Implementation of a Stateful Network Protocol Intrusion Detection Systems

S. Seng[1,2], J. Garcia-Alfaro[1] and Y. Laarouci[2]
[1]*TelecomSud Paris, Palaiseau, France*
[2]*EDF R&D, Palaiseau, France*

Keywords: Statechart, Protocol Modeling, Intrusion Detection System, Anomaly Detection, Industrial System, Critical Infrastructures.

Abstract: The deployment of a Network Intrusion Detection System (NIDS) is one of the imperatives for the control of an information system. Today, almost all intrusion detection systems are based on a static vision of network exchanges, whether for detection engines based on signatures or on behavioral models. However, this approach is limited: it does not allow to directly take into account past exchanges and thus to fully model normal or abnormal behavior, such as verifying that an authentication has taken place before authorizing a privileged request or detecting a *replay* attack. We propose to add an additional dimension to NIDS by performing stateful monitoring of communication protocols. Unified Modeling Language (UML) statecharts have been chosen to model the protocols and to perform the stateful monitoring. An implementation of this solution is integrated within an existing NIDS and validated on two industrial protocols IEC 60870-5-104 and Modbus TCP. This implementation has been realized by dissociating the stateful monitoring and the NIDS with the help of an abstraction interface allowing an easy integration of new communication protocols.

1 INTRODUCTION

1.1 Context

Faced with cybersecurity issues, the implementation of cybersecurity information systems monitoring tools is increasingly needed and even becomes a compulsory requirement. Many companies are investing in setting up a Security Operation Center (SOC), equipped with a Security Information Management System (SIEM) for the recognition and management of alerts. The origin of these alerts comes from various sensors such as Intrusion Detection Systems (IDSs).

In this paper, we focus only on Network Intrusion Detection System (NIDS). We do not distinguish on the type of detection methodology (i.e., Signature-based or Behavior-based).

1.2 Static Vision of Exchange

NIDS use filters called *dissectors* to identify and interpret the network packets they capture. These *dissectors* extract the various fields, options and structured data contained in packets. This data, once extracted, will then be transmitted to a detection engine that will determine if it is a healthy packet or a potentially malicious packet. Figure 1 depicts the idea, in which the first line represents the different stages of a NIDS.

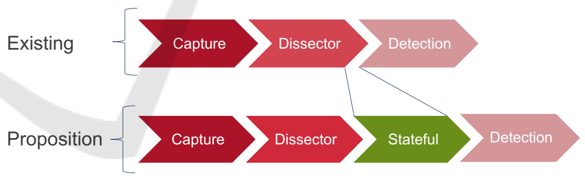

Figure 1: Addition of the stateful stage.

Almost all *dissectors* are static. They do not take into account the dynamics of network protocols. This static operation does not allow to directly take into account past exchanges, which significantly reduces the possibilities of predicting an attack. For example, it is not possible to verify that an authentication has taken place before authorizing a privileged request or to detect a *replay* attack.

1.3 Proposal and Experimentation

We propose to overcome this limitation by adding an incremental dimension to NIDS through stateful monitoring of communication protocols. As illustrated in the second row of Figure 1, the stateful monitoring

Seng, S., Garcia-Alfaro, J. and Laarouci, Y.
Implementation of a Stateful Network Protocol Intrusion Detection Systems.
DOI: 10.5220/0011327400003283
In *Proceedings of the 19th International Conference on Security and Cryptography (SECRYPT 2022)*, pages 398-405
ISBN: 978-989-758-590-6; ISSN: 2184-7711

stage is inserted after the dissectors and before the detection engine. The latter then takes advantage of the data enriched with the history of exchanges and the conformity of the packet with respect to the protocol.

Our proposed solution has a secondary goal. It has to be as agnostic as possible of use cases, i.e., it has to be as independent as possible of the NIDS by proposing an abstraction interface for the modeling of the stateful monitoring. By focusing our work on the modeling of the communication protocol and not on the modeling of the use case, we enter the field of Protocol-specification-based NIDS — which are relatively little studied (Uppuluri and Sekar, 2001). This allows us to be use case agnostic.

