Using Model-based Intrusion Detection for SCADA Networks[†]

Steven Cheung, Bruno Dutertre, Martin Fong,
Ulf Lindqvist, Keith Skinner, Alfonso Valdes

Computer Science Laboratory
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025
Email: *firstname.lastnam*e @sri.com

Abstract: In a model-based intrusion detection approach for protecting SCADA networks, we construct models that characterize the expected/acceptable behavior of the system, and detect attacks that cause violations of these models. Process control networks tend to have static topologies, regular traffic patterns, and a limited number of applications and protocols running on them. Thus, we believe that model-based monitoring, which has the potential for detecting unknown attacks, is more feasible for control networks than for general enterprise networks. To this end, we describe three model-based techniques that we have developed and a prototype implementation of them for monitoring Modbus TCP networks.

Keywords: SCADA security, intrusion detection, model-based detection, Modbus TCP

1 Introduction

Digital controls have been deployed in automation systems for many decades. A notable example of process control systems (PCS[1]) are SCADA systems in electric power distribution and distributed controls in refining and manufacturing. Historically, such systems have not been connected to wider networks, used unfamiliar protocols,

[†] This work was produced in part with support from the Institute for Information Infrastructure Protection (I3P) research program. The I3P is managed by Dartmouth College, and supported under Award number 2003-TK-TX0003 from the U.S. Department of Homeland Security, Science and Technology Directorate. Points of view in this document are those of the author(s) and do not necessarily represent the official position of the U.S. Department of Homeland Security, the Science and Technology Directorate, the I3P, or Dartmouth College.

[1] In this paper, we use the terminology "Process Control Systems" or simply "Control Systems" to refer to the entire class of digital control systems in automation, and "Process Control Networks" to refer to networks on which such systems are implemented, which include network infrastructure such as switches and routers.

and otherwise achieved some measure of security through obscurity and isolation. For reasons of increased efficiency, increasing sophistication of end devices (which now frequently feature embedded OS), and migration to open standards, such systems are now extensively interconnected to conventional enterprise systems and encapsulate legacy control protocols in conventional networking protocols such as TCP/IP. This has led to improvements in functionality and productivity, but has raised concern that the control systems are now vulnerable to remote attacks with potential catastrophic consequences.

For a variety of reasons, security practices from enterprise systems are not always easy to adapt to control systems. On the other hand, control systems tend to have static topology, regular traffic, and simple protocols. Monitoring of control systems for security is therefore potentially easier than in enterprise systems. In our approach, we construct models that characterize the expected behavior of the system, and detect attacks that cause the system to behave outside of the models. This approach has not gained traction in enterprise systems because of the difficulty of specifying correct function abstractly enough but at sufficient fidelity for adequate detection performance. We hypothesize that model-based monitoring, which offers the promise of detecting zero day attacks, may be more feasible in control systems. In this paper, we explore this hypothesis, and describe the implementation of a lightweight model-based monitor in a control system testbed.

The rest of the paper is structured as follows. Section 2 briefly describes the Modbus TCP protocol and attacks against Modbus TCP networks. Section 3 presents the model-based intrusion detection approach, and gives an overview of the techniques we have developed for monitoring Modbus TCP networks. Sections 4-7 detail the detection techniques. Section 8 describes experimental validation results. Section 9 concludes this paper.

2 Modbus TCP

Modbus [5] is a simple request-response protocol widely used in SCADA networks. A Modbus client (or master) may send a request to a Modbus server (or slave)[2]. The request includes a function code specifying the service it wants, and a list of parameters (such as the addresses of the data items). Modbus functions include reading values from discrete inputs/coils (bit-size entities) or registers (16-bit-word entities), writing values to coils and registers, and performing diagnostics for the servers. After the server receives and processes the request, it sends a response back to the client. The response can be either a normal response or an error (or exception) response.

Modbus requests and responses are encoded as Modbus Protocol Data Units (PDU), which may be encapsulated in a serial line communication protocol, or in TCP/IP (which is commonly referred to as Modbus TCP [6]).

