July 2012 UILU-ENG-12-2206

ADAPTING BRO INTO SCADA: BUILDING SPECIFICATION-BASED INTRUSION DETECTION SYSTEM FOR DNP3 PROTOCOL

Hui Lin, Zbigniew Kalbarczyk, Ravishankar K. Iyer

Coordinated Science Laboratory
1308 West Main Street, Urbana, IL 61801
University of Illinois at Urbana-Champaign

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB NO. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE July 2012	3. REPORT TYPE AND DATES COVERED

4. TITLE AND SUBTITLE Adapting Bro into SCADA: Building Specification-based Intrusion Detection System for DNP3 Protocol	5. FUNDING NUMBERS DE-OE0000097
6. AUTHOR(S) Hui Lin, Zbigniew Kalbarczyk, and Ravishankar K. Iyer	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, 1308 W. Main St., Urbana, IL, 61801-2307	8. PERFORMING RGANIZATION REPORT NUMBER UILU-ENG-12-2206
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) U.S. Department of Energy, 1000 Independence Ave. SW, Washington, DC 20585 U.S. Department of Homeland Security, Washington, DC	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Modern SCADA systems are increasingly adopting Internet technology to control industry processes. With their security vulnerabilities exposed to public networks, an attacker is able to penetrate into these control systems to put remote facilities in danger. To detect such attacks, SCADA systems require an intrusion detection technique that can monitor network traffic based on proprietary network protocols. To achieve this goal, we adapt Bro, a network traffic analyzer widely used for intrusion detection, for use with SCADA systems. A built-in parser in Bro supports DNP3, a network protocol that is widely used in SCADA systems for electrical power grids. By exploiting Bro's intrusion detection features, we apply a specification-based technique to analyze the parsed traffic. This built-in parser provides high visibility of network events in SCADA systems. Instead of exploiting an attack signature or a statistical normal pattern, SCADA-specific semantics related to each event are analyzed. Such analyses are made in terms of defined security policies which can be included at runtime. Our experiments are carried out in a laboratory-scale SCADA system environment with well-formatted but malicious network traffic. The detection capability and performance of the Broadapted intrusion detection system revealed in experiments show its potential applicability in the real SCADA system environment.

14. SUBJECT TERMS Intrusion detection; Security specification; Critical infrastructure; Bro; SCADA	15. NUMBER OF PAGES 12
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL

Adapting Bro into SCADA: Building Specification-based Intrusion Detection System for DNP3 Protocol

Hui Lin, Zbigniew Kalbarczyk, Ravishankar K. Iyer
Electrical and Computer Engineering Department
University of Illinois at Urbana Champaign
(hlin33, kalbarcz, rkiyer)@illinois.edu

Abstract

Modern SCADA systems are increasingly adopting Internet technology to control industry processes. With their security vulnerabilities exposed to public networks, an attacker is able to penetrate into these control systems to put remote facilities in danger. To detect such attacks, SCADA systems require an intrusion detection technique that can monitor network traffic based on proprietary network protocols.

To achieve this goal, we adapt Bro, a network traffic analyzer widely used for intrusion detection, for use with SCADA systems. A built-in parser in Bro supports DNP3, a network protocol that is widely used in SCADA systems for electrical power grids. By exploiting Bro's intrusion detection features, we apply a specification-based technique to analyze the parsed traffic. This built-in parser provides high visibility of network events in SCADA systems. Instead of exploiting an attack signature or a statistical normal pattern, SCADA-specific semantics related to each event are analyzed. Such analyses are made in terms of defined security policies which can be included at runtime. Our experiments are carried out in a laboratory-scale SCADA system environment with well-formatted but malicious network traffic. The detection capability and performance of the Bro-adapted intrusion detection system revealed in experiments show its potential applicability in the real SCADA system environment.

I. Introduction

SCADA (Supervisory Control and Data Acquisition) systems are distributed computer systems used for monitoring and controlling manufacturing processes. Modern SCADA systems, such as those for controlling electrical power grids, adopt general Internet communication technology to improve control efficiency. However, exposing control systems to public networks opens a door for security threats from which such critical infrastructures were previously isolated.

In contrast to regular enterprise systems, SCADA systems have unique security objectives. Although the confidentiality of system data is critical, SCADA systems are more concerned with manufacturing process availability and integrity, which has become a priority with the recent discovery of real and complex attacks. [7], [18].

A. Challenges of Applying Traditional Intrusion Detection Systems

Traditional Intrusion Detection Systems (IDSs), either signature-based or anomaly-based, cannot be directly applied to SCADA systems. Lack of analysis on real attacks in control environments makes it impossible to apply signature-based techniques. On the other hand, anomaly-based techniques, based on trained normal patterns, suffer from two drawbacks: an unbearable rate of false positives and inability to detect malicious events, either accidentally or intentionally, following those normal patterns.

Additionally, application of traditional IDSs is also challenging because they always lack sufficient capabilities to investigate network traffic based on unique proprietary protocols in SCADA systems. This insufficiency prevents in-depth analysis on network activities, which blinds traditional IDSs to SCADA-specific attacks.

B. Proposed Contributions

We introduce a specification-based intrusion detection framework [24] that provides high visibility of semantics carried by proprietary network protocols. Specifically, we adapt Bro [1], [23], a real-time network traffic analyzer, to integrate parsers of proprietary network protocols used in SCADA systems for electrical power grids, such as DNP3, a built-in parser that collects SCADA network events. Instead of exploiting an attack signature or a statistical pattern, the ID analyzes SCADA-specific semantics related to each event. To provide good portability and extensibility, analyses are run with defined security policies implemented.

As proof-of-concept, several security policies are implemented and integrated into our IDS framework to demonstrate how network semantics are analyzed. Our contribution includes redefining signature-based and anomaly-based detection methods based on such SCADA system semantics. Furthermore, an additional policy is proposed based on DNP3 definitions to guarantee the integrity and availability of control processes. In this policy, SCADA system operations are differentiated by the type of access made on remote facilities. Type-specific requirements are subsequently defined for the transmitted information to guarantee that an authorized operation is always performed.

The remainder of the paper is organized as follows. In section II, we analyze security threats and present our assumptions in this work. Section III describes in detail how the DNP3 analyzing framework is built based on Bro. In section IV, we introduce example security policies that analyze SCADA systems through network semantics. Section V presents experimental setup and the processing overhead due to the DNP3 analyzer. In section VI, we implement the proposed security policies and use them to analyze well-formatted but malicious network traffic. Related works are introduced in section VII. We conclude and describe future work in the final section.

