Firewall
C200
C300
Safety Instrumented System #1
Safety Instrumented System #2

Illustration 2: Figure 13- Application Method[17]

The Tofino Modbus Read-only Firewall is pretty simple:

- it has two ethernet ports
    - 1 ethernet port responsible with unsecured traffic
    - 1 ethernet port outputs the fitlered unsecured traffic
- It requires power supply of 9-32V DC.
- Software is upgradable via USB and the upgrade process requires that all files loaded via USB to be encrypted and signed for integrity
- Statefull firewall with deep packet inspection
- Throughput: 1000 Modbus TCP frames per second.
- Automatic reporting of traffic statistics.
- Diagnosis can also be performed on the available USB ports.

1.2 On the use of open-source firewalls in ICS/SCADA systems

Authors at [18] experiment Linux’s iptables tool and configure it in such a way that they exploit it’s potential of generating a potential SCADA firewall.

Their experiments are achieved by utilising advanced iptables features without chainging the framework itslef, so any Linux system may be turned into an effective SCADA firewall [18].

The authors have conducted a survey of available firewall utilities in BSD (PF, IPFW, IP filter) and Linux iptables, so they can find the best candidate for a SCADA firewall based on benchmark.

The main characteristic they looked for are:

- Statefull filtering
    - All of the mentioned utilities follow this paradigm
- Port, host, protocol
    - all of the utilities can filter traffic based on port, host, protocol etc.
- NAT
    - Yes to all
- String match capabilty
    - Only Iptables
- Byte match capability
    - Only iptables. Highly required feature if one attempts to filter packets based on allowing a selected number of function codes.

To validate their proposed solution, authors have built a test SCADA network set up for the experiment consisting of: 3 DirectLOGIC05 PLC’s, marked as PLC1, PLC2 and PLC3.

Each of the aformentioned PLC’s communicates via Modbus TCP, so this means that all of them is a modbus client/master.

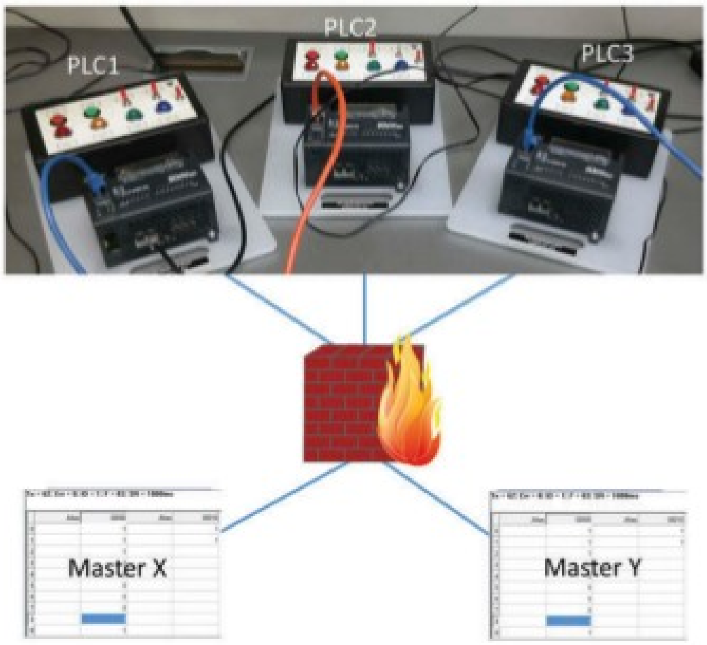


Illustration 3: Experimental Setup [17]

Each PLC contains four discrete inputs connected to four different switches and four outputs connected to four different lights, having a certain number of register to store input and output signals.

They’re rules consist of allwoing only master X to perform certain write requests to PLC1, other master Y to perform certain write requests to PLC2 and so on. Following tests have been performed:

(1) Master X sends write requests to PLC1 with

value1—All lights turn on.