---

[2] Originally, Modbus was a master-slave protocol for serial buses. Implementing Modbus on TCP has become increasingly common. In this context, a Modbus master is a TCP client, and a Modbus slave is a TCP server.

The encapsulation of a Modbus PDU over TCP/IP is called a Modbus TCP/IP Application Data Unit (abbreviated as ADU in this paper). An ADU consists of a header (called Modbus Application Protocol header or MBAP header) and a Modbus PDU. The header consists of the following fields:

- Transaction identifier: Used for pairing the request and the response corresponding to a transaction
- Protocol identifier: This field should be 0 for the Modbus protocol
- Length: Used for specifying the length of the following fields
- Unit identifier: Used for identifying a Modbus server connected to a Modbus gateway[3], say over a serial line

2.1 Attacks against Modbus TCP

There is little published information about Modbus TCP attacks. An exception is the intrusion detection work performed by Digital Bond, Inc., http://www.digitalbond.com. Attacks addressed by the Digital Bond intrusion detection rules include denial of service (e.g., rebooting Modbus servers, configuring them to provide no service—called listen-only mode, and crashing servers with a large-size request), reconnaissance (e.g., unauthorized reading of data, and gathering device information), and unauthorized write requests.

The migration to TCP/IP networks and using commodity platforms for Modbus clients make these systems potentially vulnerable to attacks against the operating system and the network, even if such attacks are not against the Modbus protocol per se. Modbus clients today typically run a version of the Windows operating system. For some application environments in which it is expensive or difficult to shut down the control system, operating system patches may not be up to date. So these systems may run with known vulnerabilities. There have been instances of worm attacks infecting control networks (for example, http://www.securityfocus.com/news/6767), even if they did not specifically target such networks.

3 Model-based Detection

We use a model-based approach for detecting intrusions. The basic idea is to construct models that characterize the expected/acceptable behavior of the entities. This approach detects attacks that cause the system to behave outside of the models, and is applicable for attacks on the Modbus protocol, on the OS platforms, and on the networking infrastructure.

The model-based approach is not new in the area of intrusion detection. For example, specification-based intrusion detection [3], and using system call sequences to

[3] Using a Modbus gateway, a Modbus TCP client can communicate with Modbus servers that run on serial lines via the gateway. The unit identifier is used to specify the server connected to the gateway.

characterize the normal behavior of Unix programs [2] can be viewed as model based. A main contribution of this paper is on developing model-based techniques for monitoring Modbus TCP networks.

An advantage of the model-based approach is its potential for detecting unknown attacks. The signature-based approach is quite effective for detecting known attacks, as it usually captures the distinguished properties of the attacks it is designed to detect. In some cases, one can develop generalized signatures to detect variations of an attack or even a class of attacks. An example is finding binary opcodes in the arguments of some Unix system calls for detecting buffer overflow attacks [4]. However, if an unknown attack does not share similar attributes with the known attacks, using generalized attack signatures may not help in catching it. Because little is publicly known about attacks against Modbus TCP networks, the model-based approach is particularly attractive, as knowledge about attacks is less crucial in this approach.

Two main challenges prevent the model-based approach from widespread use. First, it may be difficult/expensive to construct the models. Second, this approach could cause many false alarms, which occur if the models are not accurate enough (i.e., they do not capture all the normal behavior). Ongoing research is being conducted to address these concerns and some progress has been made. For example, Sekar et al. [9] combine specification-based with statistical anomaly detection techniques to ease the task of model construction and to reduce false alarm rate. We argue that one can exploit the special-purpose nature of Modbus TCP networks for constructing accurate models. This is based on our observations that these networks tend to be more restrictive and predictable than general TCP/IP networks, because network changes tend to be infrequent, the set of applications running on the network is usually small, and Modbus TCP is a relatively simple protocol.

The following sections describe the three model-based detection techniques we have developed for monitoring Modbus TCP. We employ protocol-level models for characterizing Modbus TCP requests and responses, based on the Modbus application protocol specification document and the Modbus TCP implementation guide [5, 6]. For instance, Modbus requests that contain an "unexpected" function code can be detected using these models. This may indicate a reconnaissance event, exploiting an unknown backdoor, or a denial-of-service attempt.

