A Modbus Traffic Generator for Evaluating the Security of SCADA Systems

Rami Al-Dalky, Omar Abduljaleel, Khaled Salah, Hadi Otrok and Mahmoud Al-Qutayri
Electrical and Computer Engineering Department
Khalifa University of Science, Technology and Research, UAE

***Abstract*— Supervisory control and data acquisition (SCADA) systems are used to monitor and control several industrial functions such as: oil & gas, electricity, water, nuclear fusion, etc. Recently, the Internet connectivity to SCADA systems introduced new vulnerabilities to these systems and made it a target for immense amount of attacks. In the literature, several solutions have been developed to secure SCADA systems; however; the literature is lacking work directed at the development of tools to evaluate the effectiveness of such solutions. An essential requirement of such tools is the generation of normal and malicious SCADA traffic. In this paper, we present an automated tool to generate a malicious SCADA traffic to be used to evaluate such systems. We consider the traffic generation of the popular SCADA Modbus protocol. The characteristics of the generated traffic are derived from Snort network intrusion detection system (NIDS) Modbus rules. The tool uses Scapy to generate packets based on the extracted traffic features. We present the testing results for our tool. The tool is used to read a Snort rule file that contains Modbus rules to extract the required traffic features.**

Keywords – SCADA System; Modbus; Snort; Scapy; Network security.

I. INTRODUCTION

Supervisory control and data acquisition (SCADA) systems were first used in 1960s to monitor and control equipment not only in industrial fields such as: oil & gas, water & wastewater, electrical power transmission & distribution, but also in some experimental facilities such as nuclear fusion. In the past, SCADA systems were operated in isolated environments and rarely sharing information with other systems outside its networks. As Internet and Internet-based technologies started to be integrated into SCADA systems, such systems are now exposed to an immense amount of cyber-attacks and threats. This was evident in the recent sophisticated attacks on Saudi Aramco and Qatar RasGas, and previously on the uranium enrichment facility in Natanz, Iran, and oil company Chevron, USA [1]. To detect and deter such attacks effectively and in real time, several solutions are now being harnessed for securing SCADA systems. However, these solutions are required to be evaluated in testing environment to ensure its effectiveness in securing SCADA systems.

Several research projects at various universities have been focused on building testbeds for SCADA cyber security evaluation. The PowerCyber testbed [2] was developed at Iowa State University to provide a real-world SCADA system that consists of hardware, software and field devices for SCADA cyber security evaluation. Another testbed was developed at University of Arizona called TASSCS [3]. This testbed uses OPNET simulation tool for network emulation and PowerWorld simulation tool to simulate an electric grid to evaluate the effectiveness of several techniques used to secure SCADA systems. The SCADASim testbed [4] was developed on the top of OMNET++ at Royal Melbourne Institute of Technology to evaluate the performance of a system under attack. This testbed allows the integration of the simulation environment with real world field devices such as remote terminal units (RTU) and programmable logic controllers (PLC). Moreover, there is a tool written in Python called ModScan [5]. This tool is a network mapping tool that is used to scan SCADA Modbus/TCP based network. The works presented above did not propose a malicious traffic that can be used for testing the effectiveness of SCADA security solutions.

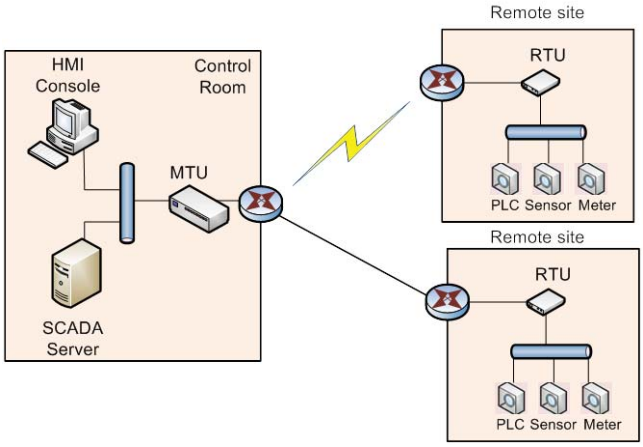

Figure 1. A typical SCADA system

To date, the literature is lacking an automated traffic generator for SCADA systems. In this paper, we describe and show how to develop an automated tool to generate a malicious Modbus traffic from Snort network intrusion detection system (NIDS) rules using Scapy to evaluate the effectiveness of the solutions to secure SCADA systems. We only generate Modbus traffic because Modbus protocol is considered the most popular industrial protocol being used today. The characteristics of a malicious traffic are defined from Snort rules database, where the tool automatically extracts the traffic features, stack them into packets and generates a malicious Modbus packets using

978-1-4799-2581-0/14/$31.00 ©2014 IEEE

Scapy. Our tool is available online and can be downloadable from [6].