(2) Master Y sends write requests to PLC1 with

value 0—No change. (request was blocked by firewall)

(3) Master X sends write requests to PLC2 with

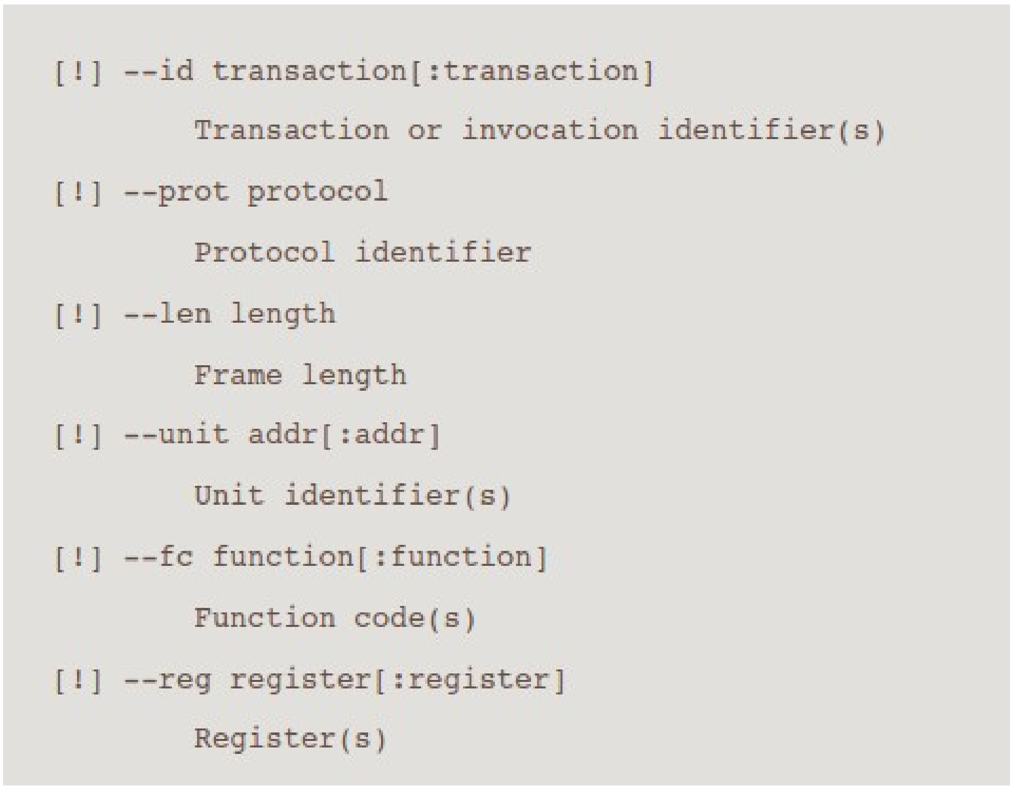
value 1—No change. (firewall blocked) [17]

1.3 Open-source IEC 61131-3 virtual machine

This other firewall solution has been implemented in kernel space as a modbus filter implemented in kernel 2.6.16 and no longer compiles against Linux kernel versions. [19]

It has, however, been rewritten and includes Modbus frame matching available in the previous versions of the code (written in 2004).

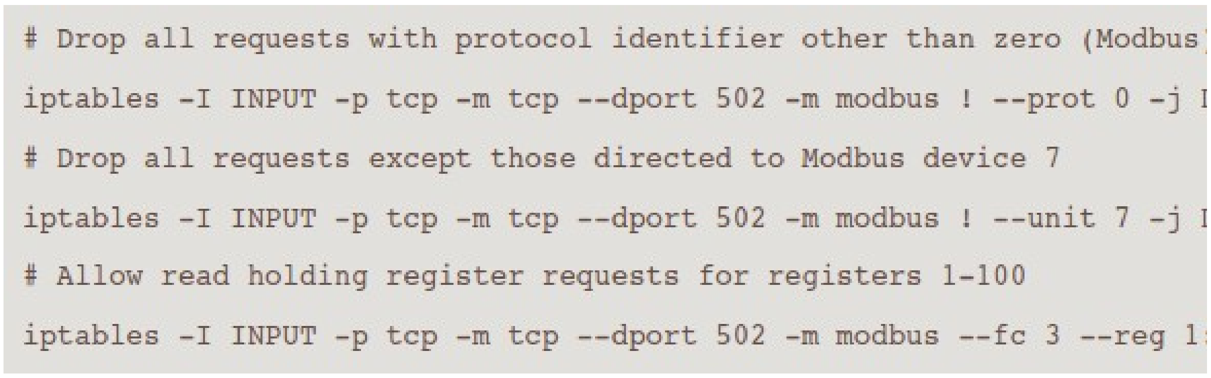
Extended packet matching can be added with iptables’ match options and the filtering can be defined using the following provided fields:



```
[!] --id transaction[:transaction]
        Transaction or invocation identifier(s)
[!] --prot protocol
        Protocol identifier
[!] --len length
        Frame length
[!] --unit addr[:addr]
        Unit identifier(s)
[!] --fc function[:function]
        Function code(s)
[!] --reg register[:register]
        Register(s)
```

