Chapter 7

DISTRIBUTED INTRUSION DETECTION SYSTEM FOR SCADA PROTOCOLS

Igor Nai Fovino, Marcelo Masera, Michele Guglielmi, Andrea Carcano and Alberto Trombetta

Abstract This paper presents an innovative, distributed, multilayer approach for detecting known and unknown attacks on industrial control systems. The approach employs process event correlation, critical state detection and critical state aggregation. The paper also describes a prototype implementation and provides experimental results that validate the intrusion detection approach.

Keywords: Industrial control systems, SCADA protocols, intrusion detection

1. Introduction

Critical infrastructures rely very heavily on information and communications technologies (ICT). These technologies provide features and services such as remote monitoring, remote management, intra-system coordination, inter-system communication and self-orchestration. Unfortunately, critical infrastructure assets are susceptible to a large number of ICT attacks [5, 10]. These attacks can be categorized into two classes. The first class includes traditional ICT attacks that leverage vulnerabilities in general purpose ICT systems; these attacks can be mitigated by adopting ICT countermeasures such as software patches, antivirus software and firewalls. The second class includes industrial system attacks that exploit vulnerabilities specific to industrial ICT systems, e.g., attacks that leverage the lack of authentication and integrity checks in SCADA communication protocols [1].

Due to the peculiarities of industrial systems, ICT countermeasures cannot be deployed efficiently in all environments. Indeed, the countermeasures are inadequate for dealing with attacks of the second class that exploit SCADA protocols. Also, even when countermeasures (e.g., signature-based intrusion detection) are successfully deployed, they may not protect against unknown

attacks. Such exposure is unacceptable as far as critical infrastructure assets are concerned.

This paper presents a novel approach for detecting attacks on industrial control systems based on the concepts of event analysis, event aggregation and correlation, and critical state detection. A prototype distributed intrusion detection system for monitoring SCADA systems is described. It uses event correlation to identify race conditions in critical states induced by malicious actions.

2. Background

Intrusion detection is a well-established field of research. The basic idea, presented in the mid-1980s (see, e.g., [3]), is to search for evidence indicating that a malicious attack is underway during a certain period of time. Intrusion detection systems (IDSs) can be classified according to (i) the source of the information used to detect intrusions; and (ii) the technique used to discriminate between licit behavior and malicious behavior. Discriminating IDSs based on their information source leads to their classification as network-based IDSs that analyze network traffic in search of malicious packets; and host-based IDSs that analyze host behavior for suspicious activities. Categorizing IDSs based on their discrimination technique gives rise to signature-based IDSs that detect attacks based on known attack patterns; and anomaly-based IDSs that detect attacks based on deviations from normal system behavior.

This paper focuses on network-based intrusion detection system (NIDS) architectures. Typical NIDS architectures incorporate a number of distributed sensors that analyze network traffic in search of attack signatures and anomalies. In the case of a SCADA system, a NIDS must be able to understand and analyze an industrial communication protocol such as Modbus, DNP3 or Profibus. These protocols, which were originally designed for serial communication, are currently embedded in TCP packets and ported over TCP/IP. Traditional NIDSs are unable to understand such "application level" protocols.

Digital Bond [4] has released a set of *ad hoc* rules for detecting certain attacks on the Modbus protocol. These rules specify unauthorized uses of the Modbus protocol, protocol errors and network scans. A traditional NIDS that incorporates these rules could identify primitive, single-packet-based attacks, in which the attacker sends a malicious packet to a Modbus device or uses a rare command. However, as shown in [10], SCADA attacks can be extremely complex and rarely involve a single step (i.e., the exploitation of a single vulnerability). Consequently, it is necessary to identify complex and dangerous attacks based on the analysis of different, low-risk atomic operations.

Several researchers have used such "attack correlation" techniques for intrusion detection in traditional ICT networks. Gross, *et al.* [6] have proposed a "selecticast" mechanism for collaborative intrusion detection that uses a centralized server to dispatch information about suspicious activities to intrusion detection sensors. Yegneswaran, *et al.* [16] employ a distributed overlay technique to monitor attacks. These approaches provide a broad picture regarding

suspicious events, but they are unable to identify complex malicious actions that are strategic as opposed to tactical.

Ning, *et al.* [15] have developed a model for identifying causal relationships between alerts on the basis of prerequisites and consequences. Likewise, Cuppens and Miege [2] engage pre-conditions and post-conditions in multiple analysis phases such as alert clustering, alert merging and intention recognition. This approach facilitates the automatic generation of correlation rules, but it can produce a large number of spurious rules that significantly increase noise in intrusion detection.