We model the expected communication patterns among different types of components in a Modbus network. For instance, consider a model for Modbus servers, which only respond to Modbus TCP requests, and do not initiate TCP connections themselves. If a Modbus server has been compromised, and it attempts to connect to other components to attack them, its behavior will violate the model, and thus it can be detected.

We use a learning-based approach for detecting changes in server or service availability. Using this approach, we can detect, for example, Modbus servers that used to provide a specified function, but have failed to successfully serve a recent invocation for that function. This may indicate a failure or a successful denial-of-service attack.

4 Protocol-level Models

The Modbus standard and reference documents [5, 6], specify the expected characteristics of Modbus requests and responses. In this approach, Modbus traffic that violates these specifications will trigger an IDS to generate alerts.

4.1 Specifications for Individual Fields

The most basic type of specification concerns a single independent field in Modbus TCP Application Data Units. In particular, we have developed specifications for function codes, exception codes, and protocol identifiers. We use our specification for function codes as an example to illustrate this type of specification.

In Modbus, function codes are represented as a byte. In other words, there are 256 possible values for this field. The most-significant bit of the function code is used by servers to indicate an exception response. Thus the legitimate function codes of Modbus requests are all less than 128. Moreover, function code 0 is invalid. The function codes in the range 1-127 are partitioned into three categories: public (corresponding to well-defined functions described in the standard document), user-defined (available for vendor-specific functions), and reserved (used by legacy products and not available for public use).

Without knowing information about the user-defined functions and the reserved functions supported by the monitored Modbus servers, one could develop the weakest specification for the function codes by including the set of public function codes that are documented in [5] (i.e., {1-8, 11-12, 15-17, 20-24, 43}), the set of possible user-defined function codes, and the set of reserved function codes.

We note that some Modbus devices may support only a subset of the public function codes, and may provide user-defined functions. For example, a Modbus device that we have studied does not support the following public function codes {11-12, 17, 20-24, 43}, and it uses two user-defined function codes, namely, 65 and 66. By incorporating information about the monitored devices, we may develop a more accurate function code specification:

$$\forall R \in \text{ModbusTCPRequest} \bullet \mathit{funCodeField}(R) \in \{1-8, 15-16, 65-66\}$$

4.2 Specifications for Dependent Fields

Another type of specification involves cross-field relationships. The acceptable values of a field may depend on the value of another field. The specifications for length, subfunction codes, and arguments in the payload of Modbus PDU (such as data addresses and the number of data items) fall into this category. We use the length field of ADU as an example.

The length field specifies the number of bytes of the following fields, including the unit identifier and the Modbus PDU fields. Because the unit identifier is always 1 byte long, and the maximum size of a Modbus PDU is 253 bytes, the value of the length field should be in the range [1, 254].

The value of the length field depends on the function code of the Modbus PDU. For some function codes, such as the read coils function (whose function code is 1), the length of the PDU is fixed. Thus the value of the length field is also fixed. For some other function codes, such as the write multiple coils function, the length varies, but there is a lower bound and an upper bound for it. A more accurate specification for the length field can be derived by taking into account the function codes. For example, our specification for the length field with respect to the read coils function is as follows:

$$\forall R \in \text{ModbusTCPRequest} \bullet \mathit{funCodeField}(R) = 1 \Rightarrow \mathit{lenField}(R) = 6$$

4.3 Multi-ADU Specifications

More complex specifications involve multiple Modbus requests or responses. For example, in a correct Modbus transaction, several fields in the request must match the corresponding fields in the response.

Depending on the function codes, a more detailed specification may cover additional relationships between the fields of a Modbus request and those of the Modbus response. For example, in a read input registers transaction, the "quantity of input registers" field of the request should be equal to half of the "byte count" field in the corresponding response, because registers are 16-bit words. This can be useful, for example, to detect that the number of bytes in the response exceeds what was asked for in the request, possibly indicating a buffer-overflow attack.