A prototype of our stateful monitoring solution has been developed. It is based on the open source NIDS Zeek[1] which it completes with a plugin. In addition, Harel's statecharts have been used to model the protocols and thus allow stateful monitoring. They seem to be more adapted to the modeling of complex systems than the traditional Finite State Machine (FSM). The prototype has been tested with the industrial communication protocols Modbus TCP and IEC 61870-5-104.

The rest of the paper is structured as follows. Section 2 elaborates further on our problem domain. Section 3 surveys related work. Sections 4 and 5 provide our contribution and reports experimental work. Section 6 provides some additional discussions. Section 7 concludes the work.

2 PROBLEM DOMAIN

The main objective of our work is to complement the *dissectors* of a NIDS in order to take into account the dynamic aspect of a network exchange. This requires modeling the behavior of network communication protocols. In practice, in the context of network communication protocols, this is similar to stateful monitoring.

2.1 Use and Relevance of Stateful Monitoring

Stateful monitoring of network communication protocols is not new and is already applied in several domains such as testing, simulation or compliance verification. For cybersecurity, the best known use of stateful monitoring concerns firewalls (Gouda and Liu, 2005). It allows a much finer control on network exchanges. For example, it allows to manage in a finer way the two directions of communication of protocols such as TCP or the capacity to take into account by firewalls protocols that use a dynamic allocation of network ports such as FTP in active mode or Remote Procedure Call (RPC). Still in the field of cybersecurity, stateful monitoring is also used for application or protocol fuzzing, in Host-based Intrusion Detection System (HIDS) or, like in this article, in NIDS.

The use of stateful monitoring adds a dynamic view of a system, providing an additional dimension to tools.

2.2 Modeling Usability Problem and UML Statechart Contribution

The simplest and most common method to perform stateful monitoring is based on automata, especially FSM or its derivatives (Mealy machines, nondeterministic FSM, etc.) (Qin-Cui et al., 2009; Goldenberg and Wool, 2013).

The majority of tools or studies that use stateful monitoring for communication protocols simplify the protocols or processes to avoid the combinatorial explosion of the number of states. This is notably the case of the Netfilter/iptables firewall, which only manages four states for the TCP protocol (NEW, ESTABLISHED, RELATED and INVALID), or of studies (Qin-Cui et al., 2009) that do not take into account some elements that are considered superfluous, such as the control of timeouts, re-transmissions, or some internal signals (synchronization, keep-alive, etc).

However, we consider that in a NIDS context, all elements are important and can constitute anomalies or weak signals of an attack.

This difficulty from traditional FSM for stateful monitoring is already known. In (Yu et al., 2015), the authors evoke the problem of the combinatorial explosion of the number of states and more generally the lack of ergonomics (readability and editing) of FSMs for humans. They propose to use Harel's statecharts (Harel, 1987) which offer concepts missing in FSMs such as hierarchy or parallelization which allow to significantly improve the readability of an automaton.

2.3 Importance of Stateful Monitoring for ICS

2.3.1 Higher Cybersecurity Risks and Impacts

We believe that Industrial Control System (ICS) are less well prepared to face cybersecurity attacks. Indeed, some specificities of ICS offer a greater expo-

[1]https://zeek.org

sure to cyber-attacks. First, industrial equipment designers, industrial solution integrators and operators are still not very aware of cybersecurity, which is why there are rarely effective protection measures against cybersecurity risks. Secondly, ICS are often designed for a much longer lifespan than in IT. It is common to still find ICS in operation 20 to 30 years after their initial setup. However, cybersecurity evolves quickly and requires regular software and hardware updates. But the availability of ICS is often a more important criterion than for IT, the updates of ICS are often grouped during the planned maintenance operations. Thus, a critical vulnerability on a system can sometimes be fixed several months, or even years, after the publication of a patch. This is even more true for critical ICS where a hardware or software update can jeopardize safety qualifications. In these cases, operational safety has priority over cybersecurity, and operators are reluctant to perform updates. Finally, attacks on ICS and especially critical ICS, due to their interaction with the physical world, can have financial, environmental and even human impacts that are much more significant than in IT. All these elements imply that the need for monitoring ICS is probably more important than for IT.