II. Security Threats in SCADA Systems

A. System Architecture of SCADA

SCADA systems are distributed systems used to control geographically dispersed facilities. Centralized data acquisition and control on the remote facilities are critical to appropriate system operation. Figure 1 presents components of the SCADA systems that are commonly used in electrical power grids. Other industry control systems, such as gas and oil pipelines, wastewater control systems, share a similar communication structure [25].

The *control center* includes several human interface computers and internet communication equipment. Its major responsibility is to acquire measurement data from a remote site. Based on the collected data, operators estimate remote devices state and issue control commands. All these process information are logged locally, such as in data historians.

The *field site* is a remote environment which contains proprietary field devices and primitive actuators and sensors. The major responsibility of the field devices is to locally control and collect measurement data from actuators and monitor sensors through a fieldbus.

The *control network* is a long distance channel connecting the control center and the field site, on which proprietary protocol is used to transmit information. Many currently used proprietary protocols, such as DNP3 and Modbus, are built over TCP/IP protocols in order to transmit information on latest Internet communication technology.

B. System Interpretation

In order to describe behavior in SCADA systems, we define the term operation. In the context of this paper, an *operation* can be regarded as a basic unit of function to modify the state of a field site, such as devices outputs or configurations. The effect of an operation, whether is malicious or not, depends on the initial state of the field site before the operation and the content of the operation.

Figure 2 presents an abstract example on how operations can affect the state of a field site. In the example, we assume that the field site has only two devices and at least one of them should be in an open state to maintain some industry processes. The state of the whole field site can be represented as combinations of the open/close outputs of two devices. In this example, Close Dev B is actually a malicious operation even though it may be transmitted in the form of well-formatted network packets as it puts the field site in a hazardous state (both devices are closed).

In a finer granularity level, each operation may consist of several rounds of network communications. With collected semantics from all network packets, it is possible to guarantee that each operation is performed in an authorized manner.

C. DNP3: Distributed Network Protocol

SCADA systems adopt proprietary network protocols to transmit measurement data and control commands. In the context of this paper, a highly structured protocol, Distributed Network Protocol (DNP3) is analyzed as a representative as it is widely used in electrical power grids, water controls, and other industry control systems [15].

DNP3 was initially built over serial lines. The protocol is divided into three layers: application layer, transport layer and data link layer. This hierarchy, however, cannot directly be matched to their corresponding layers in the TCP/IP stack. As a result, all three DNP3 layers are grouped together as a single application layer fragment over the TCP layer (Figure III). Because of this, DNP3 is currently also referred as DNP3 over TCP. In this paper, we refer the DNP3 network packet as the whole payload over TCP protocol. Our DNP3 parser works all the way up to the DNP3s original application layer in which SCADA systems semantics are located.

DNP3 supports five different types of data for remote devices: analog input, binary input, counter input, control output and analog output. A typical DNP3 request includes (1) function code- specifying controls that are performed on a field site; and (2) indices of field devices on which the control is performed. To respond to a request, field devices issue a response message carrying the requested measurement data or a state flag if an error occurs. DNP3 also supports unsolicited responses, communication initiated by the field site without a request to indicate significant events.

D. Threat Model

In this section, we present security threats faced by SCADA systems. We discuss these threats component-by-component, examining how an attacker can issue malicious operations to

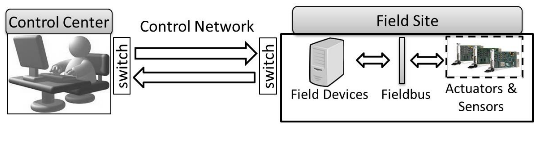


Fig. 1. General Layout of SCADA Systems used in Electrical Power Grids

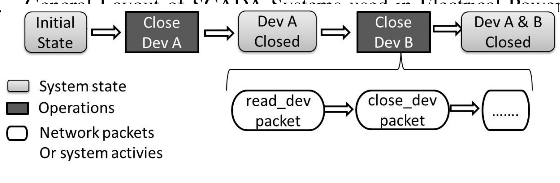


Fig. 2. Abstract Example of Control over Two-Device Field Site

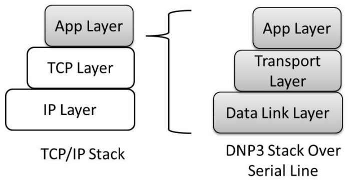


Fig. 3. Support DNP3 Over TCP

evade detection by traditional protection mechanisms such as firewall and anomaly detection.

Real attack incidents have revealed various penetration pathways into the control center have been discovered in real attack incidents [26], [12], such as dial-in modems and wireless access points (Figure 5 in [12]). Attackers who penetrate the control center successfully can then issue well-formatted but malicious operations that can put the field site into a hazardous state. These operations can follow the same traffic pattern as normal network packets, modifying only payload carrying commands or measurements.

Historically, the control network has been another entry point vulnerable to attack, as information is usually transmitted in plain texts. Recently, several authentication and encryption protocols that extend DNP3 have been designed to guarantee the integrity and confidentiality of network traffic [15], [22]. Following this design momentum, we assume that the network traffic cannot be compromised during the transmission over the control network.

Field devices deployed in the field site are vulnerable to rootkits or malware that can corrupt the devices, causing them to hide the correct state of the field site and communicate with the control center in a manner adopted by the attacker [18]. By estimating the wrong state, the control center may also issue wrong operations to the field site, generating unexpected or disastrous consequences.

The last entry point of attacks is the primitive actuators and sensors, which could be exploited to generate false or malicious data measurements. Based on these data, the control center may estimate a faulty state of the field site and thus issue wrong operations [21]. But to successfully perform such attacks, an adversarys physical access to the field site is usually required. From [10], [20], we learn that this type of attack could be avoided and detected as long as sufficient numbers of actuators and sensors are properly protected.

In the context of this paper, *we assume that we can trust (1) the transmission of the control network; (2) measurement values from primitive actuators and sensors located in the field site. Our purpose is to protect cyber-attacks which could penetrate into SCADA systems through the control center and the field devices.*

III. The DNP3 Analyzer: Intrusion Detection Framework based on Bro

A. Adapt Bros Network Analysis Framework in SCADA Systems

In this paper, we adapt Bro into SCADA to build an intrusion detection system. Bro is a real-time network traffic analyzer which is widely used in forensic analysis, intrusion detection and other network-related analysis [21]. As shown in figure 4, Bro is conceptually divided into protocol parsers (event engines), event handler APIs, and policy modules interpreter.

