Using a Packet Manipulation Tool for Security Analysis of Industrial Network Protocols

Tiago H. Kobayashi, Aguinaldo B. Batista Jr.
Agostinho M. Brito Jr., Paulo S. Motta Pires

LabSIN - Security Information Laboratory
Department of Computer Engineering and Automation - DCA
Federal University of Rio Grande do Norte - UFRN
Natal, 59.078-970, RN, Brazil
(hiroshi, aguinaldo, ambj, pmotta)@dca.ufrn.br

Abstract

Scapy is a free and open source packet manipulation environment written in Python language. In this paper we present a Modbus extension to Scapy, and show how this environment can be used to build tools for security analysis of industrial network protocols. Our implementation can be extended to other industrial network protocols and can help security analysts to understand how these protocols work under attacks or adverse conditions.

1 Introduction

In the last years, industry has paid attention to devices that support open standards. Devices based on open standards are easier to interface with other devices and software improving efficiency and productivity. The adoption of open systems can also reduce costs including maintenance and people training. Toward this way, the use of TCP/IP (Transmission Control Protocol/Internet Protocol)-based computer networks within automation systems is growing up very fast. The experience with TCP/IP internetworking can be used to improve and create new functionalities that were impossible by technical limitations. However, common problems for open systems, such as electronic intrusion and other security threats can now affect more effectively SCADA systems [6, 8, 11, 13].

Despite this migration to open standards and systems in industry, there are still few security tools specific for automation technology environments. Tools capable to carry out security tests for automation systems can be extremely valuable, once they can verify the behavior of devices before their placement into industrial plants. One solution for AT (Automation Technology) security testing is to promote the direct use or adjustment of IT (Information Technology) security tools in industrial environments, including protocol and services support.

This paper focuses on the use of Scapy [3] as packet manipulation tool that can be easily extended to perform security tests on industrial environments. Scapy is written in Python language and offers wide versatility for the development of network related applications. This packet manipulation tool gives to the user the Python command line interpreter, working as an environment for developing network software experiments.

We show how an industrial cyber security expert can use a packet manipulation environment to build its own security tests. As a proof-of-concept, we implemented the Modbus protocol [9] as a Scapy extension. With this implementation, we were able to address simple security tests by sending malformed packets to a Modbus server. The results show how easy is to perform such tests and that more complex experiments can be accomplished with this tool. Examples of such tests includes network stressing for communication assessment and protocol or device testing.

The paper is organized as follows. Section 2 describes the Modbus protocol and Modbus encapsulation within the transport layer (Modbus/TCP). Section 3 presents the Scapy packet manipulation tool approach and section 4 shows our implementation of Modbus protocol within Scapy. Implementation details about our tool and the tests realised are presented in section 5. Finally, we conclude the paper in section 6 by summarizing the work done and suggesting some future directions.

2 Modbus

Modbus [2] is an industrial protocol used for master-slave communications. The protocol was built to handle communications between PLCs (Programmable Logic Controllers) and field devices instruments (sensors and actuators). Modbus was developed by Modicon Industrial Automation Systems (currently Schneider Electric) in late

70's and recently became licensed under the GNU GPL open source license [7] for use and modification. Since Modbus protocol is simple to understand and easy to implement, it has become one of most employed protocols in automation.

Due to its simplicity and popularity, new communication channels such as Ethernet has been incorporated into Modbus. A variant of Modbus, called Modbus/TCP [2], was developed to allow Modbus communications over Ethernet networks. It uses the TCP/IP protocol stack for master-slave transactions. Additionally, Modbus/TCP brought some innovations which contributed to improve connectivity resources. In spite of flexibility achieved to process control networks, it also brought performance and security problems which are common in IT environments addressed in literature [4].

The implementation of Modbus/TCP is freely available and can be supported by an implementation guide [2]. On the Modbus/TCP implementation guide, the reader is able to find development details of the protocol, like TCP connection management, the use of TCP/IP stack, how clients and servers can communicate using the application layer, object model diagrams, sequence diagrams, and so on. Following the directions provided by technical documentation [2], it was possible to implement Modbus/TCP communication functions using Scapy.