2.3.2 Effective Monitoring

On another level, some specificities about ICS seem favorable to monitoring solutions. Indeed, compared to IT systems, ICS do not evolve much. They have equipment, especially Programmable Logic Controllers (PLC), that are deterministic in their operations. This provides industrial communication protocols with interesting properties for network monitoring (Mitchell and Chen, 2014):

- *relatively* simple protocols;
- deterministic communication, based on iterative and continuous polling between, for example, a PLC and its sensors/actuators or between a supervisory console and its PLCs;
- strict timing requirement.

These properties make industrial communications easier and more efficient to monitor than IT communications which are often more complex, evolve rapidly and have a high variability due to human activities (Cheung et al., 2006). This facilitates the creation of anomaly detection models.

2.3.3 Network-protocol-based Intrusion Detection System and Modbus

The two aforementioned points about ICS when comparing it to IT (i.e., higher cybersecurity risks and effective monitoring), are complementary and make the use of IDSs even more important. However, the heterogeneity of industrial solutions, their low hardware resources and their closed (proprietary) aspects limit the possibilities for an HIDS. That is why we focus our work on Network-based Intrusion Detection on ICS and we propose to add stateful monitoring support in industrial protocols.

Modbus TCP is a simple, open specification industrial communication protocol. It has been widely used for several years in ICS. It is supported by the majority of devices and is often the only protocol offering interoperability between devices of different technologies. Moreover, it is probably the most studied industrial protocol in the scientific literature. For these reasons, Modbus will be the use case of this article.

3 RELATED WORK

Stateful monitoring of communication protocols within NIDS is not new, the first NIDS performing this date back to before 2002 (Kruegel et al., 2002). After Behavior-statistical-based NIDS, Stateful NIDS was probably one of the first behavioral models. The number of articles on this subject being relatively high, we will focus on those dealing with ICS and, in particular, the Protocol-Specification-based NIDS and those whose work is close to our work.

Tidjon et al. (Tidjon et al., 2020) notes the current shortcoming of NIDS in not having a dynamic vision of network data exchanges. They proposed a stateful modeling method based on an algebraic language, to overcome this shortcoming. However, the method suggested by Tidjon et al. applies to the rules and signatures engine, rather than the protocols.

Carcano et al. (Carcano et al., 2010) proposes a modeling of an ICS using a virtual representation divided between coherent and incoherent states of the system. The entry in an incoherent state raises an alert. For this modeling they use standard Backus-Naur Form (BNF) notation. In a similar way, Monzer (Monzer, 2020) proposes to model an industrial system using a hybrid automaton for anomaly detection. Monzer also proposes a methodology to convert PLC programs from Grafcet language to hybrid automata. These two studies are clearly oriented on detection methods specifically related to use cases and do not offer model abstraction.

Two studies closely related to ours are Cheung et al. (Cheung et al., 2006) and Goldenberg and Wool (Goldenberg and Wool, 2013). They both propose Protocol-specification-based NIDS. The first one (Cheung et al., 2006) proposes three methods to model the behavior of Modbus TCP within NIDS, including one at the protocol level. One of the authors will propose in (Dutertre, 2008) a formal method based on Prototype Verification System (PVS) to describe Modbus and to perform conformance checking. Like us, Goldenberg and Wool (Goldenberg and Wool, 2013) proposes stateful monitoring on Modbus protocol using an FSM. However, both of these studies act as the final detection engine, while we propose a stateful engine at an intermediate level allowing to keep the original detection engine which is enriched with new data.

A search on the IEC 60870-5-104 protocol also allows to identify some interesting articles for the modeling of this protocol (Yang et al., 2014). In particular, (Yu et al., 2015) reminds us of the relevance of FSM to model communication protocols and more generally event driven systems. However, he points out that traditional FSMs have intrinsic usability problems that make it difficult for humans to use them in practice. As an alternative, he proposes the use of Harel's Statecharts or its variant standardized in the UML standard: the UML Statechart.

There is no study proposing to integrate, in a complementary way, a stateful monitoring stage to a NIDS, which is both independent of the use case and the detection engine. The latter will keep its properties and will benefit from complementary data linked to the stateful monitoring to make its predictions.