In summary, the correlation techniques described in the literature attempt to identify "malicious actions." However, in an industrial control system, routine actions can be used to implement devastating attacks. Consequently, an effective IDS for process control networks should be able to correlate licit actions and events in its search for malicious attacks.

3. Event Correlation and Anomaly Detection

As mentioned above, traditional IDSs often fail to detect complex attacks on process control networks. Most IDSs are unable to analyze SCADA communication protocols and detect attacks that exploit protocol vulnerabilities. The few IDSs that are able to analyze SCADA protocols (e.g., Snort) employ single-packet signatures; they cannot detect complex attacks where sequences of licit packets put the system in a critical state. Classical anomaly detection techniques also have limited effectiveness in industrial control environments. One of the major challenges is to specify and detect anomalies in process control networks that may have hundreds of PLCs.

Thus, we argue that a different intrusion detection approach must be devised for industrial control systems. Our approach is described as follows:

- A SCADA system is at the core of every process network in an industrial process system. A SCADA system controls every process in the industrial system. Therefore, the key to detecting intrusions is to monitor the activity of SCADA systems.

- Most industrial process systems are analyzed carefully and the possible "critical states" (or dangerous states) are usually identified,

- The data flowing between master and slave devices in a SCADA system can be used to construct a virtual image of the monitored system. The virtual states can be compared with the critical states that must be avoided. Upon tracing the evolution of the virtual states, it is possible to predict if the system is moving to a critical state.

- The industrial system is modeled in a modular fashion. A set of critical states for each subsystem comprising the industrial system is identified. The dependencies between subsystems are described so that the state of the system can be monitored. This enables the detection of many types

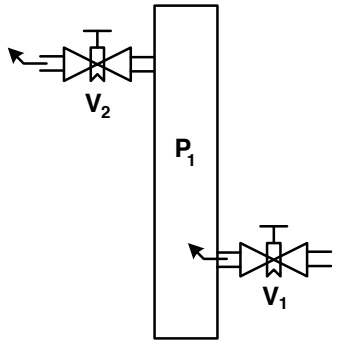

Figure 1. Example of a critical state.

of attacks. The effectiveness of this approach depends on the granularity of the virtual state representation and on the effects that attacks have on the evolution of the virtual states.

In general, anomaly detection defines the normal behavior of a system using a mathematical model and flags any deviation beyond a given threshold as a potential attack. We use a complementary method that identifies the critical states that are to be avoided (i.e., anomalous configurations that might put the system at risk) and flag them as the possible results of an attack. This approach is rarely employed in large, open ICT systems because it is almost impossible to describe all the possible combinations of behaviors that can drive a system into a critical state. However, industrial systems operate in tightly-controlled environments comprising electromechanical devices that react to a limited set of well-defined commands in relation to physical conditions that are known *a priori*. In these systems, the critical states are limited in number and are well-known; also, the task of describing them is more manageable than in traditional ICT systems.

The problem posed by false positives is addressed if attention is focused on identifying the sequences of events that could drive a system into a critical state. Moreover, focusing on system state evolution and on critical states (and not on specific attacks), makes it possible to detect new, unknown attacks that traditional signature-based IDSs would not be able to detect.

Figure 1 clarifies the notion of a critical state. The example system is a pipe P_1 through which high-pressure steam flows. The pressure is regulated by two valves V_1 and V_2.

An attacker with the ability to inject command packets in the process control network could direct the programmable logic controller (PLC) to close valve V_2 and open valve V_1. These two operations are perfectly licit when executed separately. However, the two operations performed in sequence put the system in a critical state because the pressure in P_1 could rise to a high enough level to cause the pipe to burst.

Table 1. Internal structure of PLCs.

Name	Object	Type	Comments
Discrete Inputs	1 bit	R	Provided by I/O system
Coils	1 bit	R/W	Alterable by application
Input Registers	16-bit word	R	Provided by I/O system
Holding Registers	16-bit word	R/W	Alterable by application

This simple example describes a scenario involving a two-command sequence. However, in general, the command sequences are long and complex, making it very difficult to specify their signatures for traditional IDSs. For this reason, we concentrate on the results of command sequences (i.e., the resulting states).

We represent critical states using our Industrial Critical State Modeling Language (ICSML). The current version of the language supports SCADA systems that use the Modbus protocol, but it is easily extended to accommodate other protocols.