The remainder of the paper is organized as follows. Section II represents background information on SCADA. Section III gives a brief overview of Snort NIDS. Section IV provides a general background about Scapy. Section V describes the implementation algorithm. Section VI describes the experimental setup to test our tool. Section VII shows the results of the testing. Finally, Section VIII concludes the paper.

II. BACKGROUND

A typical SCADA system consists of four major elements [7] as shown in Figure 1. The RTUs are communication devices located in remote sites. These devices consist of a central processor, set of Input/output modules and communication devices to be connected to field devices (actuators, meters, sensor, etc.) within SCADA system. Each RTU collects information from field into its memory until a master terminal unit (MTU) requests this information to be processed. Moreover, each RTU expects to receive commands from MTU and takes action based on it (e.g. switch off the transmission line). SCADA system has one or more MTUs that are used to monitor and control a large number of RTUs. MTUs are considered the heart of SCADA systems and interfaced with human machine interface (HMI).

HMI is a visualization method of the system which presents the processed data to the operator to monitor and understand the current status of the system. In addition, it allows the operator to send commands to RTUs through MTUs, views critical condition, alerts and warnings, analyzes, and archives the data that has been received. The communication infrastructure provides a connection between MTUs and RTUs. This is conducted by either wired links (Frame Relay, Ethernet, fiber optics, ATM) or wireless links (point-to-point microwave, satellite, license free spread spectrum radio).

Two of the most used protocols for communication between MTUs and RTUs are Modbus and Distributed Network Protocol (DNP3) [8, 9]. Modbus was designed by Modicon in 1979 as a serial communication protocol to communicate with industrial PLC devices over twisted pair wires. Modbus is a simple and rugged application layer protocol that has become the de-facto standard of the process control and automation industry. In SCADA system, Modbus is a master-slave protocol that allows one master (MTU or RTU) to communicate with several slaves (field devices). There are three types of Modbus that are used these days [10]: Modbus ASCII, Modbus RTU, and Modbus/TCP. In Modbus ASCII, the messages are coded in hexadecimal. It is considered the slowest one but is suitable for telephone modem or radio links. In Modbus RTU, the messages are coded in binary and it is ideal to be used over RS232.

In Modbus/TCP, the communication between the masters and the slaves uses IP addresses instead of devices addresses. As shown in Figure 2, the messages in Modbus/TCP are defined as a specific protocol data unit (PDU) frames that contain a function code of the action to be performed and the data related to this action [11]. Modbus application protocol header (MBAP) is a 7-Byte header which is used on TCP/IP to identify the Modbus application data unit (ADU). The MBAP header consists of the following fields: transaction identifier, protocol identifier, length, and unit identifier. The Modbus slaves use the transaction identifier (which is a 2 byte field) for transaction pairing by copying the transaction identifier of the request into the response transaction. The protocol identifier field (which is a 2 byte field) is set to 0 to identify Modbus protocol. The length field (which is a 2 byte field) is a byte count of the following fields, including the Unit Identifier and data fields. The unit identifier field (which is a 1 byte field) provides the address of a Modbus serial line slave that must be accessed through a gateway. After this, the ADU will be encapsulated in TCP/IP packets using TCP port number 502 which is registered by Modbus Org. for this purpose. DNP3 is an open standard, multi-layered protocol that was developed in 1993 for electrical power industry in North America [12]. It is a master-slave protocol to manage the communication between MTU and one or more RTUs. Modbus is more popular than DNP3 because it is simple, inexpensive, almost all equipment vendors continue to support it in their new products, and can run virtually over all communication media (telephone modems, microwave, fiber optics, etc.).