3 Packet Manipulation with Scapy

A packet manipulator is a software that can control a network device, building custom packets to communicate with other devices. Using such kind of software, it is possible to build packets by modifying field values which are convenient to the user even if these values are not foreseen. This fact permits to assess the behavior of network services and devices when they receive these unexpected packets. There are a lot of existing software in the market that could be used or adapted for this purpose, such as Pacgen [5], Nemesis [10] and Hping [12]. However, Scapy [3] is the one that offers the most complete and versatile environment for developing security tests.

Scapy [3] is a powerful interactive and multiplatform software for packet manipulation. It is fully developed under an open source license model and it has the Python interpreter as command board. Therefore, the user can utilize Scapy combined with Python programming environment, with all resources of this language available for building new tools.

Figure 1 shows how easy Scapy can be used to build custom network packets. The examples present, respectively, the building of a TCP segment with source port, destination port and SYN/ACK flags, and a complete packet involving the Ethernet, IP and UDP protocols.

```
# ./scapy.py
Welcome to Scapy (1.0.4.74beta)
>>> TCP(sport=10,dport=53,flags="SA")
<TCP  sport=10 dport=domain flags=SA |>
>>>
```

```
# ./scapy.py
Welcome to Scapy (1.0.4.74beta)
>>> Ether(src="00:0a:3b:4d:5c:65")/IP(version=4,
dst="192.168.0.1")/UDP(sport=53,dport=43)
<Ether  src=00:0a:3b:4d:5c:65 type=0x800 |<IP
version=4 frag=0 proto=UDP dst=192.168.0.1 |<UDP
 sport=domain dport=nicname |>>>
>>>
```

Figure 1. Example of packet manipulation with Scapy

4 Modbus Implementation in Scapy

Although Scapy does not have a definitive documentation, there is a guide that helps with the development of additional tools available in the official site of the project [3]. This documentation was a important starting point for our Modbus/TCP implementation in Scapy and may be very useful for similar purposes.

A generic class named `Modbus` defines the fields specified in Modbus protocol (see Figure 2). `Modbus` class inherits from `Packet` class, an abstract class for packet creation implemented by Scapy. The `fields_desc` variable defines a list containing the fields of the Modbus protocol header [2]. The `post_build` method is used to calculate the size of a created packet.

```
class Modbus(Packet):
    name = "Modbus"
    fields_desc = [ ShortField("transaction_id",0),
                    ShortField("protocol_id", 0),
                    ShortField("data_length",None),
                    ByteField("unit_id", 0),
                    ByteField("function_code", 0),
                    DataField("data",{}) ]
    def post_build(self, p, pay):
        if self.data_length is None:
            length = len(p[8:])+2
            p = p[:4] + struct.pack('>H',length)+ p[6:]
        return p
```

Figure 2. Modbus Class.

The most important class developed in this implementation is `ModbusFH` (see Figure 3). It creates common fields of Modbus/TCP packet (header and function fields) that will be shared by all other classes that implemented functions of the `Modbus` class. The `ModbusFH` class is used by other classes developed to implement the Modbus/TCP functions. This class builds the `transaction_id`, `protocol_id`, `data_length`, `unit_id` and `function_code` fields. The `data` field is built in each Modbus function class because each one of these functions has different contents for this field.

The other classes implement error control and functions defined in the Modbus specification. To each func-

```
class ModbusFH(Packet):
    name = "ModbusFH"
    fields_desc = [ ShortField("transaction_id", 0),
                    ShortField("protocol_id", 0),
                    ShortField("data_length", None),
                    ByteField("unit_id", 0),
                    ByteField("function_code", 0)]
    def post_build(self, p, pay):
        if self.data_length is None:
            length = len(p[8:])+2
            p = p[:4] + struct.pack('>H',length)
            + p[6:]
        return p
```

Figure 3. ModbusFH Class

tion, there is a class to request and other to response. The other Modbus functions had been addressed in the same way. Figure 4, e.g., presents the request function `Write Multiple Registers` (function code `0x10`) that is used to write a block of contiguous registers in remote device. For all functions, if the `function_code` is not defined by the user, this field receives the standard value related to that function.

```
class WriteMultipleRegistersRequest(ModbusFH):
    name = "Function Write Multiple Registers
Request"
    fields_desc = ModbusFH.fields_desc + [ Wri
teMultipleRegistersRequestBaseField("data",{}) ]
    def post_build(self, p, pay):
        if self.function_code is 0:
            p = p[:7] + struct.pack('>B',16) +
p[8:]
        if self.data_length is None:
            length = len(p[8:])+2
            p = p[:4] + struct.pack('>H',length)
 + p[6:]
        return p
```