4.4 Snort-based Implementation

We have developed Snort rules for detecting violations of some of the above specifications. Snort is an open-source network intrusion detection system (see http://snort.org), and has an easy-to-use rule language for specifying misuse and attack signatures. In many cases, these signatures can be encoded using a single Snort rule, which specifies the endpoints (e.g., IP addresses and port numbers) and other attributes of the packets (e.g., keyword pattern(s) in the packet payload). In particular, Digital Bond has developed a Snort ruleset for detecting known Modbus TCP attacks.

Typically, Snort is used as a signature-based IDS. In the signature-based approach, one analyzes an attack, identifies signatures that capture the unique, key aspects of the attack, and develops intrusion detection rules/modules for the signatures. In this work, we use Snort in a different way. We start with developing models to characterize the expected behavior based on the Modbus TCP specification. Then we develop Snort rules to detect the "complement" of the models. For example, we specify the valid Modbus function codes that are applicable for the deployed Modbus devices. From that, we can derive the set of function codes that are invalid. Then we develop Snort rules for detecting Modbus requests that contain a function code belonging to that set. In addition to developing rules for function codes, we have developed Snort rules for other attributes, including protocol identifiers, length, exception codes, and data addresses.

We have not developed Snort rules for some of the complex specifications, such as those involving multiple ADUs. Although the rule language of Snort is usually

expressive enough for single-packet attacks, using Snort to detect complex, multi-packet attacks can be complicated. To handle complex attacks that require cross-event analyses in Snort, one may need to develop Snort preprocessors, which offer modular, "plug-in" style extensions for Snort. Developing Snort preprocessors is significantly more resource intensive than writing rules. To handle the more complex specifications (and for other reasons), we are developing a custom-built IDS based on formal models of Modbus TCP, which is described in the next section.

5 Making the Model the Detector

Snort can be used as summarized previously for detecting when the observed traffic violates constraints derived from Modbus specifications. However, this approach has some limitations. As already noted, certain traffic properties that require analysis across several packets are difficult to capture as Snort rules. In addition, the Snort rules are coded by hand by a specialist with significant expertise in writing Snort rules. This is not so much an issue for rules that are generic and can be reused across many Modbus-based control systems. For example, use of the function code '0' is forbidden by the specifications. Seeing a request with this function code should raise an alarm on any Modbus system. However, generic rules can detect only gross violations of the Modbus standard. On most systems, the expected traffic is much more restricted than what the Modbus specifications allow. For example, only a subset of the function codes is typically used, and restrictions on the allowed parameters are common. For improved accuracy, it is useful to detect violations of these system-specific constraints. Developing or customizing Snort rules for each system by hand is difficult and error-prone. Since our approach starts with a formal model of the system, we are currently investigating using such a model directly as a detector.

Our approach relies on formally specifying the characteristics of a specific Modbus device. We are currently using the PVS specification language for this purpose. PVS [7] is typically used as a theorem prover for verification of complex systems or algorithms. We are not relying extensively on theorem proving, but we exploit another important feature of PVS, namely, the possibility of writing (and compiling) executable specifications. We can develop a formal model of a Modbus system that specifies the allowed set of requests to a device and constraints on the expected responses. An example showing how this can be done is in [1]. The result is an executable PVS model of the Modbus traffic. The model defines predicates such as *valid_request(r)* that specifies whether a packet *r* is an acceptable request for a device *D*, and *valid_response(r,s)* that says that *s* is an acceptable response to *r*. Since these predicates are executable, it is straightforward to use them for intrusion detection: whenever a Modbus packet r is sent to *D* , we compute *valid_request(r)*. If the result is false, we raise an alarm. Similarly, evaluating *valid_response* on a pair request/response detects unexpected Modbus transactions. We are currently implementing this approach by integrating PVS executable code within the EMERALD infrastructure.