4 DESIGN AND IMPLEMENTATION

4.1 NIDS Framework

The main objective of this paper is to realize a stateful NIDS. As mentioned in Section 1.2, compared to existing NIDS, our contribution focuses on the stateful aspect. The different existing stages of NIDS, namely capture, dissection and detection being relatively complex and in order not to redevelop existing things, we will rely on free NIDS that we will modify.

We chose the Open Source NIDS Zeek as it allow to easilly extend its functionalities. Moreover, Zeek also seems to be the most used NIDS in the scientific literature, probably for these same reasons of expandability.

4.2 Modeling Stateful

Our secondary goal is to implement protocol stateful monitoring in the most generic way and to provide an abstraction level between the communication protocol and the NIDS. To meet this objective, we propose to use standard modeling methods such as FSM. However, we also propose to take into account the difficulties mentioned in Section 2.2 about FSM and will use Harel's Statecharts.

We have choosen the W3C SCXML format to represent Harel's statecharts because it is well known, it is a standard and for the existence of several tools, such as SCXMLCC and SCXMLGUI.

4.3 Implementation

Zeek, as a framework, provides an Application Program Interface (API) and documentation for plugin integration. It is accessible in the Zeek language, C++, but also in an internal language, more abstract and simpler: BINPAC.

The implementation of our solution then consists in creating a Zeek plugin which will have to fulfill the following goals:

1. Modeling/Stateful, which consists in modeling, with the help of a Statechart, one, or more, given communication protocol, described in SCXML format.
2. Interfacing. which consists of:
   - Interface with the dissector of the communication protocol and take into account its events to transmit them to the model realized in Step 1. This last one will then be able to carry out the stateful monitoring.
   - Generate events in case of anomaly in the stateful model.
   - Provide to Zeek's detection engine API with the internal state of the Statechart and its context.

According to Zeek's philosophy, Step 1 will be done in C++ and Step 2 will be done using a BINPAC *script*.

4.3.1 Modelisation/Stateful

Several development libraries in C++ implement Statecharts. The two best known libraries are Boost and Qt. The Qt library offers the advantage of directly taking into account the SCXML format, but this support seems incomplete. It is the Boost library which was selected because it seems more tested, has a better documentation, a community and updates more important. The lack of SCXML format support by Boost is compensated by the existence of the SCXMLCC tool which allows to generate C++ code implementing Boost Statecharts from a model in SCXML format.

This Stateful modeling is then partly generated automatically from a Statechart model described in SCXML.

The remaining implementation consists in exporting functions to manipulate this model. These functions will be called by the interfacing script. Here are the three main functions:

- init: initialize an instance of a statechart for each new session of the protocol.
- dispatch: sends to the model an event received from the dissector. This event will fire a statechart transition or raise an anomaly.
- get_states: returns the current states and their contexts.

4.3.2 Interface

The implementation of the interfacing part of the Zeek plugin is relatively classic and resembles what is traditionally found in existing plugins. However, it should be noted that the taking into account of a protocol by our plugin requires the existence of a Zeek dissector for this protocol.

Figure 2 represents the global architecture of the Zeek plugin. The left-hand (light blue) rectangle represents the action that generates the plugin code associated with each protocol. The right-hand (light green) rectangle represents the Zeek plugin, composed of the Stateful stage (previously generated part and exported functions) and the BINPAC interface that interacts with Zeek.

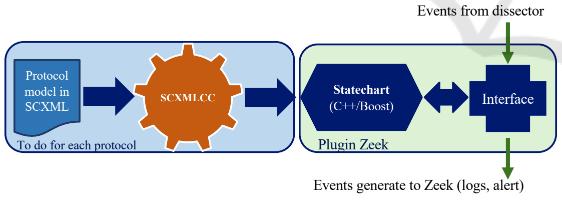

Figure 2: Zeek stateful-plugin architecture.

