Statistical analysis of SCADA network communications

Francis Deslauriers, Antoine Lemay, Étienne Ducharme, José Fernandez

Abstract—Applying notions from intrusion detection system, we built a tool named `ScadaAnalyzer` that learns from a network usual traffic and can determine if specific capture fits with the model distribution. Our SCADA packet analyzer takes advantage of the highly periodic behavior of the DNP3 protocol to detect unusual communication between hosts. This tool uses the Kolmogorov-Smirnov statistical test to compare a model distribution and a test sample. Gathering positive preliminary results during test runs suggest a safer future for SCADA computer security.

Index Terms—Traffic Analysis, SCADA, Intrusion Detection System.

1 Introduction

The electrical grid is a set of highly critical equipment spread across a large geographical area and therefore is complex to manage efficiently. With the increasing use of computer-controled devices on the grid, operators can monitor the state of the network from remote location. For example, operators can query for the state of a particular sensor on the network or send request to change its state without having a direct interaction with the device.

With the automation of the Grid comes the cyber-threats on the control network. The lack of suffisiant security on those networks would put a large number of people at risk. In cold countries, a failure of the electrical grid which powers most of the heating would have catastrophic consequences.

This article introduce `ScadaAnalyzer`, a tool developed to detect abnormal interactions between hosts on a control network. Having a small amount of noise on such network makes statistical analysis a posibility. This tool uses features from control network protocol to perform statistical analysis in order to detect intrusions and infected hosts.

• *F. Deslauriers is a undergraduate computer engineering student at Polytechnique Montréal, Montréal, Qc*
E-mail: francis.deslauriers@polymtl.ca

2 Related work

This section includes a brief introduction to SCADA networks and explains the challenges that the industry is facing regarding computer security. It also dicuss how specific features of SCADA networks can be used to detect threats and anomalies.

2.1 SCADA Networks

SCADA networks are cyber physical networks designed for Supervisory Control And Data Acquisition. This type of network is used to remotely control industrial systems such as circuit breakers, valves or robots[1]. Operators can have a real-time overview of the state of the network in order to adjust and react to measurements and alerts from all across the network. Critical control networks such as the electrical power grid are controlled and monitored by such system. An important advantage of using control networks consists of being able to affect the state of the network remotely. Also, the use of TCP/IP stack over the internet means that the power flow can be controlled and monitored without physical access to the components[1]. Typically, a SCADA is formed by multiple devices spread all accross the network forming a logical tree. The root of this tree is the Master Terminal Unit (MTU) where all data and commands are sent to and from. MTUs are in charge of a large geographical section of the grid and use polling techniques to gather data from the leaves of that logical tree. The Remote Terminal Units (RTU) are the field components that retrieve the data from the sensors and forward it to their MTU.

A major drawback of using TCP/IP stack on this type of network is that the well-known internet vulnerabilities are easily transferable to SCADA networks context. An attacker controlling even a single host can have major impacts on populations. A good example of what is at stake when we talk about security on a SCADA network is the worm unveiled in 2010 by Symantec, Stuxnet [2]. Stuxnet was specially designed to target Industrial Control Systems(ICS) in infrastructures such as power plants. Intrusion Detection Systems have an important role in these cases, but those tools may be hard to transfer to SCADA networks because of the highly specialized attackers targeting control networks. It is important to realize that traditionnal reaction to intrusion may not be applicable to highly critical networks. For exemple, an operator can not shutdown the electrical distribution network without having a major impact on the customers powered by this network.

There are multiple protocols used for communication between devices on a control network. An important protocol used in North American SCADA networks is Distributed Network Protocol version 3 (DNP3)[1]. We focused our effort on this protocol for our analysis and experimentation. The use of this particular protocol has an important impact on the traffic. For example, the master-slave feature of this protocol has a noticeable effect on the topology of the communications [3]. Also, DNP3 use periodic pooling from the masters as well as exceptions reporting from the slaves[1].