Our work to adapt Bros features into SCADA systems is also highlighted figure 4. We start it by building new parsers of proprietary network protocols. This built-in parser collects SCADA system specific events. The semantic information carried by each event, such as issued commands or collected measurement results, is delivered to the corresponding event handler APIs. These APIs could be implemented by different policy modules to perform analyses on the SCADA-specific events. The separation of events and policy make it convenient to specify different processing methods under different context.

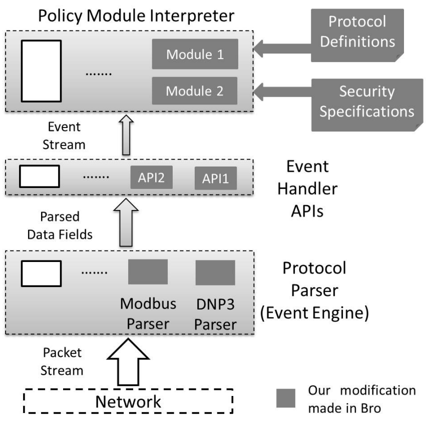


Fig. 4. Adapt Bros in SCADA Systems

At this stage, parsers of both DNP3 and Modbus protocol are integrated into Bro. Similar to DNP3, Modus protocol is also used to establish communications between the control center and the remote field site. DNP3 and Modbus protocol share many representative features in terms of the structure of network packets, data types. In this paper, we will focus our discussion on DNP3. So for the rest of the paper, *the DNP3 analyzer* is used to refer to our IDS framework with DNP3 parsers and some example policy modules (explained later in detail) being included.

B. DNP3 Parser

The parser's primary responsibility is to translate network byte streams into meaningful data fields according to a protocol definition.

The main body of Bro is written in C++. However, we are not directly developing the DNP3 parser in this complicated high-level programming language. A compiler-assisted method named as *binpac* is used to shorten the development period and avoid logical errors. For instance, in our parser design, the structures of data fields are commonly complex. As a result, a field is recursively disassembled into smaller sub-fields until it can be represented by basic data types such as integers or bytes. Directly representing this structure and properly managing memory allocations in C++ easily introduces bugs and logical errors.

Figure 5 illustrates a design procedure using binpac. DNP3 definitions are first represented by the binpac script, which is specifically designed to represent the hierarchical structure of a network protocol. With the help of the binpac compiler, binpac scripts are then automatically translated into C++ codes. The resulting DNP3 parsers in C++ codes are finally integrated into Bro with few additional lines of codes.

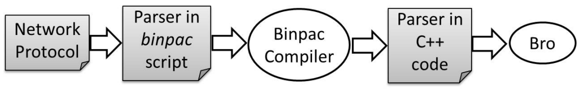


Fig. 5. Binpac Development Procedure

C. Event Handler APIs

Once the DNP3 parser finishes decoding, a network may occur which contains certain data fields or even all fields in a network packet. For example, a *dnp3_crob* event is generated by the DNP3 parser once a command to control relay output is found in the dnp3 request. The contents of the command are properly parsed and collected by the corresponding API.

Event handler APIs are predefined interfaces between policy modules and the DNP3 parsers. We provide a prototype for each event handler API, including a name and its arguments list. These APIs work in call-back mechanisms. The value of each argument is updated by the parsers. If the body of an API is implemented as part of a policy module, the protocol parser is intercepted with the execution of all implemented event handlers.

The declaration of event handler APIs and their implementation are separated in order to facilitate adapting security policy dynamically. Our design principle is to declare as many APIs as possible to cover all application-level semantics from DNP3 network packets. Selective APIs are defined at runtime according to a system's ability to perform accurate and efficient detection.

D. Policy Modules

Through the event handler APIs, policy modules can access parsed DNP3 semantic information. Each policy module is implementing selective event handler APIs by the Bros specified script language. This type-safe language includes normal arithmetic and logic operations, conditional statements, loop statements, and other functionalities that support intelligent analyses. Scripts are then interpreted and executed to get analysis results.

Policy modules can be implemented according to both protocol definitions and security specifications. As a result, some policy modules may be used to validate whether the monitored network packets conform to protocol definitions. Other policy modules, however, can perform additional validations on network packets, such as sequence validations, in order to detect malicious network behaviors. In the context of this paper, several example policy modules are described in detail in Section XXX.

E. Deployment of Bro-based IDS

Figure 6 illustrates the common architecture that connects SCADA systems and corporate environments. The detailed description of this architecture can be found in [17]. The architecture is logically divided into four zones: external environments (Zone 1); corporate environments (Zone 2); public accessible control systems (Zone 3); and SCADA systems (Zone 4). Zones are separated from one other by firewalls and demilitarized zones (DMZs).

Traditional IDSs are deployed to monitor network activities at critical points of corporate environments and public accessible environments, such as entry points of firewalls and DMZs. However, SCADA systems are left unmonitored in the original design, as traditional IDSs are not able to analyze SCADA-specific semantics from network traffic.

The proposed Bro-adapted IDS is used to monitor SCADA systems located in security-blind areas. The Bro-adapted IDSs can be deployed at entry points of control system LANs or fieldbus networks to monitor all network traffic. This purpose

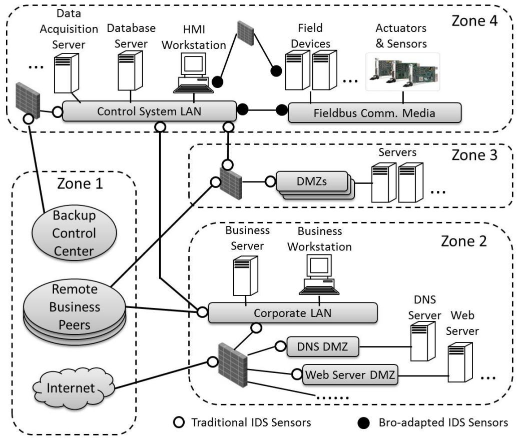


Fig. 6. IDS Deployment in Control and Corporate Systems

can be achieved by configuring hardware such as span ports or mirroring ports in most commercial switches or routers. In some situations, engineers are also able to connect human machine interface (HMI) workstations or field devices internally for configuration, such as via VPN or wireless connections. Whereas this internal connection, which bypasses perimeter networks, can be exploited as a backdoor, Bro-adapted IDS can also be directly deployed on those machines in order to reduce such a risk.