Illustration 4: Match extensions [19]

Using the extensions provided by the authors kernel module with iptables filtering rules can be established as the following:



```
# Drop all requests with protocol identifier other than zero (Modbus

iptables -I INPUT -p tcp -m tcp --dport 502 -m modbus ! --prot 0 -j

# Drop all requests except those directed to Modbus device 7

iptables -I INPUT -p tcp -m tcp --dport 502 -m modbus ! --unit 7 -j

# Allow read holding register requests for registers 1-100

iptables -I INPUT -p tcp -m tcp --dport 502 -m modbus --fc 3 --reg 1
```

Illustration 5: Drop requests [19]

4 IMPLEMENTATION

Last semester the research proof of concept application consisted of a combined firewall implemented on a Atmega128P, which filtered packets for Modbus RTU over USART. The algorithm consisted of a FSM model, which would let packets travel byte by byte and when it detected that the Modbus request was not an allowed one (which were stored on a predefined array), it altered the content of the next byte, so it would not respect the checksum.

This time, the implementation was oriented towards ModBus TCP, with a proof of concept implemented în userspace, following a future implementation în kernel space, and constructing a benchmark with the performance of the firewall on RT vs Non-RT kernel.

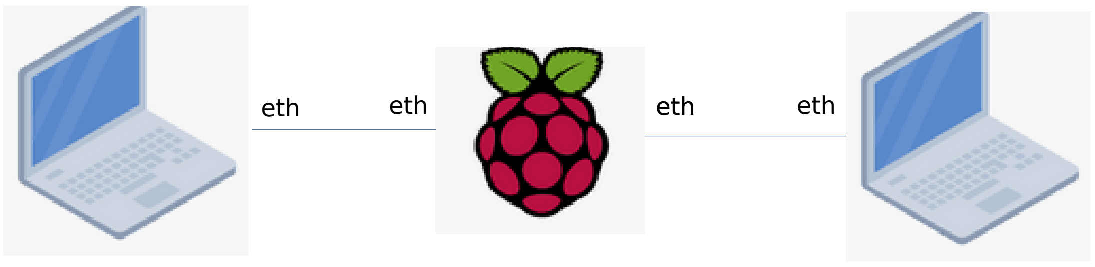
The setup consistend mainly of two PC's acting aș a master and a slave respectively on ModbusTCP, a raspberry PI with the firewall/sniffer implemented în userspace and ethernet cables.

The design of the application will not be based on FSM model anymore, and instead, it will consist of intercepting a packet, decoding it, checking the function code (if it is allowed or not), and then choosing to forward the packet to the master or not. If it does not respect the function code rules, then the packet will be dropped.