2.2 Feature detection

As mentionned earlier, we focused our analysis on the DNP3 network protocol but most of the ideas used here also apply to other control protocols such as Modbus. The use of the DNP3 protocol affects the way hosts on a network interact with each other. As Lemay [1] has shown in his thesis, those interactions occur in a very regular fashion. Also the fact that the DNP3 protocol makes use of a slave-master hierarchy is really specific to control networks. These two important characteristics help us with the fact that unusual behavior is easily detectable on this type of network. Here we will explain how we used three features of SCADA networks to detect possible cyber-attackers or infected hosts. In this section, we will discuss the features that Lemay studied.

2.2.1 Topology

On most SCADA networks, the server-client model does not stand. As mentioned before, DNP3 protocol works on a Master-Slave hierarchy. This feature of SCADA networks makes it really easy to detect abnormal behaviour. In really few occasions will an RTU initiate a conversation with a MTU without being previously polled. It would be really suspicious behavior for a RTU to send a large number of packets to the MTU, other RTUs and even hosts outside the network. This makes the network topology a good feature to test in order to detect infected units in a control network.

2.2.2 Packet Length

Another effect of the polling characteristic of most SCADA networks is that there is a really low variation of the packet size. When polling for data from the sensors, the packet containing the request is likely to be the same each time and the same applies to the response from the RTU. Recording an important change in the size of the packets travelling on the network would be a good indicator that an host is compromised.

2.2.3 Interdeparture Time

The last feature of SCADA control networks that was considered during this experimentation is the interdeparture time of packets from a particular host. We are still basing our thinking on the regularity mentioned earlier. Since control networks are most of the time used for data gathering and command sending to the RTUs the departure pattern is quite regular. For example, when polling a particular sensor a MTU will first send a polling request then receive the response and finally send an acknowledgement. The inter-departure time between the request and the acknowledgement will be stable over time since network has a really low traffic. Also, the fact that polling is done at a particular frequency makes this feature easily detectable when comparing two network packet capture. Figure 1 from Lemay's Ph.D. thesis is showing the average interdepature time distribution from the RTUs. We can clearly see that 50% of the interdepature time is distributed between about three timings.

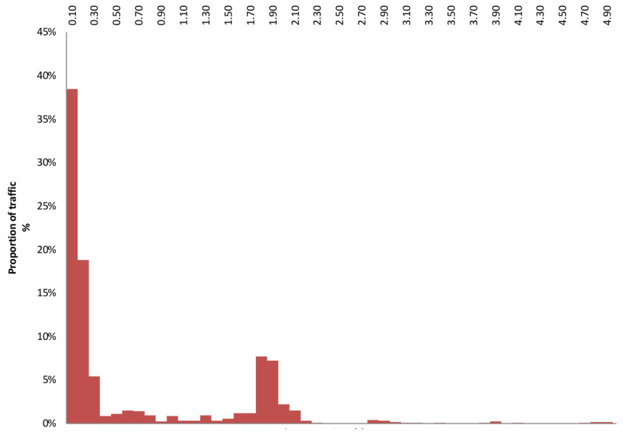

Fig. 1: Average interdepature time from RTU [1]

3 Model

As we explained in section 2.2, some patterns of communication between hosts on a DNP3.0 network are happening in a really tight distribution. It is this distribution shape that was used in our tool to detect threat on the network. In this section, we will explained how we understood the problem and how we solved it.

3.1 Challenge

With that tool we are trying to automate the detection of the distribution of specific features that are outside the regular state of the network. As we saw from the work of Lemay, the distributions are far from the normal distribution which means that we had to find a statistical test that does not make any assumption on the shape of the distribution. This tool must first learn how the hosts on the network normally behave and it is not possible to use a model from another network because the technical configuration might be different from a network to another. We also want to create a tool that can easily be upgraded and adapted to the situation. According to the network, features must be easily added, modified or removed.

3.2 Solving the problem

While analyzing the problem, it was decided that our program should have two operation modes. First of all, there is the Learning mode. The learning mode is when the program is recording every interaction between the hosts of the network. This model is then stored as a Json file and will be considered as usual pattern for the analysis stage. Next, there is the Analyzing mode. Running in this mode compare the packet capture against the distributions recorded during the learning stage. It is possible to learn and analyze with a live capture directly from the network interface card. It is also possible to use those two modes with a packet capture file that has been previously recorded on the network. Finally, a key element of our tool was that it must be possible for the user to specify a critical value for the statistical test. If not specified, a default value is used.