F. Use Bros Extension in SCADA Systems

Since the Bro-adapted IDS, specifically the DNP3 analyzer described in this paper, is built on Bro, many of Bros existing features can be used to enhance analysis. Bro is a stateful tool such that states of certain network packets can be stored in its address space. Consequently, events from different network packets can be easily correlated.

Another important feature is Bros capability to interact with host systems. The Bro client communication library (*broccoli*) can be used to write a client program that generates events Bro can understand [32]. In one of our example policies, *broccoli* is used to connect the DNP3 analyzer to some proprietary system analysis software.

IV. Example Policy Modules

In this section we present example policy modules which are based on protocol definitions or security specifications. By presenting these policy modules, we are not claiming that they formally conform to DNP3 or provide complete intrusion detection in SCADA systems. Our goal is to present several examples that demonstrate how SCADA system events are analyzed through the DNP3 analysis framework.

A. Policy Module based on Protocol Definitions

The purpose of this policy module is to ensure that the collected network packets are following protocol definitions. In the context of this paper, the conformance to DNP3 is maintained by intra-protocol and inter-packet validations. The policy module processes semantic information collected by necessary event handler APIs and generates run-time alerts once ill-formatted network packets are detected.

1) Intra-Packet Validation: A network packet consists of different data fields, such as application packet header and function code. A data field is served for a specific function for the whole network packet. Protocol-level validation verifies value of each data field.

For each single field, two aspects of information are verified. The first aspect is the analysis of valid value ranges. For example, DNP3 uses an 8-bit byte to represent a function code. However, 37 out of 256 possible values are treated as legal, and the remaining values are reserved for future use.

The second aspect involves the dependencies between fields within the same network packet. In DNP3, the structures of certain data fields are not statically defined but could vary according to the values in other fields such as function code, object type, etc.

2) Inter-Packet Validation: In addition to defining rules for the data fields within a network packet, DNP3 also defines rules between packets. For example, a SELECT packet is used in conjunction with OPERATE packet. The control center first sends the SELECT packet to select the remote field devices. After the SELECT is properly confirmed, the OPERATE packet is issued to operate on the field devices which were exactly selected before.

In order to perform this validation, our proposed IDS retains a record of previously parsed network packets. While an incoming packet is being parsed, a policy module relies on the both stored and newly parsed values to validate their dependencies.

B. Policy Module based on Security Specifications

In many situations, even if the network packets pass the intra-packet and inter-packet validation, the network traffic still contains malicious packets. According to our threat model, malware can be installed in the control center, so it is possible to perform man-in-the-middle or replay attacks by modifying specific application semantics in network packets. For instance, if the control center is required to send a command to close a breaker in order to avoid hazardous states in the field site, an attacker can modify the command to open it by simply flipping a single bit which represents open/close status.

We believe that these process-level attacks are more dangerous than traditional denial-of-service attacks, as they neither generate ill-format network packets nor produce anomaly network traffic pattern. If application-level semantics are not carefully analyzed, the attacker can easily evade detection.

With such semantic information available from the DNP3 analyzer, we can implement policy modules using methods whose implementation would be complicated or impossible for traditional IDSs. In the following sections, these semantic-driven methods based on signatures, statistical normal patterns, or system-specific specifications are illustrated.

1) Signature-based Detection: Unlike signatures in terms of byte patterns found in network packets, we can use critical states of a remote field site to define signatures. Take the power grid environment as an example, number of circuit breakers and other electrical instruments are used to control substations in remote fields. Circuits breakers are usually not all opened or closed as the power generated and consumed should be appropriately balanced. So it is critical to prevent them from entering into certain configuration which would put whole systems into danger.

This type of critical configurations would be predefined by system operators as a signature. By learning application semantics related to such configurations, the DNP3 analyzer can validate such signature successfully.

C. Anomaly-based Detection

Although some critical configurations may be able to directly describe the system state, system physical states are determined by indirect state variables in most practical cases, because those indirect state variables are usually too complex to be directly measured.

For example, in the electrical power grid, system state is uniquely defined by voltage phasors in each bus. However, directly measuring the magnitude and angle of the voltage in real time is impractical. As a result, other feasible measurements, such as power injected or power consumed, are collected to estimate the system state. This process, which is called state estimation, is an important technique for analyzing the security of power grids. State estimation results directly inform the operator whether the monitored system is in a normal or abnormal state. Traditionally, the control center in the electrical power grid performs state estimation for all connected field sites. With the introduction of computation capabilities to field sites, local states estimation can also be performed to reduce the control center's workload.

As in the case of a electrical power grid environment, many SCADA system environments adapt their system-specific techniques to perform security analysis. These techniques usually require large number of computations, which would be difficult to implement in traditional IDSs. Due to this challenging requirement, we use *broccoli* (Bros client communication library) to build a communication channel between the DNP3 analyzer and proprietary SCADA system analysis software. In other words, the DNP3 analyzer performs like a front-end of the analysis which collects semantic information at run time and leaves the actual information processing to the back-end software engine that is common in SCADA systems.

1) Specification-based Detection: Specification-based methods allow system operators to define security policies based on the system's capabilities. In addition to alerting on critical system states, alerts generated when security policies are violated would provide more context information on how attacks penetrate into systems.

In this section, we propose a recommended security policy based on the understanding of DNP3. The purpose of this security policy is to guarantee the integrity and availability of field sites by validating each operation. Additional policies concerned with specific system implementations can further be integrated into the analyzer.

Definition of Security Policy

To better define the policy, we represent SCADA systems in terms of objects, subjects, and operations in the context of DNP3.

By the definitions of DNP3, a request is initiated from the control center to perform an operation on field sites. A DNP3 request usually contains a command indicating what operations to perform and an index of target devices. The subject refers to an operator in the control center and can be defined as a normal user or administrator (root user). Administrators have higher privilege than normal users.

The object refers to resources in the remote field site and can be classified into two types: 1) actuators and sensors whose integrity and availability can affect industry processes, and 2)field devices responsible for local data acquisition and control over actuators and sensors. Software and hardware configurations are usually associated with the field devices.

Operations, the concept that we introduced in Section III.B, are initiated by the subject to provide certain service to the object. In this paper, we propose an original nomenclature to distinguish operations according to the type of access an operation makes to the field site. The terminology of access control policy in operating system design classifies operations into three types: *read*, *write*, and *execute*. The meanings of these terms are tuned to fit into the context of SCADA systems. *Read* operations, for example, are used to learn the state of field sites by collecting measurements stored in the field devices as well as the actuators and sensors. **Write** operations, such as configuration updates, are issued to handle field devices. Finally, execute operations are used to change the state of actuators and sensors such that industry processes are properly maintained.