By directly using PVS models as monitors, this method is a systematic method of deriving an intrusion-detection sensor from a formal model, without a possibly difficult and error-prone hand translation to an intermediate system such as Snort. The PVS

language is much more expressive than Snort rules, so we expect increased detection accuracy. In addition, the PVS model can be easily customized or modified to fit a specific system or device. For example, modifying a few of the model parameters is sufficient to specify the function codes and address ranges supported by a device.

6 Communication-pattern-based Detection

By specifying the expected communication patterns among network components, one can detect attacks that violate these patterns. These attacks may include compromised hosts attempting to attack other hosts, violations of network access policies, and exploiting misconfigured hosts or network services.

In SCADA networks, unlike general-purpose TCP/IP networks, the communication patterns tend to be more well-behaved (e.g., because of a small, restricted set of applications running on them) and have less frequent network changes. This characteristic makes it feasible to specify the expected communication patterns.

Let us look at an exemplary SCADA network architecture. Sandia National Laboratories (SNL), as part of the PCS security project of the Institute for Information Infrastructure Protection (http://thei3p.org), has developed a SCADA testbed. The architecture of the testbed is depicted in Figure 1. The testbed architecture consists of three zones, namely, Corporate Zone (CZ), Demilitarized Zone (DMZ), and Process Control Zone (PCZ). The corporate network, which has a historian client, is connected to the DMZ, which has a DMZ historian server. The DMZ is in turn connected to a process control network, which includes a human machine interface (HMI), SCADA historian (which populates the DMZ historian), SCADA server, OPC server, engineering workstation, domain controller, and devices in the form of a programmable logic controller (PLC) as well as a flow computer.

We then specify the network access policies, which characterize the acceptable communication patterns.

- Communication between CZ and DMZ is restricted to that between the corporate historian client and the DMZ historian server.
- Communication between DMZ and PCZ is restricted to that between the PCZ SCADA historian and the DMZ historian server.
- Within PCZ, the SCADA server may communicate with the flow computer and the PLC using the Modbus TCP protocol. The SCADA HMI may interact with the SCADA server and the engineering station. The SCADA server may communicate with the SCADA historian to report the status information of the process control network. Moreover, various hosts in the process control network may communicate with the domain controller for directory service, authentication, and other network services.

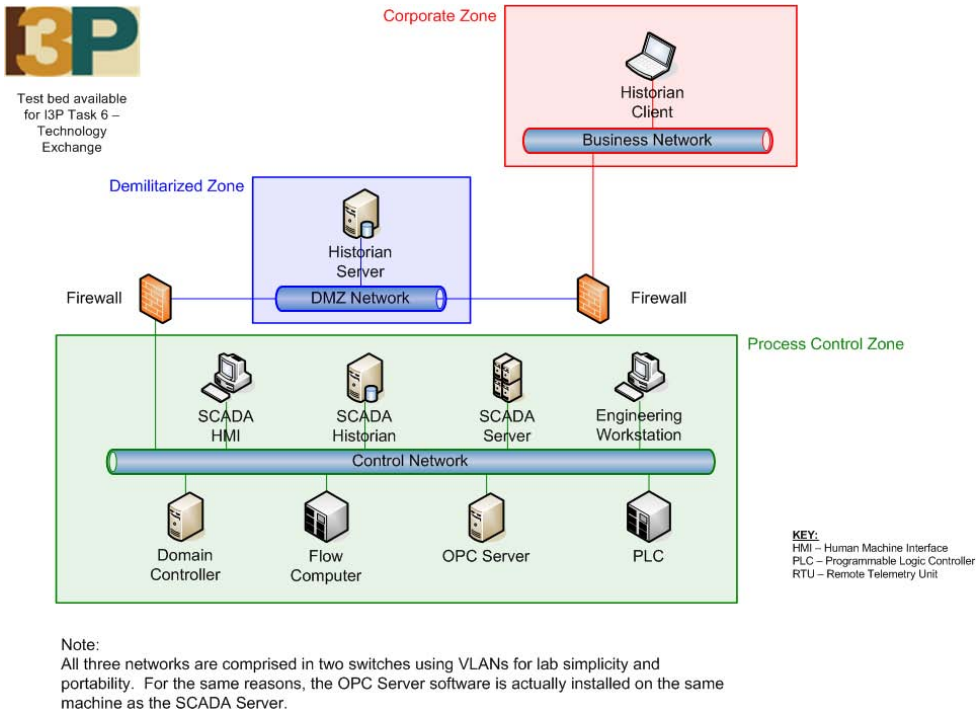

Figure 1 – SCADA Testbed Architecture (Courtesy Sandia National Laboratories)