In this tool, we used the Kolmogorov Smirnov test (KS test) that is test for goodness of fit. As Masset [5] discussed a test for goodness of fit ”is based on the maximum difference between an empirical and a hypothetical cumulative distribution”. In our case, we will use the term model distribution to talk about the hypothetical distribution. On the other side, the topology feature test confirms that specific topology has been seen in the model distribution without any statistical evaluation. We will run this test on each feature independantly. We want our tool to offer a good modularity and to be ready for the addition of needed feature handlers.

4 Implementation

4.1 Software structure

The idea behind our design was modularity. We wished to be able to add new test handlers as we want. During our design phase we decided to use the C++ programming language because of its powerful object oriented features and also because of our experience. In figure 2 you can see our three test handler classes that inherit from the FeatureTestHandler class. The next important part

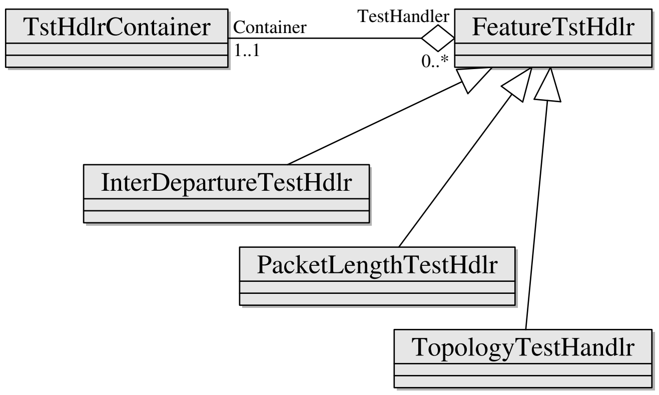

Fig. 2: Class diagram of the test Handlers and the container

is the TestHandlerContainer class that is keeping a vector FeatureTestHandler objects to use. Every test handler classes implements the FeatureTestHandler virtual interface that is shown by listing 1.

```
virtual void JsonToData(Json::Value *json) = 0;
virtual void computePacket(
             const struct pcap_pkthdr* pkthdr,
             const unsigned char * packet ) = 0;
virtual void printDistribution() const = 0;
virtual void runTest( double cAlpha) = 0;
virtual int getTestResult() = 0;
virtual Json::Value *DataToJson() const = 0;
```

Listing 1: FeatureTestHandler class virtual interface

We will walkthrough this interface and explain the use of these six methods. Some of these methods may be necessary during learning and/or analysis mode. Also, it is important to note that the TestHandlerContainer class calls these methods for each handler.

First of all, the implementation of the JsonToData method is looking for model data for a particular test handler in the model file. Each implementation of the FeatureTestHandler interface knows how to extract this data for its own feature.

Next, the computePacket method is called for each packet in the capture for each handler in the test. Each handler is aware of how to consider each packet. This method takes in argument pointers to packet header and packet structures.

Moreover, the printDistribution method is primarily for testing purposes. It is not used in production code but is really helpful during development phase.

Next, the runTest method is running the actual statistical test that will be discussed in the subsection 3.2. It takes the critical value that is to be used in that test as argument. Again, each handler is aware of how to run the test for its particular feature.

The getTestResult method then returns the results of the statistical test. In its current state, the tool is simply printing which tests have failed or passed. It would be really easy to add some other action to be taken of failed test runs. For example, we could easily send an email stating which test has failed.

Finally, the DataToJson method is to save the model to the model file after a learning session. Each implementation returns a Json document to the TestHandlerContainer which is saved in the file.

4.2 Tools & Libraries

This tool has been implemented in C++11. It is taking advantage of the many object-oriented features of this language. This software was designed with polymorphism in mind which positively impacted its modularity. In fact, it is really easy to add new features to detect and test. The user only has to implement the virtual interface defined by the FeatureTestHandler and add a newly allocated object in the TestHandlerContainer object.

Traffic analysis for security purposes has been used for several years and many tools are free available out there. When building a tool like the ScadaAnalyzer, it is important to take advantage of the work others have open sourced. This program was built using the free and open source libpcap library to help with the online and offline packet parsing[7].