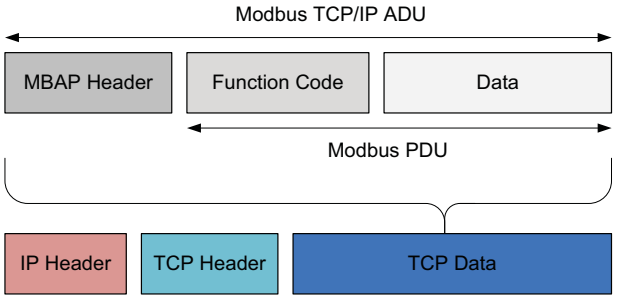

Figure 2. Modbus PDU over TCP/IP

III. SNORT NIDS

Snort is a light-weight open-source NIDS which is widely deployed and used. In fact, it is the most widely deployed intrusion detection technology worldwide, with millions of downloads to date, and it has become the de facto standard for the industry [13]. Snort is typically a PC-based NIDS, but also has been integrated in third-party solutions. Most of today's commercial IDSs are built on Snort. Snort can be configured to operate in three modes: sniffer, packet logger, and NIDS. Snort's popularity comes from operating as NIDS. Snort consists of five components [14]: Packet Decoder, Preprocessor, Detection Engine, Logging and Alerting System, and Output Module.

These components together allow Snort to detect an attack and generate alerts in specific format from the detection system. The Packet Decoder prepares the incoming packet for processing by the detection engine. The Preprocessor normalizes the decoded packets and makes it easier for Snort to digest. Also, it can classify, alert or drop packets before forward them to detection engine. The Detection Engine is the most important component of Snort. It checks a packet's header as well as its payload against thousands of rules which represent well-known attacks. The Logging and Alerting System along with Output Module generate alerts and logs based on the findings of detection engine.

Snort maintains a list of rules that represents well-known attacks. Snort uses a simple lightweight rules description language that is flexible and quite powerful. Below is a sample format of Snort rule:

alert tcp any any -> any 502 (msg:"SCADA Modbus write multiple coils - too many outputs"; content:"|0F|"; depth:3; offset:7; sid: 15447.

Snort rules are divided into two parts [14]: rule header and rule options. The rule header consists of rule's action, protocol, source IP, source port, destination IP, and destination port; respectively. There are five actions in Snort rules: alert, log, pass, activate, and dynamic. The protocol field in Snort rule can be one of the following protocols: IP, TCP, UDP, and ICMP. The rule options form the heart of Snort's intrusion detection engine. There are so many rule options that are separated from each other using semicolon. In this paper, we are considering the following major options: *msg*, *content*, *depth*, *offset*, and *sid*. The *msg* option represents the message to print by the logging and alert engine when the rule is matched. The *content* option contains string and binary data to be searched in the packet payload and trigger the rule in case of matching. In the rule represented above, the content to be searched is '0F'. The *offset* and *depth* options are used along the content option to provide more specification for the searching of the pattern in the payload. The *offset* option specifies where to start searching for a pattern in the packet payload. The *depth* option determines how far into a packet payload Snort should search for specific pattern. In the rule represented above, the *offset* and the *depth* specifies that Snort should search for the pattern after the first 7 bytes within the next 3 bytes. The *sid* option is used as a unique identifier for Snort rules.

IV. SCAPY TOOL

Scapy is a powerful interactive packet manipulation tool created by Philippe Biondi [15]. Scapy is a Python interpreter that is able to create, manipulate, or decode packets on the network, to capture packets and analyze them. It also allows you to inject packets into the network and supports a wide number of network protocols. There are two main things that Scapy can do: sending packets and receiving answers. You can define a set of packets; send them, receives answers, and match requests with answers as a list of packet couples (request, answer). Scapy uses the python interpreter as a command board, which means that you can directly use python language to create layered network protocol packets, send those packets out, and receive intelligible responses. Scapy can support Modbus protocol using Modbus extension library [16] that can be imported along with Scapy library from any Python code without extending the run-time environment. The way Scapy uses the Python interpreter makes it easier to extend the functionality of Scapy by adding custom code (without any alteration for the Python interpreter or the Scapy supersystem) to add control system and industrial protocol specific functionality.