Our security policy is expressed as a set of specific rules established for each operation type.

1. *Read* operations can be issued from normal users (non-administrator users) and the collected data must *maintain its integrity*.
2. Before issuing *write* operations from a control center to a field site, an administrator (root) must authenticate himself to log in into the control center and the issued network packets must *maintain its integrity* before its execution in the field site.
3 Before issuing *execute* operations from a control center to a field site, an administrator (root) must authenticate himself to log in into the control center and a *read* operation which obeys rule 1 must be executed beforehand. The issued command must *maintain its integrity*.

For the r*read* operation, it must be guaranteed that the data *read* from the field site truly reflects its state. The *write* operation, according to rule 2, should only be issued by an administrator with a root password. Commands sending with the *write* operation must maintain its integrity. The threat is that attackers may exploit the *write* operation to misconfigure the field devices or even install rootkits/malware in them. For the *execute* operation, a temporal dependency is defined that it must always follow an appropriate *read* operation. The purpose of this rule is to guarantee that a system administrator is always aware of the current state of the field site (through the first issued *read* operation) whenever an *execute* operation is issued.

Detection Mechanisms

To enforce the predefined security policy, intra- and inter-operation rules must be maintained. For example, in rule 2 and rule 3, authentications and DNP3 network activities are correlated to guarantee that the operation is trusted.

Both intra- and inter-operation rules are established as temporal dependencies between operation types. As the DNP3 analyzer collects semantics information from network packets at runtime, such as each operation's type of access, temporal dependencies can be validated by storing and comparing operation types carried by network packets.

V. Performance Analysis on DNP3 Analyzer

In this section, we focus on experiments concerned with the DNP3 parser, which is served as the foundation for further security analysis in the DNP3 analyzer.

A. Setup of Experiment

All experiments (including security analysis in Section VIII) are carried out in an laboratory-scale emulated SCADA system consisting of proprietary hardware and software commonly found in todays power grid environment. (Figure 7). The hardware configuration includes a simulated control center and a remote field site, which are configured into two different logical or virtual local area networks by a connected switch. A control center consists of a Windows XP workstation that communicates with the field site to request measurement data or issue control commands. The field site includes a SEL 3530 Real Time Automation Controller (RTAC) [6] and a SEL 421 Protection and Control Relay [5]. The relay connects to three electrical buses to directly collect their runtime currents and voltages. The RTAC device, which mediates between the control center and the relay, forwards commands from the control center to the relay and sends measurement data back from the relay to the control center.

The software configuration consists of *Communication Protocol Test Harness* from Triangle MicroWork inc.[], installed in the control workstation. This Windows application provides a GUI interface for controlling real field devices such as the RTAC and the relay. Several software simulated field devices used in SCADA systems also come with this software package. Experiments in this paper are made on both simulated and real field devices, because simulated devices provide complex functionality that is impractical to configure in a laboratory environment, while real field devices provide realistic responses.

B. Breakdown of Parser Code

Table I lists the code size of each component in DNP3 parser in terms of *binpac* script. The *parsing module* describes the code block to decode network streams into meaningful data fields. Complex user-defined data fields are usually recursively defined by basic data types. The column of *data type* represents the size of the codes to declare those data structures. The last part of the code declares each *Event Handler API* that can be used by policy modules.

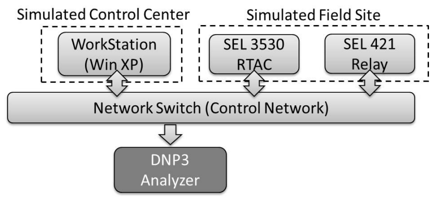


Fig. 7. Laboratory Scale SCADA System Testbed

TABLE I
BREAKDOWN OF DNP3 PARSER CODE

Parsing Module	Data Type	Event Handler APIs
272 lines	551 lines	773 lines

The size of the parsing module, which is always executed, is kept small. The event handler APIs contribute to a large amount of codes. The purpose is to provide a sufficient number of APIs to analyze the parsed information. Each event handler is executed only when the API is implemented to perform specific analysis. Consequently, a large number of declared APIs does not necessarily reduce the performance overhead.

C. Policy Module based on DNP3 Definitions

A policy module is developed according to DNP3 definitions. This policy module performs both intra- and inter-packet validation to guarantee that well-formed network packets are received.

TABLE II
EVENT HANDLERS DEFINED TO VALIDATE DNP3 PACKETS

Event Handler APIs	Implementation Complexity
dnp3_application_request_header	49 lines of scripts
dnp3_application_response_header	29 lines of scripts
dnp3_object_header	278 lines of scripts
other	88 lines of scripts

Table II presents event handlers implemented to validate monitored DNP3 network packets. When the DNP3 analyzer generates one of these events, the corresponding analysis is made in terms of Bros specified script. For example, in the *dnp3_application_request_header* event handler, function code field, which is one of its formal parameter, is validated whether it is in the valid range or not.

This policy module is developed based on synthetic legal traffic generated from *Communication Protocol Test Harness* to the RTAC machine. This *Protocol Test Harness* is configured to generate all its included operations. The policy module is refined until no alerts are generated under legal network packets.

This policy module is tested further by running it against synthetic illegal network traffic generated by injecting errors bit by bit into the legal network traffic collected during the module development. In order to emulate modifications made by the attacker, CRC and checksum values are recalculated after bit injection. As a result, the DNP3 analyzer can only detect syntax errors by performing intra- and inter- packet validation.

TABLE III
DETECTION OF ILL-FORMATTED NETWORK PACKETS BY DNP3 ANALYZER

DNP3 Operation	Injected Errors	Alerts from DNP3 Analyzer	Alerts from Wireshark
Read	1192	436	794
Write	4512	2874	3520
Execute	568	656	527
File Op	2528	2306	1877

Table III shows the detection results from both our DNP3 analyzer and Wireshark, which also parses DNP3. Twenty different types of requests supported by the test harness are issued. Both the requests and the corresponding responses are collected. All packets with error injected are replayed under the monitor of the DNP3 analyzer.

Due to space limitations, different DNP3 packets are grouped into four types according to nomenclature in Section V. Operations related to physical files (File Op) are considered separately as a file usually consists of different type of operations.

Alerts from the DNP3 analyzer include syntax errors and intra- and inter-packet errors. Alerts generated by Wireshark include all possible warnings that can be found, such as malformed packets, failed to decoding and other warnings.