A key aspect of this tool is that it records a model of what is normal on the network.

In order to save this model, we use a Json file called the `model.json`. The JSON parsing and writing of this file is done with the help of JSONcpp `C++` library[4]. It is important to note that this library is licensed under a Public domain license.

Finally, we use the C++ Boost library for file system operations[6].

5 Performance analysis

In this section, we explain our experiental methodology for testing the efficiency of our tool. We are also exposing the results of those tests.

5.1 Methodology

In order to test the suitability of our tool, we used a packet capture recorded from a simulated network. This traffic was generated in an ICS sandbox design by LEMAY [1]. The testbench was made of seven virtual machines running in VMware Workstation software. As shown in figure 3, six of these virtual machines were representating RTUs and one was the MTU. The use of a simulated network instead of a real world dataset was motivated by the limited availability of production traffic capture and the fact that we wanted to simulate real world malware. For our learning phase, we used packet capture from normal interactions on this simulated network. In order to simulated an infected host on the network, we later installed the Waledac malware on host named RLS103. This is the setup that we used in our the analysis phase. Two scenarios were executed in order to test `ScadaAnalyzer`. Firstly, the Waledac scenario where a RTU is infected with the Waledac worm. Secondly, the Advance Persistent Threat is mimicking an intrusion in a RTU by an attacker. It is important to note that we performed our tests with a confidence interval of 0.05.

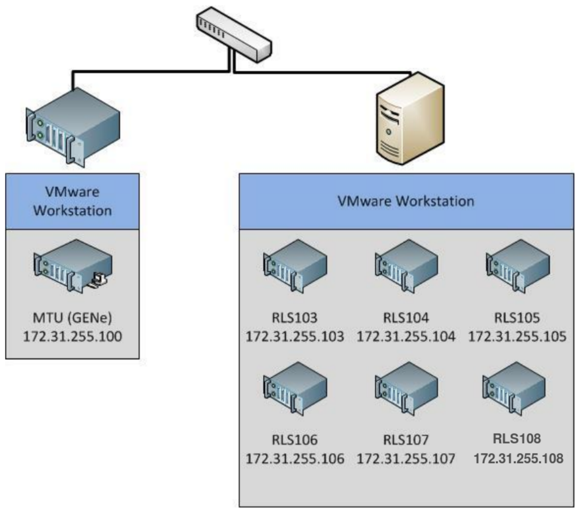

Fig. 3: Testbench network map [1]

5.2 Results

5.2.1 Topology

In table 1 shows the result of the topology test. Those are the interaction that our tool has not encountered in the model distribution. The source and destination columns are representing the source and destination hosts respectively.

TABLE 1: Unknown interaction detected

Source		Destination
RSL103	172.31.255.103	89.18.58.10
RLS103	172.31.255.103	117.102.35.90
RLS103	172.31.255.103	69.203.207.115
RLS103	172.31.255.103	83.87.159.131
RLS103	172.31.255.103	119.192.145.145

5.2.2 Packet size

Table 2 exposes the results of the packet size tests with two different samples. The dataset column represents the name of the scenario simulated in our custom sandbox. The Status column shows if the sample data matches with the model data. The status is compute from the Dstat and Critical Value columns. The column titled DStat represents the largest difference between the cumulative distributions value of the test and the model samples. The critical value is compute from the confidence interval, the size of the test sample and the size of the model sample. First, the Waledac dataset is the scenario shown in section 5.2.1 where an host has been infected by the Waledac worm. Secondly, the Advance Persistent Threat is a scenario where a cyber-attacker has been able

TABLE 2: Packet size

Dataset	Status	Dstat	Critical value (α = 0.05)
Waledac	Failed	0.032	0.026
Advance Persistent Threat (APT)	Failed	0.411	0.013

to take control of a RTU in the network. From this infected node, he is now trying to expand is control to other RTUs in the network using the tool meterpreter.

5.2.3 *Interdeparture time*

Table 3 shows the results of the interdeparture time test run. The column names have identical meaning that in table 2. This data was gathered testing the Waledac scenario against the model distribution.