The system under consideration is modeled in a modular manner as a collection of subsystems. Each subsystem is, in turn, modeled as a set of sub-subsystems, and so on. Thus, the overall system can be decomposed to the desired level of granularity.

At the most basic level, a system is composed of a set of PLCs, whose internal structure is essentially a sequence of registers and their corresponding values (Table 1).

ICSML is specified as follows:

```
<Critical State> ::= <term> | <Critical State><op><Critical State>
                     | NOT<Critical State> | (<Critical State>)
<op>             ::= AND | OR
```

As mentioned above, a system is decomposed in terms of subsystems, which are, in turn, decomposed in terms of sub-subsystems, and so on.

```
<system>         ::= <sysName>
                     | <sysName>[<component>.<componentList>]
                     | <system>.<system>
<componentList>  ::= <component>.<componentList> | e
<sysName>        ::= valid system name over the ASCII character set
```

Note that PLCs constitute the basic building blocks of a system. A component in ICSML represents a specific PLC (and its status) in a given subsystem. Each PLC in a Modbus network is identified by an IP address, port and identification number.

```
<component>      ::= PLC[<address>:<port>:<id>].<comp_status>
<address>        ::= <byte>.<byte>.<byte>.<byte>
<byte>           ::= 0 | 1 | ... | 255
<port>           ::= 0 | 1 | ... | 65535
```

```
<id>             ::= <byte>
<comp_status>    ::= CO[<reg_index>]<rel><bit>
                   | DI[<reg_index>]<rel><bit>
                   | IR[<reg_index>]<rel><word>
                   | HR[<reg_index>]<rel><word>
<reg_index>      ::= 0 | 1 | ... | 65535
<bit>            ::= 0 | 1
<word>           ::= 0 | 1 | ... | 65535
<rel>            ::= <= | >= | < | > | =
```

Using ICSML, it is possible to formally describe the critical states of a system and subsystems. For example, suppose that the following facts hold in the example above: (i) the output stream of valve V_1 is connected to PLC 192.168.0.1 port 502 id 1 and the holding register 10 contains 100 if V_1 is open and 0 if V_1 is closed; (ii) the input stream of valve V_2 is connected to PLC 192.168.0.2 port 502 id 1 and the holding register 20 contains 100 if V_2 is open and 0 if V_2 is closed; and (iii) the system is in a critical state if valve V_1 is open less than 50% and if valve V_2 is open more than 50%. Then, the critical state can be formalized in the following manner using ICSML:

```
PLC[192.168.0.1 : 502 : 1].HR[10] < 50 AND
PLC[192.168.0.2 : 502 : 1].HR[20] > 50
```

The relationships between subsystems are modeled using transition rules that specify what happens (in terms of changes in the values of components) in one or more subsystems given that something has changed in some other subsystem. The syntax of a transition rule is very simple. Given two components C_1 and C_2, a transition rule is an expression of the form $C_1 \rightarrow C_2$. The transition rule states that if the status of the first component is described by the expression C_1, then the status of the second component changes to that described by expression C_2.

ICSML permits the description of a set of critical states that represent – at the desired level of detail – unwanted occurrences of the subsystems contained in the industrial system under scrutiny. Also, ICSML permits the modeling of data flows between SCADA masters and slaves to create a virtual representation of the state of the entire system. The set of production rules should faithfully represent the industrial system and its subsystems along with the relationships between the various subsystems.

Of course, production rules alone are inadequate to ensure that changes in the virtual system states accurately model those in the industrial system over time. For this reason, it is necessary to periodically poll the SCADA network components to map changes related to external events (e.g., variations in sensor readings) to virtual states. Also, as we discuss below, it is important to recognize as early as possible that a state is evolving into a critical state.

3.1 High-Level Event Correlation

It would be useful to correlate all the licit low-level events to discover critical patterns that are potentially caused by malicious attacks. However, it is practically impossible to enumerate all the critical states of a complex system with hundreds of devices, let alone correlate all the low-level events.

Masera and Nai Fovino [9] have presented a methodology for modeling system features that are relevant to detection and correlation. The methodology expresses a complex system and its subsystems as a "system-of-systems" – a set of interconnected collaborative systems. A system is a set of independently-managed subsystems that provide services in a producer/consumer environment with a locality property. Thus, a system (which is also a subsystem) is recursively defined in terms of subsystems. Each subsystem has four attributes:

- **Name:** Uniquely identifies the subsystem.
- **Description:** Describes the purpose of the subsystem.
- **Service List:** Specifies the services provided by the subsystem.
- **Data Flow List:** Specifies data flows as tuples of the form $< S_1, S_2 >$, which represents a flow of information from subsystem S_1 to subsystem S_2.