Scapy does not only perform the jobs done by many network tools, such as Nmap, Hping, ARP scan, and TShark but also carries out a lot of specific tasks that most other tools can't handle, like sending invalid frames or combining techniques such as: VLAN hopping with ARP cache poisoning. In addition, Scapy has an advantage over other tools (like Nmap) where the answer of specific request is not reduced to open, closed, or filtered, but is the whole packet response. In summary, Scapy has a flexible model that tries to avoid the limitations of other tools, where you can add any value you want in any field, and stack them as you want and send the packet.

V. SCADA MALICIOUS TRAFFIC GENERATOR

The algorithm which is developed and implemented in Python using Scapy libraries to generate Modbus traffic from Snort rule is described in this section in details. First, the algorithm reads SCADA rules from Snort rules file checking for Modbus-related rules (lines 2-5). This is done by checking the protocol id and the destination port in each rule's header. If the protocol is TCP and the destination port is *502* (which is the port registered for Modbus), then this rule is a Modbus rule. If the rule is not Modbus, it will be ignored and the algorithm will read the next rule. If the rule is Modbus, the rule will be passed to *extract_info* function to extract the rule's header and options (lines 6-7).

Algorithm 1: the Modbus traffic generator tool

Input: Snort rules file
Output : Modbus/TCP packet that will trigger a Snort alert

```
1. for each rule r in file.rules do
2.     if  r is not a Modbus-related rule
3.         Ignore// ignore the rule and check the next one
4.         back to the loop
5.     end if
6.     Extract rule header
7.     Extract rule options
8.     Prepare MBAP & Modbus header
9.     if the content of r is empty  then
10.         Generate MBAP & Modbus headers //using default values
11.    end if
12.    else if  the content is not empty and offset <7  then
13.        //change the specified bytes in the headers
14.        modify MBAP header fields using options content
15.        if  offset + content.length > MBAP.length then
16.            Use the rest of content bytes in the next header fields
17.        end if
18.    end if
19.    else if the content is not empty and  7<= offset <=12 then
20.        //if true then modify Modbus Header fields
21.        Modify Modbus header fields using options content
22.        if  content.length  > Modbus.length then
23.            add the rest of content bytes to the payload
24.        end if
25.    end if
26.    else
27.        // offset > 12 then add  content bytes to Modbus payload
28.        Append content as payload to Modbus
29.    end
30.    Attach MBAP to the Modbus header
31.    Generate TCP & IP headers  //using the rule header information
32.    Attach Modbus to the TCP  header  //encapsulate MBAP in TCP
33.    Attach the TCP to the IP header  //encapsulate TCP in IP
34.    Establish TCPSession
35.    Send packet
36.    Close session
37.  end for
```

For the rule options, we are mainly interested in two options: *content* and *offset*. When *extract_info* function splits the rule into header and options, it parses the rule options based on the semicolon that separates the rule options as mentioned before. Each extracted rule option is stored in a string and the resultant strings are saved in an object of a class that we defined in Python to hold these values to be used later. After extracting the rule's header and options, the algorithm prepares the MBAP and Modbus headers (line 8). Initially, the two headers are built with

default values that we assumed. The MBAP header default values are: *0x00001* for transaction identifier field, *0x0000* for protocol identifier field (to identify Modbus protocol), and *0x00* for unit identifier field. For Modbus header the default value are: *0x03* (read holding registers) for function code field, *0x0000* for start address field, *0x0001* for quantity field and none for Modbus payload.

Next, the algorithm modifies both MBAP and Modbus headers. That is, the default values get changed to match the extracted rule options (lines 9-29) that are stored in the rule object. First, the algorithm checks whether the *content* option is empty or not. If it is empty, then no changes are required to both headers' default values (lines 9-11). However, if the content is not empty, then the algorithm checks the *offset* option to find which fields need to be changed in the both headers. As stated before, the Mobdus header is encapsulated in the MBAP, where the first 7 bytes in the TCP payload refer to the MBAP. Therefore, if the *offset* value is less than 7, the algorithm will make a change in the MBAP header or in the both headers depends on the *offset* and the *content* length (lines 12-18). For instance, in the rule below, the *offset* value is 6 and the *content* length is 2. Therefore, the value of the 7th byte in the MBAP header (which is the unit identifier) will to be changed to the first byte of the content ('03'). However, the *content* length plus the *offset* is greater than the length of the MBAP. This means that, the second byte of the *content* will replace the first byte in the Modbus header (which is the function code field).