TABLE 3: Interdeparture Time

Host		Status	Dstat	Critical value (α=0.05)
MTU	172.31.255.100	Failed	0.052	0.035
RLS103	172.31.255.103	Failed	0.124	0.091
RLS104	172.31.255.104	Passed	0.053	0.095
RLS105	172.31.255.105	Passed	0.081	0.094
RLS106	172.31.255.106	Passed	0.062	0.099
RLS107	172.31.255.107	Passed	0.055	0.099

6 Discussion

First, we can see from the topology test in table 1 that we detected an host at IP address 172.31.255.103 that is trying to send packets to other hosts that are not in the subnet. It is most likely the Waledac worm trying to reach its Command and Control (C&C) server. As Lemay explained, this worm has a list of peer that it needs to query in order to notify the C&C of its presence [1]. This type of behavior is easy to detect, specially when a RTU is trying to reach addresses outside of the network. Moreover, the IP addresses that RLS103 is trying to reach are spread across three continents: North America, Europe and Asia.

Secondly, we can see in table 2 that in our test with the Waledac scenario there is a 0.032 gap and that 0.026 was the critical value for that sample. Also, we can see that the Advance Persistent Threat (APT) scenario is 0.399 points out of the critical value. In both of those tests we have effectively detected the threat. From this data, it is possible see that these type of abnormal behavior can easily be detected. Those results match with those that Lemay has gathered. This easily detectable traffic is most likely due to the fact that the worm Waledac is trying to reach its C&C server with no consideration of the type of network it is in. In the same line, meterpreter software that is used in the APT scenario is trying, by default to send exploits and payload as fast as possible which would unsuual packet size.

Finally, we can see from table 3 that the distribution of the interdeparture time for the RLS103 RTU is statistically different from the model distribution. The difference is high enough to reject the null hypothesis which indicates a potential threat to the network. Also, the MTU has also failed the test with a 0.052 gap with the model which is an unexpected result. This can be explained by the incomplete model dataset. A packet capture on a longer period would help having a better idea of the of the normal behavior.

7 Conclusion

This article presented a SCADA traffic statistical analysis tool that can be used onsite or offsite to detect cyber-threats. We have successfully detected threats using features of SCADA control networks. We also had a false positive during our test runs. This false positive is probably due the limited size of the packet capture during the learning phase. Moreover, usual tools and malware were shown to be easily detectable because of the default behavior that is likely to be motivated by ”a faster infection is a better infection”. Which means that a cyber-attacker would need to put a lot of effort to modify existing tools to minic the way hosts are interacting on this specific network.

Further work could include expanding the tool with more features. For example, it would be interesting to develop a test handler that would specially detect if an host is trying to reach and address outside the control network subnet. We are already detecting this type of behavior with the topology handler but it would be useful to set a specific way to react to that really alarming event. Also, it would be interesting to specify different critical value for each test. It is easy to picture an use-case where the user would like a specific feature to be more sensible to a variation of distribution.

This tool answering to the need for a Intrusion Detection System adapted to SCADA networks. ScadaAnalyzer is highly customizable according to the needs of the user and the situation.

References

[1] A. Lemay, *Defending the SCADA Network Controlling the Electrical Grid from Advanced Persistent Threats (Ph.D. Thesis)*, Polytechnique Montréal, Québec, Canada, 2013.

[2] N. Falliere, L. O. Murchu and E. Chien, *W32.Stuxnet Dossier Version 1.4*, Symantec Security Response, 2011.

[3] G. Clark and D. Reynders, *Pratcical Modern SCADA Protocols: DNP3, IEC 60870.5 and Related Systems*, China: Newnes (Elsevier), 2008.

[4] Baptiste Lepilleur, *JSONcpp*, http://jsoncpp.sourceforge.net/

[5] Frank J. Massey, Jr.*The Kolmogorov-Smirnov Test for Goodness of Fit* Journal of the American Statistical Association Vol. 46, No. 253 (Mar., 1951), pp. 68-78

[6] Boost project, *Boost C++ Library*, http://www.boost.org/doc/libs/1_55_0/libs/filesystem/doc/index.htm

[7] LibPcap, *libpcap, a portable C/C++ library for network traffic capture.*, http://www.tcpdump.org/