The algorithm is quite simple, it consisted of the following:

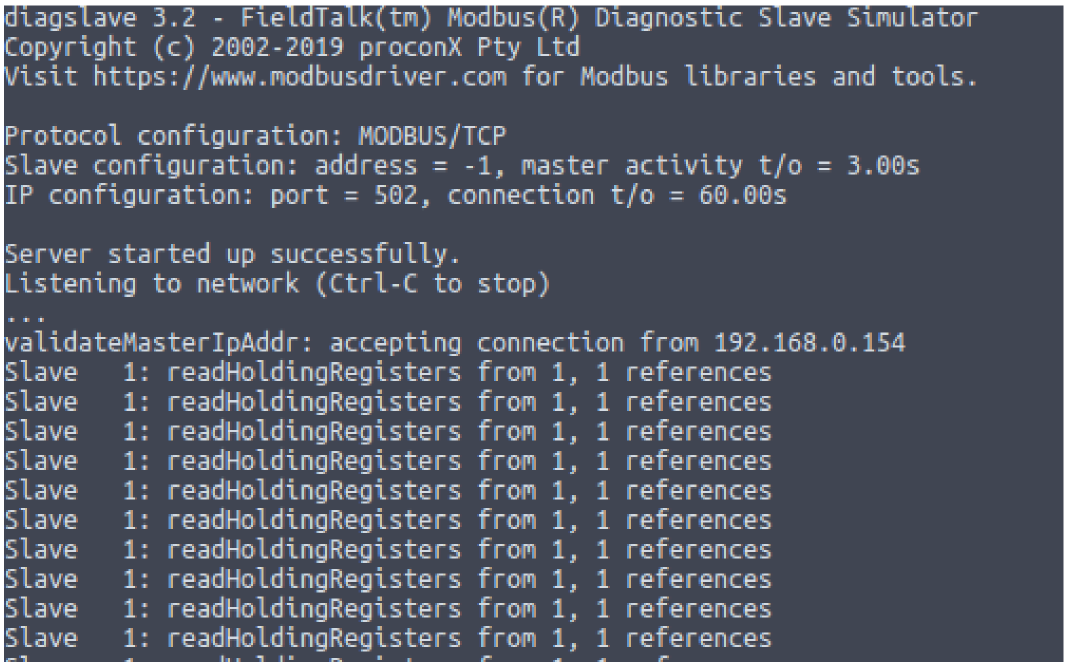
- Opens a socket, intercepts all IP packets traveling through all the available interfaces
- uses iphandler to check protocol (în this case tcp) and tcp source port, which is 502 (for ModBusTCP)
- sends the appropiate packets to processPacket function, which prints the eth, IP and TCP header, and then prints the Payload în hex format.

- Opens another socket, using SO_BINDTODEVICE flag, so it will forward the packet to the master (if it passes the firewall, în this case, only read operations allowed).



Modbus simulations were done using mdpoll and diagslave for the modbus master and modbus slave application simulators.

To prove the functionality here is the start of a listening modbus slave, which managed to make a connection:

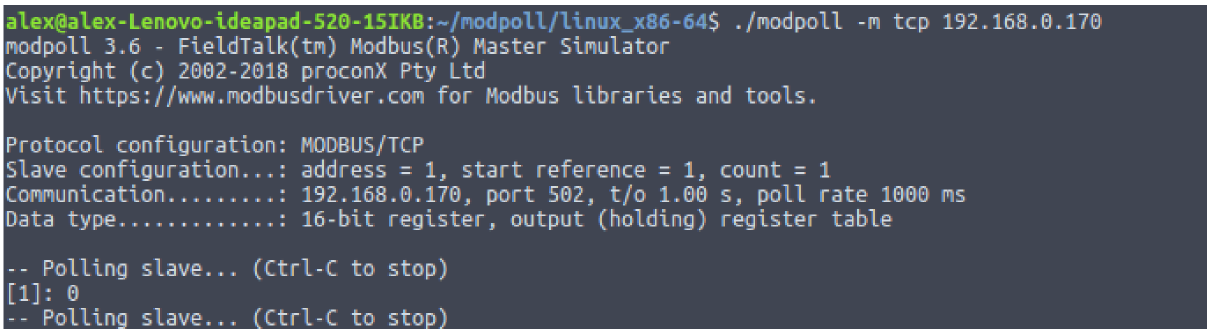


```
diagslave 3.2 - FieldTalk(tm) Modbus(R) Diagnostic Slave Simulator
Copyright (c) 2002-2019 proconX Pty Ltd
Visit https://www.modbusdriver.com for Modbus libraries and tools.

Protocol configuration: MODBUS/TCP
Slave configuration: address = -1, master activity t/o = 3.00s
IP configuration: port = 502, connection t/o = 60.00s

Server started up successfully.
Listening to network (Ctrl-C to stop)
...
validateMasterIpAddr: accepting connection from 192.168.0.154
Slave   1: readHoldingRegisters from 1, 1 references
Slave   1: readHoldingRegisters from 1, 1 references
Slave   1: readHoldingRegisters from 1, 1 references
Slave   1: readHoldingRegisters from 1, 1 references
Slave   1: readHoldingRegisters from 1, 1 references
Slave   1: readHoldingRegisters from 1, 1 references
Slave   1: readHoldingRegisters from 1, 1 references
Slave   1: readHoldingRegisters from 1, 1 references
Slave   1: readHoldingRegisters from 1, 1 references
Slave   1: readHoldingRegisters from 1, 1 references
```