alert tcp 10.20.30.40 ANY -> $HOME_NET 502 (msg:"demo alert"; content:"|03 05|"; offset:6)

If the *offset* value is between 7 and 12, then the algorithm will only make a change to the Modbus header based on the *content* length and the *offset* value (lines 19-25). If the *content* length and the *offset* value are within the range of Modbus header, then the *content* value will be added to the Modbus header. However, if the *content* length and the *offset* value are partially within the range of Modbus header, then part of the *content* will be added to Modbus header and the rest will be added to the payload. In case of having the *offset* value greater than 12, then whole *content* value will be appended to the payload of the Modbus packet (lines 26-29).

After preparing the MBAP and Modbus headers, the *send* function is called to prepare the packet to be sent. Firstly, it attaches the Modbus PDU to the MBAP header to create Modbus TCP/IP ADU. Then, the Modbus ADU will be encapsulated in a TCP/IP packet which is generated using the rule's header information obtained from *extract_info* function (lines 30-33). At this step, the packet is built and ready to be sent. The *send* function establishes a TCP session with the receiver, sends the generated packet, waits for acknowledgments, and closes the session (lines 34-36). The process will start again by reading the next rule in the file.

Figure 3 shows a snapshot of the tool's web page [6]. The web page contains a description about the tool and the requirements to run the tool. The page also contains a file that provides a step by step procedure to run the tool and the commands that are required to generate the required traffic.

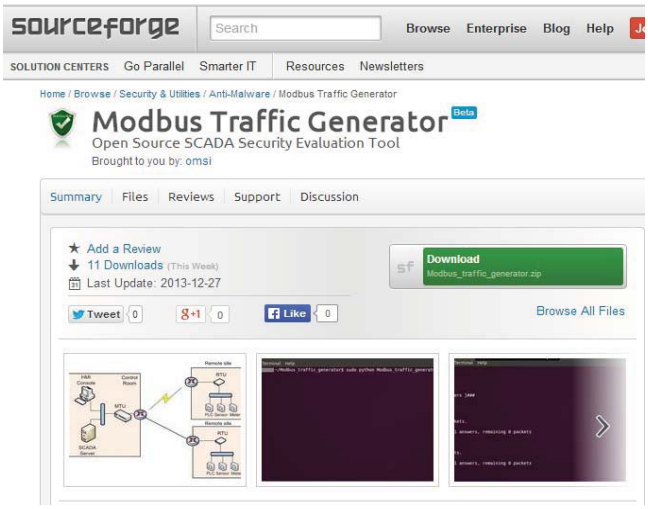

Figure 3. Homepage of the tool

VI. EXPERIMENTAL SETUP

To test the functionality of our tool, we performed an experiment to generate Modbus/TCP traffic from Snort rules using our tool and test the ability of the generated traffic in triggering the Snort rules that was used to generate this traffic. As shown in Figure 4, the testbed comprises three machines: sender, receiver and Snort NIDS. The three machines are connected with 1 Gbps links to 10/100/1000Base-T Dualcomm Ethernet Switch Network TAP [17]. The Ethernet tap has a hard-wired mirroring port that send a copy of network packets seen on port 1 to port 5. We connected Snort NIDS machine to the mirroring port to monitor the traffic between the sender and the receiver. Table I specifies some parameters used in our experiment.