In our implementation, we developed Snort rules to detect deviations from the communication patterns. For example, violations of the communication pattern involving the flow computer (which is a Modbus TCP server) can be detected using the following Snort rules. (Readers are referred to the Snort manual (see http://snort.org) for detailed information about the rule language.)

```
########################################
# Ruleset for MODBUS_SERVER_ONLY
########################################
# TCP connection initiation attempts to a non-Modbus port of
# MODBUS_SERVER_ONLY hosts
alert tcp any any -> $MODBUS_SERVER_ONLY !$MODBUS_PORT
(msg:"Unauthorized connection attempt to a non-Modbus port
of a MODBUS_SERVER_ONLY host"; flags: S;
classtype:bad-unknown; sid: 3006001; rev:1;)
# TCP connection initiation attempts by a MODBUS_SERVER_ONLY host
alert tcp $MODBUS_SERVER_ONLY any -> any any
(msg:"Unauthorized connection attempt from a MODBUS_SERVER_ONLY
host"; flags: S; classtype:bad-unknown; sid: 3006002; rev:1;)
# TCP connection involving (MODBUS_SERVER_ONLY, MODBUS_PORT)
# and non-Modbus client
alert tcp $MODBUS_SERVER_ONLY $MODBUS_PORT <> !$MODBUS_CLIENT any
(msg:"Unauthorized traffic between (MODBUS_SERVER_ONLY,MODBUS_PORT)
and non-Modbus client"; flags: A+; classtype:bad-unknown;
sid: 3006003; rev:1;)
```

For a packet to trigger a Snort rule, it needs to match the rule header and the rule options. The rule header describes the protocol (e.g., tcp), source IP address, source port number, destination IP address, and destination port number. In these rules, MODBUS_SERVER_ONLY is a variable that corresponds to the set of the hosts that only act as Modbus TCP servers, and play no other network roles. In the testbed, the flow computer belongs to that set. Moreover, MODBUS_CLIENT corresponds to the set of hosts that are Modbus TCP clients, for example, SCADA_SERVER in the testbed. MODBUS_PORT denotes the port number used by the Modbus TCP servers. The flags:S rule option pertains to the TCP SYN flag, which corresponds to the connection initiation stage of the TCP 3-way handshake. Also, the flags:A+ rule option pertains to the TCP ACK flag.

The above three rules pertain to different ways to violate the communication pattern corresponding to Modbus TCP servers. In other words, they collectively enforce the communication pattern for the Modbus servers.

7 Detecting Changes in Server/Service Availability

We use a heuristic approach for learning the models pertaining to the availability of servers and services. The basic idea is that when an "interesting" event (e.g., new Modbus unit ID detected) is observed for the first time, an informative message is generated. The detectors keep track of the states of the monitored components, and do not generate multiple alerts for the same events. We have developed two detectors, namely, EMERALD Bayes sensor and EModbus.

The EMERALD Bayes sensor is actually a coupled sensor, with a detector that represents models of misuse through conditional probability relations to observable evidence, and a service discovery component [10]. These sensors reinforce each other so as to improve the quality of their results. The service discovery component learns active services on a monitored network, and as these are discovered it maintains a Bayes instance that rapidly detects when the service is down. The set of valid services is made known to the misuse detector, which is probabilistically more suspicious of sessions accessing services not known to be valid. Conversely, the system is less suspicious of sessions to services that appear to be down for reasons not directly attributable to the particular session—for example, a legitimate client failing to access a service that is apparently the victim of a denial of service. In practice, this has made the overall sensor accurate in the detection of slow probes, timely in the identification of service failures, and appropriately informative as to the state of victim sessions.