4.4 Modeling Modbus TCP

The use case chosen for this experimentation is the industrial communication protocol Modbus TCP. It is a very simple communication protocol, working on the master/slave principle. Here is a simplified description of the functioning of Modbus: A Modbus master (TCP client), sends a request to a Modbus slave (TCP server) by sending a message, this last one answers to the master by returning a message. Each request from the master corresponds to a single network packet and have a single network packet in response from the slave. The slave does not have the capacity to initiate an exchange with the master, it can only answer its requests. The absence of response from a slave to a master request within a defined time (timeout) is foreseen by the protocol. In the same way, a slave which would not understand a request or would be unable to answer it will return an error message. Each Modbus message have a fully defined format in the standard according to its type.

Figure 3 represents a possible very simplified statechart model of the Modbus TCP protocol. Figure 3 is an extract of *scxmlgui*.

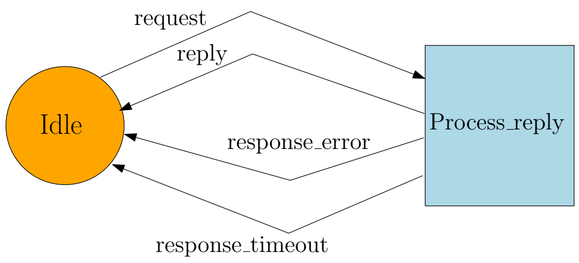

Figure 3: Statechart simplified modeling of Modbus TCP.

4.5 Modbus Dataset

Finding a dataset to test our prototype was not easy. We initially wanted to use the network exchanges of a real industrial system including a supervisory console Schneider Magelis GTU and a Schneider M340 PLC. However, we realized that Schneider does not use the standard Modbus TCP functions. Instead, Schneider reimplements its own protocol on top of the 0x90 function of Modbus TCP which is not documented.

We then thought of using Modbus TCP datasets listed by (Choi et al., 2019). Unfortunately, the content and the operating mode of these datasets are not fully described and require a consequent cleaning to extract the useful data. Finally, for our experimentation, we simulated a system composed of two Modbus TCP simulators.

The first simulator uses software *Modbus Poll*[2] to simulate a Modbus master and the second simulator uses software *Diagslave Modbus Slave Simulator*[3] to simulate a slave. Our platform is composed of two simulated devices that communicate directly through a network interface.

The dataset thus generated represents 124 Modbus TCP packets corresponding to 62 requests and 62 associated responses over a time span of 62 seconds. The *tcpreplay*[4] tool is used to replay the dataset according to the same time periodicity as the simulated platform.

[2]https://www.modbustools.com
[3]https://www.modbusdriver.com/diagslave.html
[4]https://tcpreplay.appneta.com/

5 RESULTS

The dataset generated in Section 4.5 does not contain any anomaly of use of the Modbus TCP protocol. Figure 4 is an extract from the event log of the Zeek plugin which illustrates after an initialization phase the succession of events:

- [*OnEventRequest*] '*Request*': corresponding to the arrival of a Modbus TCP request, followed by the change of state *Idle* $->$ *Process_reply* where the automaton goes from state *Idle* to *Process_reply*
- [*OnEventReply*] '*Reply*': corresponding to the arrival of a Modbus TCP reply, followed by the change of state *Process_reply* $->$ *Idle* where the automaton goes from state *Process_reply* to *Idle*

```
scxml -> global
global -> Idle
[OnEventRequest] 'Request'
Idle -> Process_reply
[OnEventReply] 'Reply'
Process reply -> Idle
[OnEventRequest] 'Request'
```

Figure 4: Modbus TCP Plugin logs.

5.1 Missing Reply

In order to highlight the anomaly detection, we modified the dataset by removing a Modbus TCP response. The deletion of the network packet causes a desynchronism in the TCP exchanges. This has no impact on Zeek and our plugin. Figure 5 corresponds to the event log of our plugin and shows on line 3 that an anomaly is reported following the reception of a *Request* event when we are in the state *Process_reply*. This event does not correspond to a valid transition according to our Modbus TCP model represented in Figure 3.

```
[OnEventRequest] 'Request'
Idle -> Process_reply
[Anomaly] On event: request
Process_reply -> Process_reply
[OnEventReply] 'Reply'
Process_reply -> Idle
```

Figure 5: Modbus TCP Plugin logs with missing Reply.