The notion of a service provides the glue for describing a system-of-systems in terms of subsystems. Every action that a subsystem performs can be viewed as a service, and every subsystem communicates with other subsystems by providing services. Damage to a subsystem usually manifests itself as a corresponding lack in some service. Indeed, the modeling of a service has a central role in our critical state event correlation approach. Each service has four attributes:

- **ID:** Uniquely identifies the service.
- **Service Dependency List:** Specifies the service dependencies as tuples of the form $< Service, DamageThreshold >$ where $Service$ is the ID of a service and $DamageThreshold$ is the maximum level of damage to the service that can be tolerated.
- **Dependencies:** Describe the relations between the service and the services in the service dependency list using first-order logical expressions.
- **Service Value:** Specifies the value of the service (essential, valuable or expendable).

Thus, we represent a complex system in terms of subsystems, dependencies and services. Each subsystem is a black box that produces and consumes services (from which the data flows are derived). The subsystems, services and dependencies yield a system graph that represents the intrinsic interdependencies between the elements of the industrial system of interest.

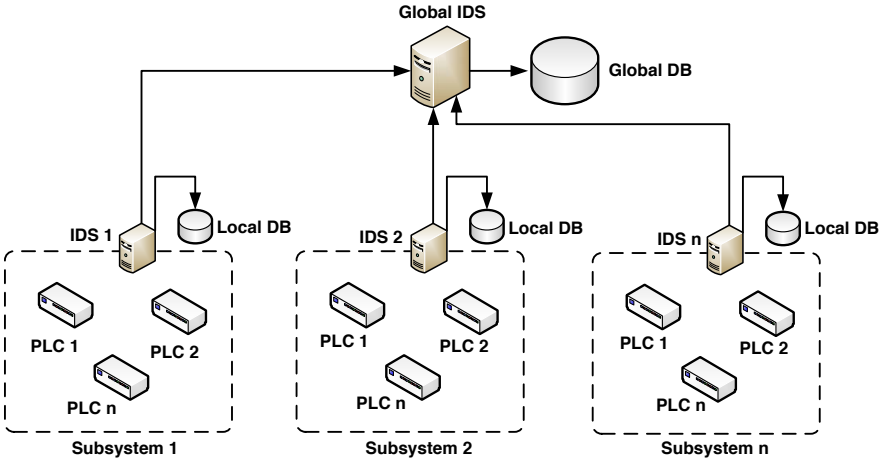

Figure 2. Global intrusion detection system architectures.

Using ICSML, it is possible to define high-level critical states in which each critical state represents a particular scenario where certain services provided by a collection of subsystems are partially or completely damaged, potentially moving the entire system into an unsafe state. The description of the critical states at this level is compact because it only involves the description of the failure of the higher-level services. In other words, a high-level critical state is reached when certain high-level services fail partially or totally, causing the system to move to a dangerous state.

The high-level and low-level event correlations can be merged using the following three-step procedure:

- When a low-level critical state is identified, information about the impacted subsystem services is delivered as an event to the high-level event correlation engine (HLEC).
- The HLEC propagates subsystem service failures across the whole system based on an exploration of the system graph.
- After the HLEC completes the propagation of service failures, it analyzes the entire system by comparing its general status with the set of high-level critical states in search of a match.

4. Architectural Overview

The distributed IDS is presented in Figure 2. The SCADA system under consideration is decomposed into component subsystems, each of which is monitored by a local IDS (figure 3). Each local IDS implements low-level correlation and detection, and raises specific alerts. The entire system is monitored by a