As errors are injected bit by bit, the number of errors is directly related to the length of the request. For both parsers, the number of alerts is not equivalent to the number of detection, as several alerts may correspond to the same error. In addition to the syntax error and intra-packet validation, the DNP3 analyzer provides inter-packet validation. For example, if a field indicating the number of objects in the DNP3 response is corrupted, the DNP3 analyzer detects not only a syntax error but also an inconsistency in terms of number of objects between the DNP3 request and this response. For a single error, the DNP3 analyzer provides different aspects of information. As a result, we can have more alerts than the number of errors (detection of errors in execute operation in Table III)

D. Parsing Overhead of Single Packet

In SCADA systems, once an operation is performed, it is difficult to recover its effect. Quick detection is critical as it can instantly be remedied. As a result, we present overhead generated by the DNP3 parser over a single packet.

Figure 8 presents the DNP3 parser's execution time normalized to time overhead of Wireshark. We run Wireshark, Bro without the DNP3 parser, and Bro with the DNP3 parser against network packets of four different sizes. In general, the size of the DNP3 network packet is not as large as listed. The large network packet can also degrade field device performance: for example, the *protocol test harness* configures the largest size of DNP3 packet as 1024 bytes by default.

With different design purposes and principles, Bro runs more slowly than Wireshark, whose ultimate goal is parsing. Bro's efficiency in large scale computing environment has been discussed in [16]. The real overhead which is due to the DNP3 parser, including both parsing and execution of the mentioned protocol validation module, is less than twenty percent.

With packet size changing in this range, neither Wireshark nor Bro obviously result in different process overhead, as this size is actually smaller than the general TCP payload.

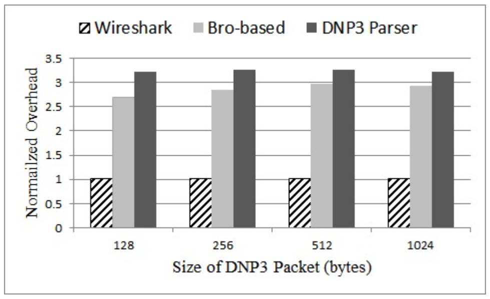


Fig. 8. Detection Overhead of DNP3 Parser on Single Packet

VI. Evaluation on Different Security Policies

In this section, we focus on the security analysis of simulated SCADA system environments. Several easy-to-implement intrusions based on simple socket APIs have been developed to change the state of the Rtac and the relay machine. The security polices proposed in Section V are implemented by Bros specified script language. The effectiveness and implementation complexities of policies are tested against the developed attacks as a proof-of-concept.

A. Implementation of Attacking Cases

In the simulated SCADA systems, we refer to an attack machine that injects errors into DNP3 network packets. The location of the injection is the selected payload; thus carried commands or measurements are compromised. The error-injected packets are reconstructed as well-formatted ones with TCP headers and IP headers and issued to the network.

We run our malicious codes in separate machines to better manage them and avoid unnecessary compromises. The codes themselves, however, simply require sock API, which can be found in many commonly used OSs and can therefore be integrated into the workstations used in SCADA systems. In the following sections, we present implementations of different sample security policies mentioned in Section V. Because implementations have been integrated, the DNP3 analyzer runs while the attack machine performs malicious actions. Two possible consequences are potential concerns in all our experiments: the detection capability in terms of false positives and false negatives and the chance that the DNP3 analyzer fails to analyze any network packets.

B. Signature-based Policy

In order to validate the signature-based policy, we predefined critical system states as signatures in terms of the relay machine's binary output values. The DNP3 analyzer is configured to analyze the network traffic and estimate the relays state. By comparing the estimated states to the signature at runtime, the analyzer generates alerts.

The workstation in the control center is properly configured to continuously issue commands to change the binary outputs of a simulated SEL 3515 Software Relay included in the *protocol test harness* [3], [2]. In DNP3, the operation to set/clear a binary output can be represented by a mere two bytes in the application layer, one indicating the index of a binary output and the other an set/clear command. A software relay is chosen instead of the physical one because few outputs are configured.

Within the assumed threat model, the control center can be compromised or completely controlled by a vengeful former employee. To simulate this attack scenario, the attack machine performs replay attacks based on the monitored network traffic (Figure 9). Errors are injected into bytes representing binary outputs. The attack machine randomly chooses the index of output, changes the set/clear byte and issues the request again to the network. With the CRC and checksum being properly modified, the malicious but well-formatted DNP3 requests successfully change binary outputs of the software relay into unexpected values.

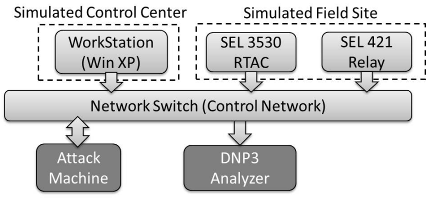


Fig. 9. SCADA Systems Testbed with Attack Machine

In the context of this experiment, we use a combination of the five binary outputs to represent states of the software relay. Three critical system states are predefined in terms of such combinations. In order to perform such detections, two event handler APIs *dnp3_object_header* and *dnp3_crob* (CROB is short for control relay output binary) are implemented. We further define a state table accessed by both *dnp3_object_header* and *dnp3_crob*. This table contains five binary values used to estimate the states of the software relay.

After the state table is initially assigned the relay states, the analyzer monitors every DNP3 network packets. It generates a*dnp3_object_header* event when a well-formed header of measurement or command objects is found in a DNP3 request. The implemented handler informs the DNP3 analyzer that a request is found, what control commands are issued, and other connection related information. In this context, DNP3 analyzer generates *dnp3_crob* events once it finds objects to configure binary outputs of the software relay. With the information collected from each event, the analyzer updates the state table accordingly. If the table values match the predefined critical states, alerts are generated.

Table 4 shows the complexity of the implementation, in terms of Bros specified scripts, required to perform the signature-based analysis. Both implementations to analyze *dnp3_object_header* and *dnp_crob* events require not more than 100 lines of Bro scripts. The others part in Table 4 indicates the amount of script for accessary procedures, such as definitions of the state tables and mechanisms of the log framework. With the small number of scripts, the DNP3 analyzer can efficiently narrow down its analysis to a specific aspect of the network, such as the relay control binary outputs in this example.

TABLE IV
Implementation Complexity of Signature-based Policy

Event Handler APIs	Implementation Complexity
dnp3_object_header	5 lines of scripts
dnp3_crob	78 lines of scripts
other	62 lines of scripts

To run the experiment, the attack machine replays the errors injected DNP3 requests to the software relay. The latency between each request is set as 1 second. The experiment is continuously carried out for approximately 10 hours, generating a total of 9344 requests.