Figure 4. WriteMultipleRegisterRequest Class

5 Using Modbus/TCP Scapy Implementation

The proposed tool was tested using a local network composed of two hosts interconnected, as presented in Figure 5. The client host executes the Scapy with our extension. The server runs the jamod Modbus/TCP server [1] on port 502/TCP. Jamod [1] is a fully object oriented library that implements several functions of the Modbus/TCP specification. This library is listed as a link in section Technical Resources of site Modbus-IDA [2]. Future works with our Modbus testing tool will be performed directly in automation hardware.

Figure 6 presents how data can be read and written through the network. In this listing, the user creates two Modbus/TCP packets: `Read Coil Request` (line 5) and `Write Single Coil Request` (line 6). The user starts a new TCP connection with the server, store packets in a list (variable `l` in line 7) and sent them to the network when the user calls the `connectionSocket`

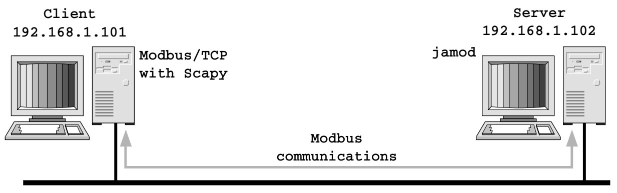

Figure 5. Testbed infrastructure

function (line 9) provided by using Python internal libraries.

```
1  # ./modbus.py -s session
2  INFO: Using session [mysession]
3  Welcome to Scapy (1.0.5.20beta)
4  Implementacao Modbus: 1.2 $
5  >>> a = ReadCoilRequest(data={"address":0,"count":4})
6  >>> k = WriteSingleCoilRequest(data={"address":0,
7  "value":[0,1,0]})
8  >>> l = [a,k]
9  >>> connectionSocket("192.168.1.102",502,l)
10 Begin emission:
11 Finished to send 1 packets.
12 *
13 Received 1 packets, got 1 answers, remaining 0 packets
14 Begin emission:
15 Finished to send 1 packets.
16 *
17 Received 1 packets, got 1 answers, remaining 0 packets
18 >>>
```

Figure 6. Example of Modbus/TCP communication with Scapy

Figure 7 presents how the jamod Modbus/TCP server receives and processes the packets sent by Scapy. The server boots up (line 1) and starts to listen incoming connections. When our remote client request a new connection, the server respond to it starting the connection (lines 7 and 8). The last lines in Figure 7 show the received requests and the responses to the `Read Coil` (lines 9 and 10) and `Write Single Coil` (lines 11 and 12) packets, respectively.

```
1  # java -Dnet.wimpi.modbus.debug=true \
2  net.wimpi.modbus.cmd.TCPSlaveTest 502
3  jModbus Modbus Slave (Server)
4  Listening...
5  Listenening to ServerSocket[addr=0.0.0.0/0.0.0.0,
6  port=0,localport=502](Port 502)
7  Making new connection Socket[addr=/192.168.1.101,
8  port=60100,localport=502]
9  Request:00 00 00 00 00 06 00 01 00 00 00 04
10 Response:00 00 00 00 00 03 00 81 02
11 Request:00 00 00 00 00 06 00 05 00 00 ff 00
12 Response:00 00 00 00 00 06 00 05 00 00 ff 00
```