In addition, the appearance of a new service after some interval of system operation may be inherently suspicious, because the system has learned all the valid services supported. In experimental deployments, such detections have been linked to the installation of rogue services or the exploitation of services that should have been disabled (such as a poorly configured FTP server on a desktop).

Service discovery is thus essential to the accuracy and sensitivity of the Bayes sensor. Discovery of TCP services is part of the EMERALD implementation for Process Control Systems (PCS), but it may have less utility because of the smaller number of

TCP services on such systems. The Modbus function code, which determines the functionality supported by a Modbus device, is analogous to a service in the sense that we use for Bayes inference. As such, we have implemented EModbus, a component that discovers supported function codes on the Modbus devices deployed on a PCS. By analogy to its TCP service discovery counterpart, EModbus increases sensitivity to stealthy misuse in the PCS, and identifies changes in service behavior. Finally, discovery of new function codes after an appropriate learning interval is suspicious and may indicate a malicious reconfiguration of a Modbus device.

8 Experimental Validation

We have developed a multi-algorithm intrusion detection network appliance for Modbus TCP networks, based on the EMERALD intrusion detection and correlation framework [8, 4, 10]. The network appliance includes intrusion detection sensors developed for monitoring enterprise TCP/IP networks—EMERALD Bayes sensor (Bayesian protocol anomaly detector), eXpert-Net (stateful protocol analyzer), and Snort (widely used signature-based IDS)—and sensors that are specific to Modbus TCP or to the monitoring environment. The sensors in the latter category are based on the signature-based rules developed for Modbus TCP by Digital Bond, protocol-level rules described in Section 4, communication pattern rules in Section 6, and EModbus in Section 7. All these sensors report to a probabilistic alert aggregation engine, called eAggregate, which resides on the appliance.

The experiment was conducted on a SCADA testbed at SNL, whose architecture is essentially the same as that depicted in Figure 1. Also, SNL developed a multi-step attack scenario for the testbed. In the attack scenario, an adversary first compromises a system on the corporate network, which is typically connected to the Internet. From there, the attacker gains access to the DMZ historian server, then a historian machine in the process control network zone, and finally performs reconnaissance and attacks the Modbus servers and other hosts in the process control network. During an attack run, tcpdump traces were collected, which were later used to validate the sensors.

Results from this experiment provide evidence that the model-based intrusion detection approach is effective for monitoring SCADA networks, and that it is complementary to the signature-based approach. Different sensors detected different aspects of the multi-step attack scenario. Specifically, Snort and the EMERALD Bayes sensor detected network scans. The signature-based rules developed by Digital Bond detected events involving an unauthorized host sending read and write attempts to an Modbus server. EModbus generated Modbus server/service discovery messages during a Modbus attack. The protocol-level rules detected invalid Modbus requests (e.g., Modbus requests that contained unsupported function codes). And the communication pattern rules generated alerts for attack steps that violated the expected communication patterns.

9 Summary and Conclusion

We have taken advantage of the greater regularity and stability (by comparison to enterprise systems) of process control systems in regard to topology, communication,

and configuration to implement lightweight model-based intrusion detection components.

Our work concerns the Modbus protocol encapsulated within TCP/IP, but we believe it is applicable to other control system protocols as well. Model-based detection is an important complement to signature-based approaches in that the former provides at least some potential for detection of zero-day attacks, while the latter effectively detect specific known attacks.

We explored three techniques for model-based detection. First, we developed protocol-level models for characterizing Modbus TCP, and developed Snort rules for detecting violations of some of these models.

We exploited the regularity of communication patterns in process control networks. This involves specifying the expected communication patterns among the components in these networks.

Finally, we implemented a module for monitoring network services to detect (the appearance of) service changes in a control network. It has been our experience in enterprise systems that TCP service discovery is often security relevant. We have adapted that concept for monitoring Modbus TCP functions served by Modbus servers. The detection module uses heuristic techniques to learn the models pertaining to the availability of servers and services.