In order to determine whether detection is correct, we intentionally issue a DNP3 read request right after each malicious request. The read request polls out all inputs/outputs values from the software relay, including binary outputs used as signatures in this experiment. By comparing the system states from read requests and alerts generated from the DNP3 Analyzer, actually both zero false positive and false negatives are found. This is a reasonable result as signature-based detection does not require much intelligence during detection. The success of detection largely depends on the choice of signatures. The drawback of this policy is that even with all critical system states being detected, some states of the software relay are still compromised, but not leading system to critical contingencies.

On the other hand, to validate the chance that the DNP3 analyzer misses any network packets, we plan to reduce the latency between each pair of malicious requests. However, we have found that 1 second is the lowest frequency that can be configured to the software relay. Lower frequency results in dropped requests or infinite loops. However, this situation is not found in the real hardware relay. In a later experiment directly involving the physical relay, we issue malicious network packets with much more intense frequency.

C. Anomaly-based Policy

In this experiment, the attack machine, integrated into the control center, sends DNP3 requests to the physical relay. We connect the DNP3 analyzer to a Matlab module performing state estimation in order to analyze the system state after suspicious operations (Figure 10). The Matlab module, available from the Mathwork community [4], performs weight least square state estimation [8] on IEEE 14-bus and 30-bus test systems. The Matlab modules are installed in a different Ubuntu machine and connected to the DNP3 analyzer through the Ethernet.

Bad measurement data detection mechanisms are not included in the state estimation module, which lacks a threshold value that is impractical to obtain in small-scale laboratory environment. Consequently, instead of experimenting with detection capability, we focus on the chance that the DNP3 analyzer fails to analyze network packets as a result of additional online computations.

In order to build the communication channel between the DNP3 analyzer and the state estimation module, we have implemented event handlers of *dnp3_response_header*, *state_estimation_recv* and *state_estimation_sent* events in the analyzer and a broccoli client program in the state estimation machine based on the Bro client communication library. The DNP3 analyzer generates *dnp3_response_header* when a well-formed DNP3 response is issued by the replay. We implement this *dnp3_response_header* event to send a message to the broccoli client to start state estimation in a Matlab engine. After finishing the computation, the broccoli client sends results back to *state_estimation_recv* event which are then logged.

As shown from Table V, the DNP3 analyzer does not perform much work, serving merely as a front-end measurement collector. The module's most complex component is the broccoli client, which mediates between the DNP3 analyzer and the back-end Matlab engine.

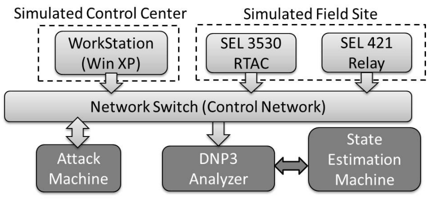


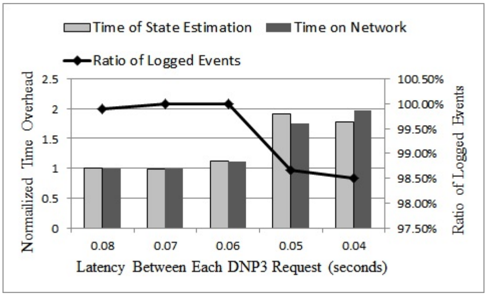
Fig. 10. SCADA System Testbed with State Estimation Module

After constructing the communication channel, we start the attack machine in the workstation to send DNP3 requests to the physical relay. As certain delay can be expected from the DNP3 analyzer, we insert latency between DNP3 requests. For each latency value, a total of 20000 requests have been run.

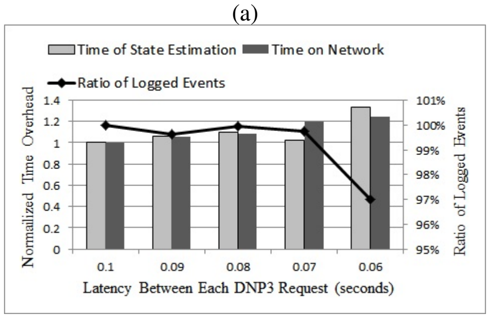
TABLE V
IMPLEMENTATION COMPLEXITY OF ANOMALY-BASED POLICY

Event Handler APIs	Implementation Complexity
dnp3_response_header	5 lines of scripts
state_estimation_recv	17 lines of scripts
state_estimation_sent	5 lines of scripts
other	128 lines of scripts
broccoli client	347 lines of scripts

Figure 11 (a,b) illustrates what is observed when state estimation is made on 14-bus and 30-bus test systems. For both cases, we start from latency values when all events can be monitored by the DNP3 analyzer. As the latency is decreased, the DNP3 analyzer misses more and more events. During our experiments, both the network communication and state estimation times for each request are also collected and normalized to the values collected when no events are missed.



(a)



(b)

Fig. 11. SCADA System Testbed with State Estimation Module

From Figure 11 (a), a pulse of event loss is found when latency is reduced to 0.05 seconds from 0.06 seconds. Coming with this pulse is the big increase of the state estimation time. This is probably due to the fact that when one computation is still in progress, it may be interrupted by another. Extra time is needed from the Matlab engine to handle such exceptions. The other factor is unpredictable network communication time, which relies on the network condition. As a result, the DNP3 analyzer is also found to lose one or two packets in rare situations even if the latency is increased to as much as 0.15 seconds. This phenomenon poses a noticeable threat if the DNP3 analyzer is connected to proprietary software. In order to spare more time on the state estimation, we can further define an additional security policy to raise alerts when a request is sent too frequently. However, for the time spent on the networks, an attacker can intentionally send some flood traffic to degrade the network performance and issue a malicious request. Unpredictable network conditions enable malicious packets to evade detection even if they are sent with long latency. To thoroughly eliminate this uncertainty, the DNP3 analyzer may be installed on the same machine as the proprietary system analysis software.

D. Specification-based Policy

1) Validation on Execute (Write) Operation: This experiment still follows the experimental setup presented in Figure 9. As rules defined to restrict *write* operations are similar to *execute* operations, the experiment focuses exclusively on the latter. The authorized sequence order in the defined security policy for performing an *execute* operation is 1) authentication of the operator, 2) a *read* operation, and 3) a *execute* operation. The attack machine uses localhost root SSH login to simulate the authentication actions. A *read* operation and an execution operation, consisting of SELECT and OPERATE DNP3 requests, are subsequently issued. While the execution operation is always performed, the authentication and the *read* operation are randomly ignored to simulate the attackers unauthorized actions.