The corresponding master/client application on the other ubuntu PC connected via ssh:

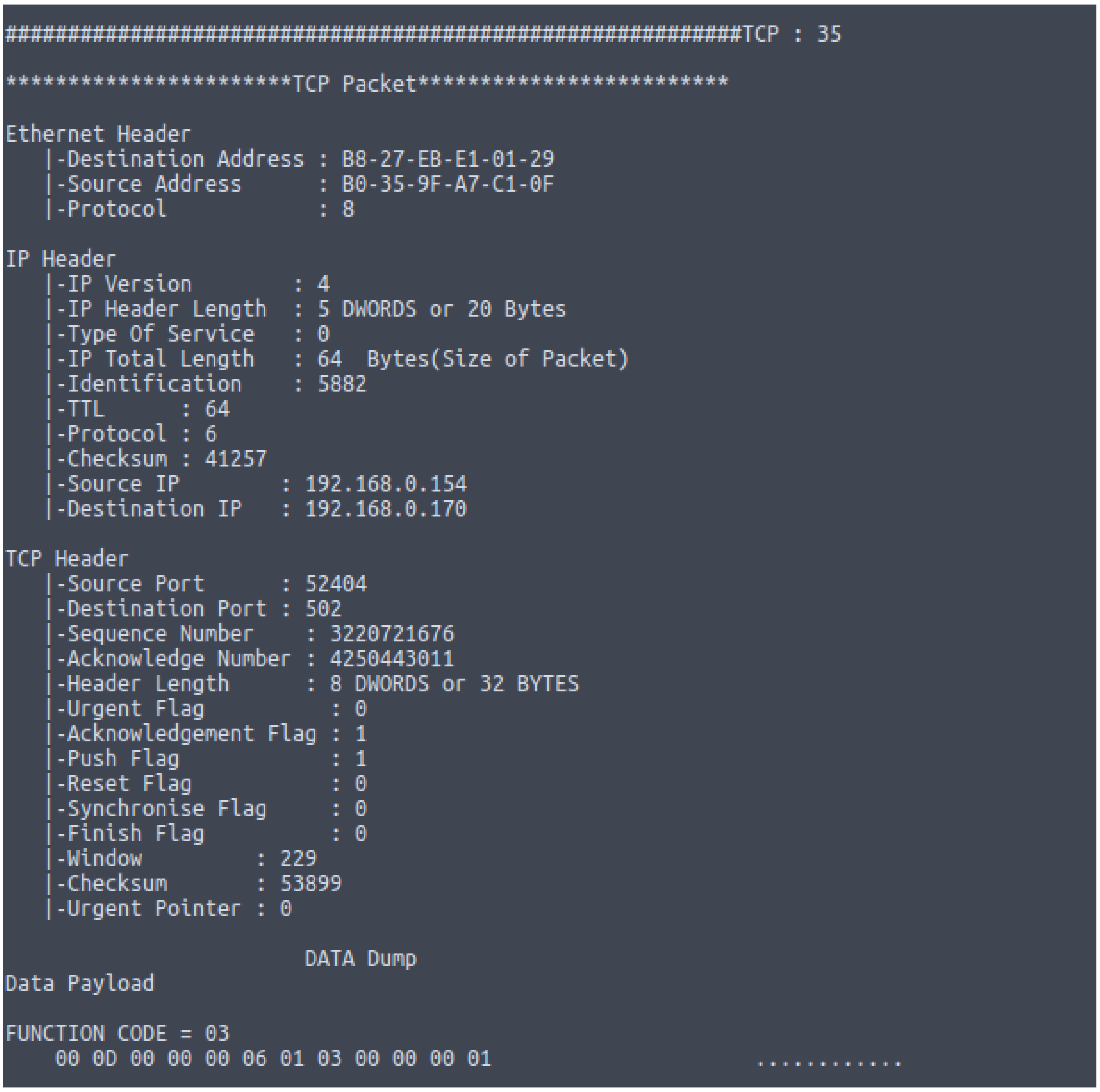


```
alex@alex-Lenovo-ideapad-520-15IKB:~/modpoll/linux_x86-64$ ./modpoll -m tcp 192.168.0.170
modpoll 3.6 - FieldTalk(tm) Modbus(R) Master Simulator
Copyright (c) 2002-2018 proconX Pty Ltd
Visit https://www.modbusdriver.com for Modbus libraries and tools.

Protocol configuration: MODBUS/TCP
Slave configuration...: address = 1, start reference = 1, count = 1
Communication.........: 192.168.0.170, port 502, t/o 1.00 s, poll rate 1000 ms
Data type.............: 16-bit register, output (holding) register table

-- Polling slave... (Ctrl-C to stop)
[1]: 0
-- Polling slave... (Ctrl-C to stop)
```