Our implementation is in the form of a multi-algorithm detection appliance based on the EMERALD detection and correlation framework. This appliance is integrated into a control system testbed implemented at Sandia National Laboratories. Initial experimental results provide evidence that model-based intrusion detection is a promising approach for monitoring process control networks.

As for future work, we are exploring a more systematic approach for building intrusion detection sensors from a formal model of Modbus devices. This involves building executable specifications in the PVS language, and using them online as detectors. This enables better detection of violations of the Modbus specification, including complex cases that involve multiple packets. To our knowledge, this is the first intrusion detection system that is in fact a formal model of the underlying specification.

About the Authors – Dr. Steven Cheung is a Computer Scientist in the Computer Science Laboratory of SRI International. His research interests include network security, intrusion detection, and sensor networking.

Dr. Bruno Dutertre is a Senior Computer Scientist at SRI International. His main research interests include formal methods, software architecture, and the application of logic, theorem proving, and model checking to the engineering of high-integrity systems. He has extensive experience in modeling and formal verification of real-time systems, security protocols, and other applications.

Martin W. Fong is a Senior Software Engineer with the Computer Science Laboratory at SRI International. He is a key developer for SRI's EMERALD intrusion detection software, and is the chief architect of the EMERALD M-Correlator prototype.

Dr. Ulf Lindqvist is a Program Manager in the Computer Science Laboratory of SRI International. He manages research and development programs, including process control security efforts for critical infrastructures and SRI's support for the US Department of Homeland Security's Cyber Security Research and Development Center.

Keith M. Skinner is a Senior Software Engineer with the Computer Science Laboratory of SRI International specializing in the research, design, implementation, and deployment of intrusion detection components and systems.

Alfonso Valdes is a Senior Computer Scientist with the Computer Science Laboratory at SRI International. He is an expert on statistical algorithms for detection and modeling and the application of such techniques in the information security arena. He has led statistical algorithm development in SRI's Next-Generation Intrusion Detection Expert System (NIDES) and later EMERALD.

References

[1] B. Dutertre. Formal modeling and analysis of the Modbus protocol. Technical report, Computer Science Laboratory, SRI International, October 2006.

[2] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. A sense of self for Unix processes. In Proceedings of the 1996 IEEE Symposium on Security and Privacy, pages 120–128, Oakland, California, May 6–8, 1996.

[3] C. Ko, M. Ruschitzka, and K. Levitt. Execution monitoring of security-critical programs in distributed systems: A specification-based approach. In IEEE Symposium on Security and Privacy, pages 175–187, Oakland, California, May 4–7, 1997.

[4] U. Lindqvist and P.A. Porras. Detecting computer and network misuse through the production-based expert system toolset (P-BEST). In Proceedings of the 1999 IEEE Symposium on Security and Privacy, pages 146–161, Oakland, California, May 9–12, 1999.

[5] Modbus IDA. Modbus application protocol specification v1.1a, June 4, 2004.

[6] Modbus IDA. Modbus messaging on TCP/IP implementation guide v1.0a, June 4, 2004.

[7] S. Owre, J. Rushby, N. Shankar, and F. von Henke. Formal Verification for Fault-Tolerant Architectures: Prolegomena to the Design of PVS. IEEE Transactions on Software Engineering, 21(2):107–125, February 1995.

[8] P. A. Porras and P. G. Neumann. EMERALD: Event monitoring enabling responses to anomalous live disturbances. In Proceedings of the 20th National Information Systems Security Conference, pages 353–365, Baltimore, Maryland, Oct. 7–10, 1997.

[9] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang, and S. Zhou. Specification-based anomaly detection: A new approach for detecting network intrusions. In Proceedings of the 9th ACM Conference on Computer and Communications Security, pages 265–274. ACM Press, 2002.

[10] A. Valdes and K. Skinner. Adaptive, model-based monitoring for cyber attack detection. In

[11] H. Debar, L. Me, and F. Wu, editors, Recent Advances in Intrusion Detection (RAID 2000), LNCS, Toulouse, France, Oct. 2000.