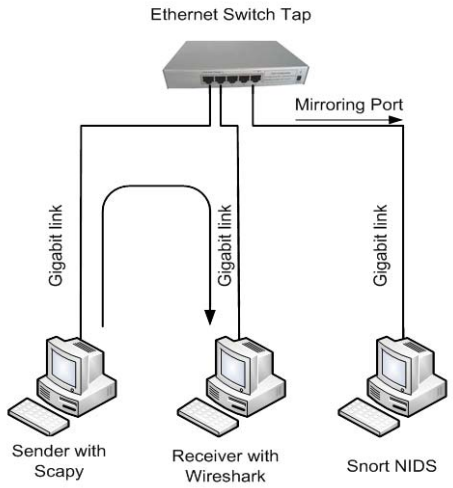

Figure 4. Testbed setup

TABLE I. EXPERIMENT PARAMETERS

Parameter	Value
Machine Types	HP Z400 Workstation
CPU	Intel Xeon Quad-Core 3.2GHz
RAM	8 GB
OS	Ubuntu 10.10, Fedora 14
NIC	Broadcom NetXtreme Gigabit Ethernet Plus

The sender machine has our tool along with Snort rule file and Scapy to generate the traffic. At the receiver, we used Netcat command line to listen on port number 502. The command with its options: *nc –l –k 502*. The –l option is used to specify that nc should listen for an incoming connection on the specified port rather than initiate a connection to a remote host. The –k option is used to force nc to stay listening on the specified port for another connection even though its current connection is completed.

VII. EXPERIMENTAL RESULTS

To test Modbus traffic generator tool, we have added three Modbus Snort rules to the input file that is used by the tool to extract the traffic characteristics. Figure 5 below shows the rules. The tool reads the rules consequently and generates malicious Modbus packets that trigger these rules at Snort. For instance, let's take the first rule in the input file. The rule will be triggered at Snort machine when Snort detects a Modbus packet in which the 9th byte of the TCP payload (the starting address or the reference number in the Modbus header) has the value '0x09'. Wireshark [18] has been used at the receiver machine to verify the generated packets.

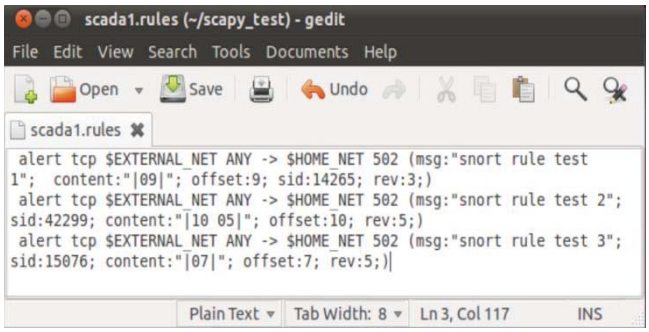

Figure 5. Modbus traffic generator input file

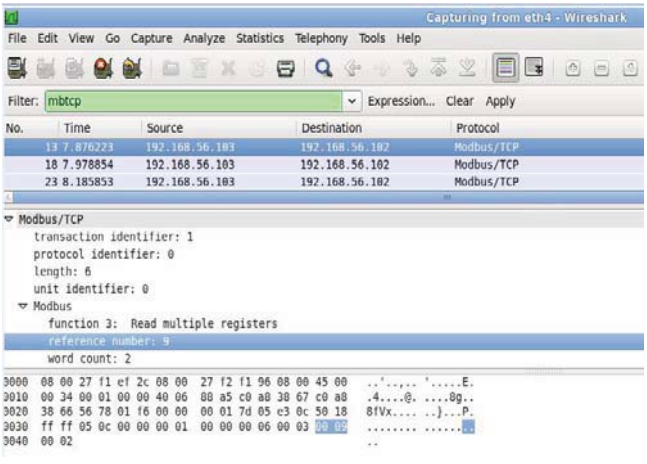

Figure 6. Modbus packet captured at the receiver using Wireshark

Figure 6 shows a snapshot of Wireshark at the receiver machine; we used mbtcp key word in the filter to display Modbus/TCP packets. In the figure, we can notice that Wireshark has detected the Modbus/TCP packet with the default MBAP header values. The 9th byte of the TCP payload which is highlighted in blue has the value '0x09'. Once the packets are sent from the sender to the receiver, the Ethernet switch tap sends a copy of network packets from port 1 to port 5. Snort analyzes the incoming packets, looking for well-defined signatures that represent malicious packets. As shown in Figure 7, the incoming packets have triggered three Snort rules which are the same rules that have been used by the tool to generate the malicious Modbus packets. To confirm that the triggered rules are the same rules that have been used by the tool, we can compare the sid and the msg in Snort's alerts with the sid and the msg of the rules at the sender side. For example, in the input file, the first rule's sid is 14265 and the msg is 'snort rule test' which are matching the ones in the alerts. This means that the tool was successfully generating malicious Modbus packets that triggered Snort rules.