And some of the output generated by the sniffer after packet decoding:



```
###########################################################TCP : 35

***********************TCP Packet*************************

Ethernet Header
   |-Destination Address : B8-27-EB-E1-01-29
   |-Source Address      : B0-35-9F-A7-C1-0F
   |-Protocol            : 8

IP Header
   |-IP Version        : 4
   |-IP Header Length  : 5 DWORDS or 20 Bytes
   |-Type Of Service   : 0
   |-IP Total Length   : 64  Bytes(Size of Packet)
   |-Identification    : 5882
   |-TTL      : 64
   |-Protocol : 6
   |-Checksum : 41257
   |-Source IP        : 192.168.0.154
   |-Destination IP   : 192.168.0.170

TCP Header
   |-Source Port      : 52404
   |-Destination Port : 502
   |-Sequence Number    : 3220721676
   |-Acknowledge Number : 4250443011
   |-Header Length      : 8 DWORDS or 32 BYTES
   |-Urgent Flag          : 0
   |-Acknowledgement Flag : 1
   |-Push Flag            : 1
   |-Reset Flag           : 0
   |-Synchronise Flag     : 0
   |-Finish Flag          : 0
   |-Window         : 229
   |-Checksum       : 53899
   |-Urgent Pointer : 0

                        DATA Dump
Data Payload

FUNCTION CODE = 03
    00 0D 00 00 00 06 01 03 00 00 00 01                    ............
```

5 BIBLIOGRAFIE

[1] A. Mahboob and J.Zubairi, "Intrusion Avoidance for SCADA Security in Industrial Plants" 2010. [Online]. Available:

https://www.researchgate.net/publication/216485705_Intrusion_avoidance_for_Scada_security_in

_industrial_plants

[2] L. Rosa et al., "Attacking SCADA systems: a practical perspective," *Integr. Netw. Serv. Manag. (1M), 2017 IFIP/IEEE Symp*., 2017.

[3] "Understanding SCADA system Security". 2001. [Online]. Available: http://www.iwar.org.uk/cip/resources/utilities/SCADAWhitepaperfinal1.pdf.

[4]
S.Tom, "Chinese Hacking Team Caught Taking Over Decoy Plant," 2013. [Online]. Available: https://www.technologyreview.com/s/517786/chinesehacking-team-caught-taking-over-decoy-water-plant/.