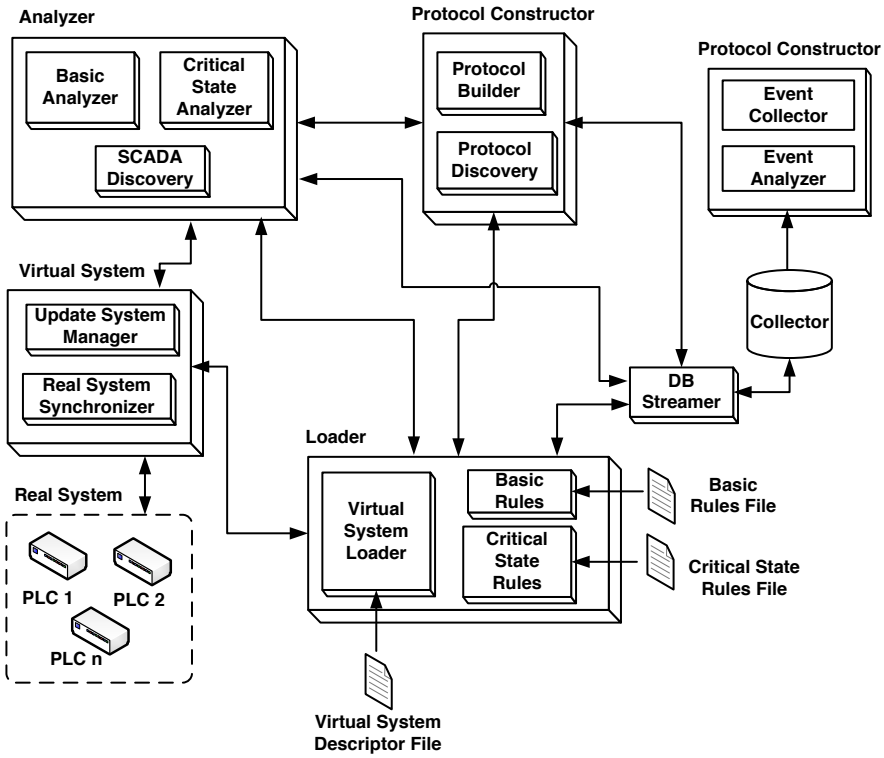

Figure 3. Local intrusion detection system architecture.

global IDS, which receives information from the local IDSs and implements high-level correlation and detection.

A local IDS has two information sources: network traffic flowing through the monitored subsystem, and direct queries that are periodically sent by the IDS to PLCs for information about their status. A local IDS contains a virtual representation of the monitored subsystem, and uses the two information sources to keep the local virtual system (LVS) in line with the real subsystem. The IDS identifies local unsafe states by comparing the state of the LVS with the set of critical states of the subsystem described using ICSML. Once a critical state is detected, the local IDS raises an alert and sends the list of subsystem services impacted by the critical state to the global IDS. The global IDS maintains a high-level virtual system, which is updated with the information received from the local IDSs. Then, the global state of the system is compared with the set of high-level critical states in search of a match.

4.1 Local IDS Prototype

The local IDS prototype incorporates five modules and fourteen functional components implemented in C# (Figure 3).

- **Loader (LS):** This module is responsible for initializing the system. It uses three XML files. The XML virtual system descriptor file contains the information used to create the virtual system. The basic rules file, commonly used in IDSs such as Snort, specifies malicious commands. The critical state rules file contains ICSML descriptions of rules related to critical states. The two rule files are imported and loaded into memory by separate functional components.
- **Protocol Constructor (PC):** This module contains the functional components that manage the construction and interpretation of SCADA protocols (currently Modbus and DNP3).
- **Analyzer (AZ):** This module monitors SCADA communications and performs single-packet detection, event correlation and local critical state detection.
- **Virtual System Manager (VSM):** This module stores a virtual representation of the SCADA system in memory. It updates the virtual system using the command flows captured by other components as input and by periodically querying the real system.
- **Database:** This MySQL database stores the alerts received from the analyzer.

4.2 Global IDS Prototype

The global IDS prototype receives critical state alerts from the local IDSs and correlates them to identify global critical states. The structure is simpler than that of the local IDS because it does not need modules to handle specific SCADA protocols. The global IDS has three modules:

- **Loader (LS):** This module is responsible for initializing the system. It uses two XML files, one to create the global virtual system image and the other to store the global critical states.
- **Global Virtual System Manager (GVSM):** This module manages the global virtual system, keeping track of its evolution and using as input the alerts received from the local IDSs.
- **Critical State Discovery (CSD):** This module analyzes the global virtual system to identify global critical states.

5. Experimental Tests

Several experiments were conducted to verify the performance of the distributed IDS. The SCADA testbed employed for the experiments reproduces the network, hardware and software used in a typical gas power plant [10]. In addition to evaluating IDS performance, the experiments analyzed the delays introduced by single-signature analysis, packet capture, virtual system updates and critical state analysis.