5.2 Reply Timeout

In a similar way to the deletion of a packet, we wanted to check that the timeout management is well taken into account. To do this, we have voluntarily delayed the arrival of a response so that it causes a timeout. Figure 6 corresponds to the event log of our plugin and shows on line 3 that a timeout has been applied. The management of timeouts has been taken into account in our Modbus TCP model represented in Figure 3, i.e., it is a valid transition which has not raised any anomaly but which has caused a change of state from *Process_reply* to *Idle*. In line 5, the late arrival of the response is no longer expected and causes an anomaly.

```
[OnEventRequest] 'Request'
Idle -> Process_reply
[OnEventResponseTimeout] 'Response_timeout'
Process_reply -> Idle
[Anomaly] on event Reply
Idle -> Idle
```

Figure 6: Modbus TCP Plugin logs with Reply timeout.

5.3 Stateful NIDS

The implementation of the Zeek plugin and the experimentation with Modbus TCP show that our prototype works. A stateful monitoring is correctly done. The live evolution of the statechart can be visually consulted using the *scxmlgui* tool which allows to see the states being activated and deactivated according to the events received from the dissector. The network packets corresponding to the Modbus TCP protocol that do not respect our simplified model generate anomalies which are logged.

5.4 Abstraction

Our secondary goal of designing an abstract implementation that can take into account several protocols without having to modify the plugin is theoretically achieved because it consists to add the statechart modeling description of protocols in SCXML format. In practice, the description of a protocol model in SCXML format must use the same naming scheme for the transitions as the dissector events for this protocol.

6 DISCUSSION

Our solution is positioned as a new protocol stateful monitoring stage within a NIDS. It is a complementary stage that does not aim to replace existing ones. In particular, it is not intended to replace the final detection engine. In its current state, it only allows to generate new events in Zeek (anomalies) when a protocol model is not respected. The current treatment of these events is a logging. Our solution does not explicitly generate alerts because this is the responsibility of the detection engine. In any case, it would be unable to detect attacks that would respect our modeling of this protocol.

To improve the detection performance of the NIDS, the detection engine would have to take into account the new events generated by our plugin or be able to query it. For example, for a signature-based detection engine, it is possible to write new rules that directly take into consideration events generated by the plugin and corresponding to anomalies. In addition, rules of a signature-based detection engine can now check the state of a protocol session as an additional criterion, thus reducing the risk of false positives. However, this requires rewriting the signatures to integrate these new additional capacities. In addition, this also requires to build a comparative benchmark with quality datasets. These conditions are generally quite difficult to meet.

This work was initially done with an industrial use case concerning a remote control system for electrical networks with the IEC 61870-5-104 communication protocol. This protocol is more complex, less known, but especially the dissector and the dataset that we used are not public. This is why, for this article, we have redefined the use case using Modbus TCP. The stateful monitoring of Modbus TCP has been done using a very simplified model that does not consider *broadcast* messages and has only two states. It would of course be possible to make a more complete model, which would allow to identify more anomalies. For example, we could consider checking for each different type of request that the associated answer corresponds to this request.

Our prototype on the IEC 61870-5-104 protocol has been tested on several real systems. It has allowed to identify several anomalies in equipment implementations. However, since no attack took place on these platforms, the prototype can only detects errors in the implementation of the protocols. Following these experiments, our prototype should be deployed as a conformity checker tool for the IEC 61870-5-104 protocol.

We have not measured the cost in hardware resources of our plugin, but no doubt that this cost is significant. Indeed, our solution requires to keep in memory all active network sessions. The overall performance of the NIDS is reduced. However, our solution is aimed at industrial systems where the number of sessions is reduced and in this context, the additional cost is probably negligible.

7 CONCLUSION

We have proposed a new method based on communication protocol stateful monitoring to improve the performance of existing NIDS. Stateful monitoring, although existing for several years, is proposed here with two novelties:

- It is positioned as an intermediate and complementary stage that remains agnostic of the use case. The NIDS retains all its original capabilities and is provided with new information on the history of past exchanges and protocol compliance.
- It uses a level of abstraction with the NIDS allowing the easy addition of new protocols and possible sharing of protocol models using the open format SCXML.

The results obtained are encouraging and show that this solution works: anomalies are well detected and the detection engine has access to the current state of the state graph.