[5]
“SCADA Systems Face Diverse Software Attack Threats,” 2013. [Online].

Available: https://www.afcea.org/content/scada-systems-face-diverse-softwareattack-threats. [Accessed: 30-Jan-2019].

[6] G. JAKABOCKZI and E. ADAMKO, "Vulnerabilties of Modbus RTU Protocol - A case Study", Ann. Oradea Univ. Fascicle Manag. Tchnol. Eng., 2015.

[7]
K. Gregg "Sloppy' Chinese hackers scored data theft coup with 'Night Dragon'" 2011. [Online] Available: https://www.computerworld.com/article/2513128/security0/-sloppy-chinese-hackers-scored-data-theft-coup-with-night-dragon-.html. [Accessed: 31-Jan-2019]

[8] Drury, Bill (2009). Control Techniques Drives and Controls Handbook (PDF) (2nd ed.) Institution of Engineering Technology

[9] Modbus Application Protocol Specification V1.1b [Online] Available: http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf

[1 D.C Tranca, PhD Thesis, Optimizaions in Industrial Internet of Things [Online]

0] Available:

https://drive.google.com/file/d/1TB9TDCKJkuFsnKEjocKFjU282Q2vJkgg/view

[1 Technical Committee 57, [Online] Available: http://tc57.iec.ch/index-tc57.html
1]

[1 R. Mackiewicz, Sterling Heights, "Technical Overview and Benefits of IEC 61850
2] Standard for Substation Automation" [Online] Available:

https://library.e.abb.com/public/04519389e504d7ddc12576ff0070704d/
3BUS095131_en_IEC61850_Overview_and_Benefits_Paper_General.pdf

[1 M.T.A Rashid, S. Yussof, Y. Yussoff, and R. Ismail, "A review of security attacks on IEC
3] 61850 substation automation system network", in Conference Proceedings - 6th International Conference on Information Technology and Multimedia at UNITEN: Cultivating Creativity and Enabling Techynology Through the Internet of Things, ICIMU 2014, 2015.

[1 R. Ajami, Anh Dinh, "Embedded Network Firewall on FPGA", [Online] Available:
4] https://www.researchgate.net/publication/220841654_Embedded_Network_Firewall_on_FPGA

[1 M.Cheminod, L. Durante, A. Valenzano, C. Zunino, "Performance Impact of
5] Commercial Industrial Firewalls on Networked Control Systems" [Online] Available: https://ieeexplore.ieee.org/abstract/document/#