Data Rate Kbps	Alerts Expected	Alerts Detected
2.24	800	800
22.50	8,000	8,000
44.79	16,000	16,000
89.58	32,000	32,000
156.77	56,000	56,000
179.17	64.000	64,000
223.96	80.000	80,000
313.64	112,000	112,000
358.33	128,000	128,000
403.12	144,000	144,000
492.71	176,000	176,000

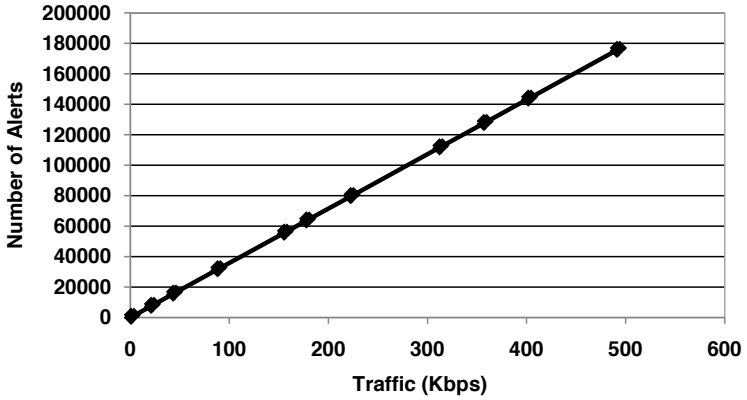

Figure 4. Alerts expected and alerts detected.

5.1 Single Signature Analysis

The amount of bandwidth that can be analyzed is a key critical issue in a NIDS. To evaluate this aspect, we implemented standard master/slave communications involving 100 request/response messages comprising 40 read requests, 50 write requests and 10 special functions. The IDS was configured to monitor Modbus and DNP3 traffic using a set of 2,000 *ad hoc* rules.

Figure 4 shows the results obtained when SCADA packets were sent simultaneously to a group of PLCs. The first column of the table shows the data rate (for an average SCADA packet size of 253 Bytes); the second and third columns show the numbers of expected alerts and detected alerts, respectively. Note that the IDS can analyze a large number of packets per second without packet loss.

Figure 4 also compares the number of expected alerts with the number of alerts raised by the IDS. Note that the IDS was able to raise all the alerts expected up to a data rate of 500 Kbps, which is very satisfactory given the

Table 2. Packet capture performance.

Requests Sent	100,000
Responses Sent	100,000
Request Size	315 Bytes
Response Size	315 Bytes
Request Rate	1 Request/ms
Traffic Rate	615.2 Kbps
Packet Loss	0

number of rules inspected and the low bandwidth available in industrial networks.

5.2 State-Based IDS

The state-based portion of the IDS has the largest impact on the real-time performance and, therefore, needs accurate control. For this reason, we carried out several tests, one for each step necessary to check the critical states of the system.

Packet Capture Packet loss is rare because "thread programming" was used to implement the IDS. Such losses can occur in the case of network congestion, but this is unlikely in a SCADA network. We tested the packet capture performance by sending a large number of packets at a very high bit rate.

The request/response transaction used in the packet capture experiment involved the master sending the slave a large request packet of 260 Bytes (maximum size allowed in Modbus); and the slave then responding with a large response packet of 260 Bytes. The request and response packets were both captured by the IDS.

The experiment to measure packet loss repeated this request/response transaction 100,000 times in order to generate a large amount of network traffic. The results are shown in Table 2. Note that the packet size is 315 Bytes (TCP header: 260 + 20 Bytes; IP header: 20 Bytes; Ethernet header: 15 Bytes).

No packet loss occurred for a burst of 100,000 packets at a rate of 615.2 Kbps (which is extremely high for a SCADA network). The experiment clearly demonstrates the reliability of the IDS.

Virtual System Updates The IDS updates the virtual system image in two steps: (i) it finds the PLC related to the content of the packet; and (ii) it updates the virtual object that represents the PLC.

The first step has no impact on IDS performance because the list of PLCs is stored in a hash table. The time required to find a PLC is the same for tables with 1 or 1,000 PLCs – around 0.0042 ms in our test environment.

Number of Coils	Average Time (ms)
1	0.0012168
50	0.0030485
100	0.0044824
500	0.0173109
1,000	0.0334344
2,000	0.0624535

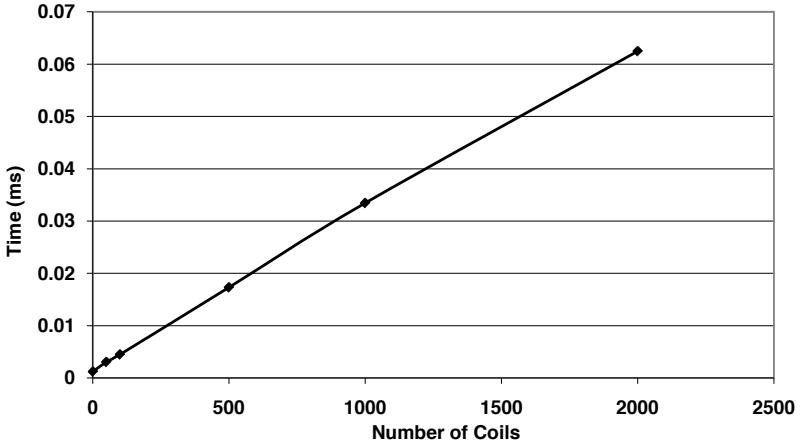

Figure 5. Virtual system performance test.