Figure 7. Alerts generated by Snort for the incoming packets

VIII. CONCLUSION

In this paper, we have described in adequate detail a novel tool that can be extremely useful in evaluating the security of SCADA systems. Our tool has the ability to generate malicious SCADA traffic of Modbus protocol type. The malicious traffic is generated based on Snort rules which are used to trigger malicious packets. From the Snort rules, our tool automatically extracts the traffic features, stacks them into packets and generates a malicious Modbus packets using Scapy. Our tool successfully generates malicious SCADA traffic that triggered the exact Snort NIDS rules that was used to generate this traffic. As a future study, we plan to extend our tool to generate other types of SCADA protocols as that of DNP3.

REFERENCES

[1] R. Axelrod, R. Iliev, "The Strategic Timing of Cyber Exploits," In APSA 2013 Annual Meeting Paper (2013).

[2] A. Hahn, B. Kregel, M. Govindarasu, J. Fitzpatrick, R. Adnan, S. Sridhar,and M. Higdon, "Development of the PowerCyber SCADA security testbed," in Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research - CSIIRW '10, p. 1, 2010.

[3] M. Mallouhi, Y. Al-Nashif, D. Cox, T. Chadaga, and S. Hariri, "A Testbed for Analyzing Security of SCADA Control Systems (TASSCS)," In Innovative Smart Grid Technologies (ISGT), 2011 IEEE PES, pp. 1-7. IEEE, 2011.

[4] C. Queiroz, A. Mahmood, and Z. Tari, "SCADASim—A Framework for Building SCADA Simulations," IEEE Transactions on Smart Grid,vol. 2, no. 4 (2011): 589-597.

[5] M. Bristow, "ModScan: a SCADA Modbus network scanner," Available at https://code.google.com/p/modscan/

[6] "Modbus Traffic Generator", Available at http://sourceforge.net/projects/modbus-traffic-generator.

[7] D. Choi, S. Lee, D. Won, and S. Kim, "Efficient Secure Group Communications for SCADA," IEEE Transactions on Power Delivery, vol.25, no.2, pp.714,722, April 2010.

[8] K. Stouffer, J. Falco, and K. Scarfone, "Guide to Industrial Control Systems (ICS) Security," NIST Special Publication 800-82 – Recommendations of the National Institute of Standards and Technology, June 2011.

[9] J. Verba, "Idaho National Laboratory Supervisory Control and Data Acquisition Intrusion Detection System (SCADA IDS)," In the Proceedings of the 2008 IEEE Conference on Technologies for Homeland Security, Boston, May 12-13, 2008, pp. 469-473.

[10] P. Dao-gang, Z. Hao, Y. Li , and L. Hui, "Design and Realization of Modbus Protocol Based on Embedded Linux System," The 2008 International Conference on Embedded Software and Systems Symposia. July 29-31, 2008, pp.275,280.

[11] P. Huitsing, R. Chandia, M. Papa, and S. Shenoi, "Attack Taxonomies for the Modbus Protocols," International Journal of Critical Infrastructure Protection, Volume 1, Pages 37-44, December 2008.

[12] G. Clarke, and D. Reynolds, "Practical Modern SCADA Protocols: DNP3, IEC 60870.5 and Related Systems," Newnes, Oxford, United Kingdom, 2004.

[13] M. Roesch, "Snort – Lightweight Intrusion Detection for Networks," Proceedings of USENIX LISA'99, November 1999.

[14] M. Roesch. "Snort – Lightweight Intrusion Detection for Networks," Proceedings of USENIX LISA'99, November 1999.

[15] P. Biondi, "Scapy, a powerful interactive packet manipulation program," Available at http://www.secdev.org/projects/scapy/

[16] "Modbus TCP packet library," Avaiable at https://www.scadaforce.com/modbus

[17] "Network Tap," Available at http://www.dual-comm.com/

[18] G. Combs, "Wireshark network protocol analyzer," Available at http://www.wireshark.org/