[16] https://www.prosoft-technology.com/kb/assets/intro_modbustcp.pdf

[17] https://www.tofinosecurity.com/sites/default/files/eps9211-et-hn3-1.pdf

2 ANEXE

```
#include<netinet/in.h>

#include<errno.h>
#include<netdb.h>
#include<stdio.h>   //For standard things
#include<stdlib.h>  //malloc
#include<string.h>  //strlen

#include<netinet/ip_icmp.h> //Provides declarations for icmp header
#include<netinet/udp.h> //Provides declarations for udp header
#include<netinet/tcp.h> //Provides declarations for tcp header
#include<netinet/ip.h>  //Provides declarations for ip header
#include<netinet/if_ether.h>    //For ETH_P_ALL
#include<net/ethernet.h>    //For ether_header
#include<sys/socket.h>
#include<arpa/inet.h>
#include<sys/ioctl.h>
#include<sys/time.h>
#include<sys/types.h>
#include<unistd.h>

void ProcessPacket(unsigned char* , int);
void print_ip_header(unsigned char* , int);
void print_tcp_packet(unsigned char * , int );
void print_udp_packet(unsigned char * , int );
void print_icmp_packet(unsigned char* , int );
void PrintData (unsigned char* , int);

struct sockaddr_in source,dest;
int tcp=0,udp=0,icmp=0,others=0,igmp=0,total=0,i,j;

int main()
```

```
{
    int saddr_size , data_size;
    struct sockaddr saddr;

    unsigned char *buffer = (unsigned char *) malloc(65536); //Its Big!

    printf("Starting...\n");

    int sock_raw = socket( AF_PACKET , SOCK_RAW , htons(ETH_P_ALL)) ;
    //TBD: setsockopt(sock_raw , SOL_SOCKET , SO_BINDTODEVICE , "eth0" ,
strlen("eth0")+ 1 );

    if(sock_raw < 0)
    {
        //Print the error with proper message
        perror("Socket Error");
        return 1;
    }
    while(1)
    {
        saddr_size = sizeof saddr;
        //Receive a packet
        data_size = recvfrom(sock_raw , buffer , 65536 , 0 , &saddr ,
(socklen_t*)&saddr_size);
        if(data_size <0 )
        {
            printf("Recvfrom error , failed to get packets\n");
            return 1;
        }
        //Now process the packet
        ProcessPacket(buffer , data_size);
    }
    close(sock_raw);
    printf("Finished");
    return 0;
}

void ProcessPacket(unsigned char* buffer, int size)
{
    //Get the IP Header part of this packet , excluding the ethernet header
    struct iphdr *iph = (struct iphdr*)(buffer + sizeof(struct ethhdr));

    struct tcphdr *tcph=(struct tcphdr*)(buffer + iph->ihl*4 + sizeof(struct ethhdr));
    ++total;
    if(((unsigned int)(iph->protocol)) == 6) // if it's tcp and port is the one specific to
modbus
    {
        ++tcp;
        print_tcp_packet(buffer, size);
    }
    printf("TCP : %d", tcp);
}

void print_ethernet_header(unsigned char* Buffer, int Size)
{
    struct ethhdr *eth = (struct ethhdr *)Buffer;

    printf( "\n");
    printf( "Ethernet Header\n");
    printf( "   |-Destination Address : %.2X-%.2X-%.2X-%.2X-%.2X-%.2X \n", eth-
>h_dest[0] , eth->h_dest[1] , eth->h_dest[2] , eth->h_dest[3] , eth->h_dest[4] ,
eth->h_dest[5] );
    printf( "   |-Source Address      : %.2X-%.2X-%.2X-%.2X-%.2X-%.2X \n", eth-
>h_source[0] , eth->h_source[1] , eth->h_source[2] , eth->h_source[3] , eth-
>h_source[4] , eth->h_source[5] );
    printf( "   |-Protocol            : %u \n",(unsigned short)eth->h_proto);
```

```
}

void print_ip_header(unsigned char* Buffer, int Size)
{
    print_ethernet_header(Buffer , Size);

    unsigned short iphdrlen;

    struct iphdr *iph = (struct iphdr *)(Buffer  + sizeof(struct ethhdr) );
    iphdrlen =iph->ihl*4;

    memset(&source, 0, sizeof(source));
    source.sin_addr.s_addr = iph->saddr;

    memset(&dest, 0, sizeof(dest));
    dest.sin_addr.s_addr = iph->daddr;

    printf( "\n");
    printf( "IP Header\n");
    printf( "   |-IP Version        : %d\n",(unsigned int)iph->version);
    printf( "   |-IP Header Length  : %d DWORDS or %d Bytes\n",(unsigned int)iph-
>ihl,((unsigned int)(iph->ihl))*4);
    printf( "   |-Type Of Service   : %d\n",(unsigned int)iph->tos);
    printf( "   |-IP Total Length   : %d  Bytes(Size of Packet)\n",ntohs(iph->tot_len));
    printf( "   |-Identification    : %d\n",ntohs(iph->id));
    printf( "   |-TTL      : %d\n",(unsigned int)iph->ttl);
    printf( "   |-Protocol : %d\n",(unsigned int)iph->protocol);
    printf( "   |-Checksum : %d\n",ntohs(iph->check));
    printf( "   |-Source IP        : %s\n",inet_ntoa(source.sin_addr));
    printf( "   |-Destination IP   : %s\n",inet_ntoa(dest.sin_addr));
}

void print_tcp_packet(unsigned char* Buffer, int Size)
{
```