The second step takes more time, especially if it involves many registers or coils. Consequently, we conducted an analysis of PLC update times in a worst-case scenario. This worst-case scenario occurs when the IDS receives a packet with the function code 01 (read coils) with 2,000 coils to be read (maximum value in the Modbus specification [11–13]). The scenario requires the IDS to update the values of 2,000 coils.

Experiments were conducted for 1, 50, 100, 500, 1,000 and 2,000 coils to be updated. Figure 5 shows the average time taken to update values in the virtual PLC. The request/response transaction was repeated 1,000 times in order to obtain the average update time. As expected, the average time increases with the number of coils to be updated, but the increase is linear.

Critical State Analysis The performance of the critical state analyzer depends on two factors: (i) the number of conditions in each rule; and (ii) the number of rules.

To analyze the impact of rule size, we employed a request/response transaction with IDS capture and rule checking. The transaction involved the master sending a generic request to the slave; and the slave then sending the appropriate response. The IDS captured the request/response transaction and checked

Number of Conditions	Average Time (ms)
2	0.0204746
4	0.0217611
8	0.0244149
16	0.0301169
32	0.0370071
64	0.0550301
128	0.1206957
256	0.2127598
512	0.4226185
1,024	1.0706136

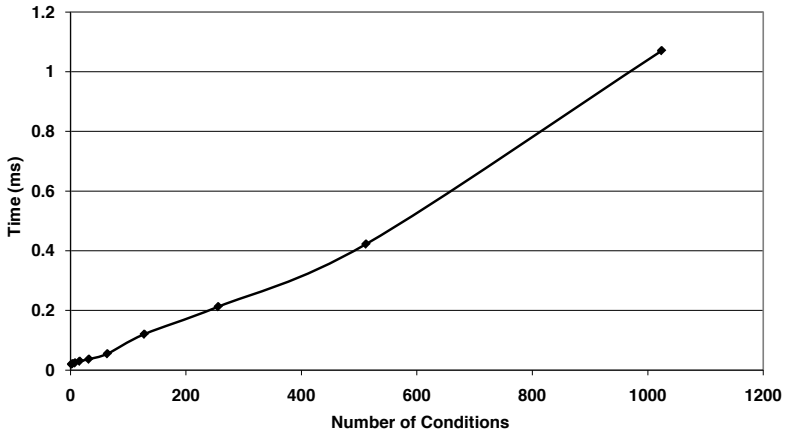

Figure 6. Critical state analyzer performance (Test 1).

if the virtual system entered into a critical state based on only one rule with a certain number of conditions.

Experiments were performed using rules with 2, 4, 8, 16, 32, 64, 128, 256, 512 and 1,024 conditions. In each case, the request/response transaction was repeated 1,000 times to obtain the average time for checking a rule.

Figure 6 shows the results of the experiments. Note that the elapsed time increases with the number of rule conditions and that the growth is linear.

Similar experiments were conducted to evaluate the impact of the number of rules. However, in this case, each rule had two conditions. The experiments used 10, 50, 100, 500, 1,000 and 2,000 rules. The results are shown in Figure 7. Note that the elapsed time increases with the number of rules and that the increase is linear. The results also demonstrate that critical state rules analysis is the performance bottleneck because it requires the most time of all the IDS operations.

Number of Rules	Average Time (ms)
10	0.1123061
50	0.5153591
100	1.0248889
500	2.6010271
1,000	5.0175991
2,000	9.9285867