Figure 7. Jamod server log

Our tool was tested for building malformed packets. They are also useful for building future security tools to test server implementations against unusual or unpredicted situations. In the example presented in Figure 8, the user fills the `data_length` with a number (765 in decimal) that is larger than the size of the packet that has been

```
# ./modbus.py -s session
INFO: Using session [mysession]
Welcome to Scapy (1.0.5.20beta)
Implementacao Modbus: 1.2 $
>>> a = ReadCoilRequest(data_length=765, data=
{"address":0,"count":4})
>>> connectionSocket("192.168.1.102",502,a)
```

```
# java -Dnet.wimpi.modbus.debug=true \
net.wimpi.modbus.cmd.TCPSlaveTest 502
jModbus Modbus Slave (Server)
Listening...
Listenening to ServerSocket[addr=0.0.0.0/0.0.0.0,
port=0,localport=502](Port 502)
Making new connection Socket[addr=/192.168.1.101,
port=60348,localport=502]
java.lang.IndexOutOfBoundsException
        at java.io.BufferedInputStream.read(Unknown
Source)
        at java.io.DataInputStream.read(Unknown Source)
        at net.wimpi.modbus.io.ModbusTCPTransport.
readRequest(ModbusTCPTransport.java:131)
...
```

Figure 8. Sending a badly-formed packet from a client to a Modbus/TCP server

sent. When the jamod server receives the bad packet, it presents several error messages with software exceptions. The same idea can be used to drive tests for Modbus interface cards in automation devices such as PLCs.

The tool can be also used for spoof client and server messages. Using Scapy, e.g., the user is able to perform traffic capture and, if a request function is detected, our tool is able to **inject false responses**, pretending to be a real Modbus/TCP server. Furthermore, the user is able to **inject false requests** to the network, pretending to be a real Modbus/TCP client.

6 Conclusions

This paper presents a tool for Modbus/TCP packet manipulation based on Scapy. Using this tool, the user is able to inject Modbus/TCP traffic without any concern with the network programming. This feature allows the user to focus his attention on the tests that he wants to perform. The proposed tool can be seen as a first Scapy extension for industrial protocols and can be used to build and perform several security tests.

A more complex testbed is being implemented at our laboratory using several devices that are common in automation environments. Our tool will be used to perform more complex tests that will include malformed packets, stressing and system behavior in adverse conditions. We hope to address experiments that will help professionals to better understand some security aspects of industrial devices and networks.

Future works includes the implementation of other industrial protocols and use our tool to analyse security vulnerabilities in these protocols.

7 Acknowledgments

The authors would like to thank the Department of Computer Engineering and Automation (DCA), Federal University of Rio Grande do Norte (UFRN) and REDIC (Instrumentation and Control Research Network) for the support received during the development of this work.

References

[1] jamod. `http://jamod.sourceforge.net/`.

[2] Modbus-ida. `http://www.modbus.org`.

[3] P. Biondi. Scapy. `http://www.secdev.org/projects/scapy/`.

[4] E. Byres, J. Carter, A. Elramly, and D. D. Hoffman. Worlds in collision - ethernet and the factory floor, December 2002.

[5] B. Cato. Pacgen: Where do you want to send packets today?, February 2007. `http://pacgen.sourceforge.net/`.

[6] D. Dzung, M. Naedele, T. V. Hoff, and M. Crevatin. Security for industrial communication systems. In *Proceedings of the IEEE*, volume 93, pages 1152–1177, June 2005.

[7] F. S. Foundation. Gnu general public license. `http://www.gnu.org/copyleft/gpl.html`.

[8] M. Freeman. Network security - who's responsible? [ethernet in plant floor automation applications]. *IEEE Review*, 50:53, November 2004.

[9] Modbus-IDA.org. *Modbus Application Protocol Specification V1.1a*, June 2004. `http://www.modbus.org/docs/Modbus\_Application\_Protocol\_V1\_1b.pdf`.

[10] J. Nathan. Nemesis packet injection tool suite, February 2007. `http://nemesis.sourceforge.net/`.

[11] P. S. M. Pires and L. A. H. G. Oliveira. Security aspects of scada and corporate network interconnection: An overview. *Proceedings of the International Conference on Dependability of Computer Systems (DEPCOS-RELCOMEX'06)*, pages 127–134, May 2006.

[12] S. Sanfilippo. Hping - active network security tool, February 2007. `http://www.hping.org/`.

[13] A. Treytl, T. Sauter, and C. Schwaiger. Security measures for industrial fieldbus systems - state of the art and solutions for ip-based approaches. *Proceedings International Workshop on Factory Communication Systems*, pages 201–209, September 2004.