Future work will consist in experimentally verifying the effective improvement of the performance of a NIDS. This would require rewriting the signatures of the signature detection engines or adapting the models of the anomaly detection engines to take into account the new information available. This would then require comparing solutions with and without stateful monitoring using a quality dataset.

Other future work consists in improving the stateful monitoring engine by correcting a specific problem that can occur in NIDS: packet loss. Indeed, although it did not appear in our experimentation, the loss of one or more packets is a very probable situation in NIDS. It can lead to a desynchronism between the real state of a network exchange and the stateful monitoring. This would require resynchronization methods such as automaton branch prediction.

REFERENCES

Carcano, A., Fovino, I., Masera, M., and Trombetta, A. (2010). State-Based Network Intrusion Detection Systems for SCADA Protocols: A Proof of Concept. In Rome, E. and Bloomfield, R., editors, *Critical Information Infrastructures Security*, Lecture Notes in Computer Science, pages 138–150, Berlin, Heidelberg. Springer.

Cheung, S., Dutertre, B., Fong, M., Lindqvist, U., Skinner, K., and Valdes, A. (2006). Using Model-based Intrusion Detection for SCADA Networks. *Proceedings of the SCADA security scientific symposium. Vol. 46. 2007.*

Choi, S., Yun, J.-H., and Kim, S.-K. (2019). A Comparison of ICS Datasets for Security Research Based on

Attack Paths. In *Critical Information Infrastructures Security*, volume 11260, pages 154–166. Springer International Publishing, Cham. Lecture Notes in Computer Science.

Dutertre, B. (2008). Formal Modeling and Analysis of the Modbus Protocol. In *Critical Infrastructure Protection*, IFIP International Federation for Information Processing, pages 189–204, Boston, MA. Springer US.

Goldenberg, N. and Wool, A. (2013). Accurate modeling of Modbus/TCP for intrusion detection in SCADA systems. *International Journal of Critical Infrastructure Protection*, 6(2):63–75.

Gouda, M. and Liu, A. (2005). A model of stateful firewalls and its properties. In *2005 International Conference on Dependable Systems and Networks (DSN'05)*, pages 128–137. ISSN: 2158-3927.

Harel, D. (1987). Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274.

Kruegel, C., Valeur, F., Vigna, G., and Kemmerer, R. (2002). Stateful intrusion detection for high-speed network's. In *Proceedings 2002 IEEE Symposium on Security and Privacy*, pages 285–293. ISSN: 1081-6011.

Mitchell, R. and Chen, I. (2014). A survey of intrusion detection techniques for Cyber-Physical Systems. *ACM Computing Surveys*, 46(4):55:1–55:29.

Monzer, M.-H. (2020). *Model-based IDS design pour ICS*. PhD Thesis, Université Grenoble Alpes.

Qin-Cui, F., Zi-ying, L., and Ke-jia, F. (2009). Implementation of IEC60870-5-104 protocol based on finite state machines. In *2009 International Conference on Sustainable Power Generation and Supply*, pages 1–5. ISSN: 2156-969X.

Tidjon, L., Frappier, M., and Mammar, A. (2020). Intrusion Detection Using ASTDs. In Barolli, L., Amato, F., Moscato, F., Enokido, T., and Takizawa, M., editors, *Advanced Information Networking and Applications*, Advances in Intelligent Systems and Computing, pages 1397–1411, Cham. Springer International Publishing.

Uppuluri, P. and Sekar, R. (2001). Experiences with specification-based intrusion detection. In *International Workshop on Recent Advances in Intrusion Detection*, pages 172–189. Springer.

Yang, Y., McLaughlin, K., Sezer, S., Yuan, Y., and Huang, W. (2014). Stateful intrusion detection for IEC 60870-5-104 SCADA security. In *2014 IEEE PES General Meeting | Conference Exposition*, pages 1–5. ISSN: 1932-5517.

Yu, C., Shen, Y., Huang, L., Huang, H., Zhang, X., Jia, S., and Liu, J. (2015). The implementation of IEC60870-5-104 based on UML statechart and Qt state machine framework. In *2015 IEEE 5th International Conference on Electronics Information and Emergency Communication*, pages 392–397.