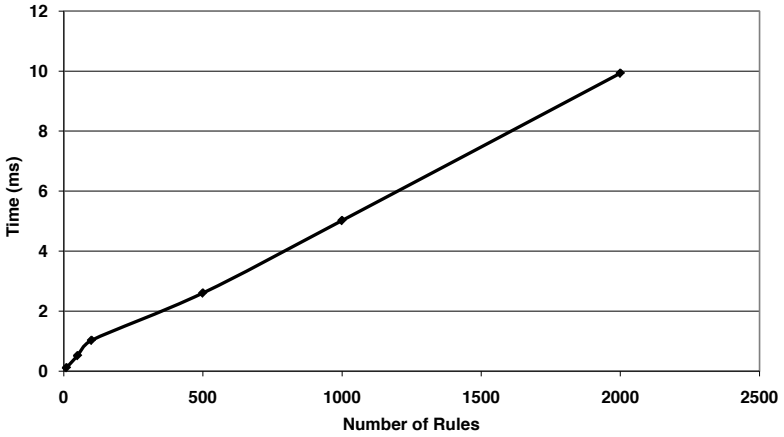

Figure 7. Critical state analyzer performance (Test 2).

6. Conclusions

The distributed intrusion detection approach developed for industrial control environments takes into account the state of the system of interest instead of attack signatures and anomaly heuristics. The approach rests on the assumption that the ultimate goal of an attacker is to put the system into a critical state. Consequently, instead of searching for the evolution of an attack, the approach tracks the evolution of the system. This approach addresses problems posed by false positives and permits the detection of unknown attacks.

Experimental results indicate that the IDS prototype exhibits good performance with respect to packet capture, virtual system updates and critical state analysis. Our future research will extend the prototype for application in a real-world industrial control environment. In addition, we plan to incorporate a critical state prediction feature, which will anticipate the evolution of the system into a known critical state on the basis of local sensor information.

References

[1] A. Carcano, I. Nai Fovino, M. Masera and A. Trombetta, SCADA malware: A proof of concept, presented at the *Third International Workshop on Critical Information Infrastructure Security*, 2008.

[2] F. Cuppens and A. Miege, Alert correlation in a cooperative intrusion detection framework, *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 202–215, 2002.

[3] D. Denning, An intrusion-detection model, *IEEE Transactions on Software Engineering*, vol. 13(2), pp. 222–232, 1987.

[4] Digital Bond, Modbus TCP IDS signatures, Sunrise, Florida (www.digitalbond.com/index.php/research/ids-signatures/modbus-tcp-ids-signatures).

[5] G. Dondossola, J. Szanto, M. Masera and I. Nai Fovino, Effects of intentional threats to power substation control systems, *International Journal of Critical Infrastructures*, vol. 4(1/2), pp. 129–143, 2008.

[6] P. Gross, J. Parekh and G. Kaiser, Secure selecticast for collaborative intrusion detection systems, *Proceedings of the International Workshop on Distributed Event-Based Systems*, 2004.

[7] M. Masera and I. Nai Fovino, Modeling information assets for security risk assessment in industrial settings, *Proceedings of the Fifteenth EICAR Annual Conference*, 2006.

[8] M. Masera and I. Nai Fovino, Models for security assessment and management, *Proceedings of the International Workshop on Complex Network and Infrastructure Protection*, 2006.

[9] M. Masera and I. Nai Fovino, A service-oriented approach for assessing infrastructure security, in *Critical Infrastructure Protection*, E. Goetz and S. Shenoi (Eds.), Springer, Boston, Massachusetts, pp. 367–379, 2007.

[10] M. Masera, I. Nai Fovino and R. Leszczyna, Security assessment of a turbogas power plant, in *Critical Infrastructure Protection II*, M. Papa and S. Shenoi (Eds.), Springer, Boston, Massachusetts, pp. 31–40, 2008.

[11] Modbus IDA, MODBUS Application Protocol Specification v1.1a, North Grafton, Massachusetts (www.modbus.org/specs.php), June 4, 2004.

[12] Modbus IDA, MODBUS Messaging on TCP/IP Implementation Guide v1.0a, North Grafton, Massachusetts (www.modbus.org/specs.php), June 4, 2004.

[13] Modbus.org, MODBUS over Serial Line Specification and Implementation Guide v1.0, North Grafton, Massachusetts (www. modbus.org/specs.php), February 12, 2002.

[14] I. Nai Fovino and M. Masera, Emergent disservices in interdependent systems and system-of-systems, *Proceedings of the IEEE Conference on Systems, Man and Cybernetics*, vol. 1, pp. 590–595, 2006.

[15] P. Ning, Y. Cui and D. Reeves, Constructing attack scenarios through correlation of intrusion alerts, *Proceedings of the Ninth ACM Conference on Computer and Communications Security*, pp. 245–254, 2002.

[16] V. Yegneswaran, P. Barford and S. Jha, Global intrusion detection in the DOMINO overlay system, *Proceedings of the Network and Distributed System Security Symposium*, 2004.