TABLE VI
IMPLEMENTATION COMPLEXITY OF VALIDATING EXECUTE OPERATION

Event Handler APIs	Implementation Complexity
dnp3_request_header	31 lines of scripts
SSH::heuristic_successful_login	Already defined in Bro
state_estimation_sent	5 lines of scripts
other	56 lines of scripts

To perform such sequence validation, only the *dnp3_request_header* needs to be implemented (Table VI) . The DNP3 analyzer generates such events whenever a well-formed dnp3 request header is monitored. Another SSH event which detects successful SSH login is directly used.

TABLE VII
EXPERIMENTAL RESULT TO VALIDATE EXECUTE OPERATION

Latency	0.059224 seconds
Number of Error Injected	269831
Mismatch Found	269831

To run the experiment, the control workstation continues performing 360000 rounds of such series of operations for around 10 hours (Table VII). If either an authentication or a *read* operation is not issued, the control workstation would record an error occurrence. The DNP3 analyzer parses out the type of operation from the network packets and validates the sequence as defined in the policy. During this 10 hour period, zero false positives or negatives are found.

This experiment is problematic in that the latency between each round of operations is kept as 0.059224 seconds on average, because authentication events are monitored on a different loopback Ethernet interface from the one monitoring the read and execute operations. If the latency is too small, we find that events from two Ethernet interfaces may not come in order as they are generated. Since Bro does not provide low-level synchronization, the policy module should be further refined in order to handle such synchronization issues.

After the sequence validation, the untrusted field devices, i.e. the RTAC machine in the testbed, can corrupt the integrity of each operation. Validating the integrity of each operation follows similar methods as does validating read measurement data. This will be presented in the next section.

E. Validation on Read Operation

To reveal possible threats that can occur during read operation, we perform a man-in-the-middle attack by compromising the RTAC machine. To avoid unnecessary compromise, a separate *Evil RTAC* machine replaces the original ones. This *Evil RTAC* machine forwards DNP3 requests to the relay machine. While a response is received from the Relay, this Evil RTAC randomly injects errors into the measurements, i.e. analog inputs values in this experiment. and sends it back to the control center.

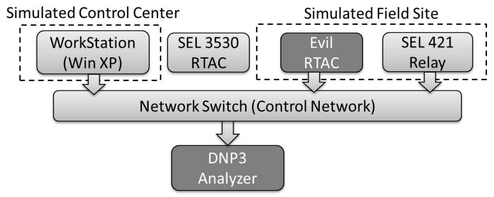


Fig. 12. Man-in-the-Middle Attack in SCADA System Testbed

The DNP3 analyzer monitors all traffics among the control workstation, *Evil RTAC* and the physical relay machine. In our simulated SCADA system testbed, all these three devices are physically connected to the same switch. As a result, the switch can be configured to mirror traffic to DNP3 analyzer. In practice, a field bus is used to connect the RTAC to a single or multiple relay machines. The DNP3 analyzer can be attached to the fieldbus to perform similar monitoring.

TABLE VIII
EXPERIMENTAL RESULT TO VALIDATE READ OPERATION

Latency	0.059224 seconds
Number of Error Injected	269831
Mismatch Found	269831

Table VIII shows that only *dnp3_analog_input32_woFlag* event handler needs to be implemented to perform such validation. The DNP3 analyzer generates this event when analog inputs are carried in the response without attaching additional flag values. Corresponding APIs may be implemented for other value types in the relay machine. The DNP3 analyzer distinguishes responses from the relay to the *Evil RTAC* from the one further forwarded to the control workstation by device IP addresses. After parsing values from both responses, comparisons made on each analog input value can indicate whether the reading measurements are compromised or not.

To run the experiment, the control workstation issues DNP3 read requests continuously for about 11 hours without any latency inserted between each request. Within the configuration of the workstation, the average latency measured between each request is 0.004505 seconds. In other words, around 220 requests are sent per second. This frequency is sufficient for most SCADA system requirements. Each read request actually polls all the measurements from the Relay machine in order to increase the packet size. A total of 7742657 errors are injected into bytes representing analog input values; the DNP3 analyzer generates zero false positives and false negatives during this 11 hour period.

VII. Related Work

A. Intrusion Detection in SCADA Systems

Anomaly-based detection technique is based on building a baseline behavior in terms of statistical metrics during a systems error-free operation period. An intrusion is detected whenever a run-time audit trace deviates from the established baseline behavior. This technique is first retrofitted to be used in SCADA systems. In [28], [27], [11], network-related parameters, such as IP address or port number, are adopted to construct normal network traffic patterns. Suspicious events, such as scanning in SCADA network, could also be detected. However, anomaly-based detection techniques fail to detect malicious network packets following the network traffic pattern, such as the attack cases proposed in this paper.

Specification-based techniques detect intrusions by recording deviations from logical system behavior. Such authorized behavior is defined by logical specifications, instead of an attack signature or a statistical pattern. Work presented in [33] applies this technique to advanced metering infrastructures. This work focuses on the design of security policies and how to formally verify them. Our work focuses on a framework that can be used in most SCADA systems.

B. Semantics Analysis in SCADA Systems

Research work from [19], [9] relies on the semantics collected from real critical infrastructure to estimate the attempt of control commands that can put system in danger.

[14], [13] propose performing analysis on semantics from each command by analyzing possible effects on remote field sites. This approach is similar to our retuned signature-based policy. As mentioned, detections simply based on certain critical system states are not sufficient. Our framework provides much better extensibility.

VIII. Conclusions and Future Work

In this paper, we adapt Bro to build a DNP3 analyzer as an example to introduce a specification intrusion detection framework into SCADA systems. With the help of a built-in DNP3 parser, the analyzer is able to generate real-time events observed in a SCADA system network. A large number of event handler APIs are declared to collect a sufficient amount of semantics to investigate the network.

Based on the semantics, we re-implement signature- and anomaly-based detection policies. Furthermore, a separate policy is defined to provide different security requirements for each operation type. All proposed security policies are implemented and integrated into the DNP3 analyzer. With experiments including self-built malicious software, the performance and detection capabilities of the DNP3 analyzer are presented in similar network environments similar to those of SCADA systems. More importantly, the implementations of security policies are kept at a simple level, proving good extensibility.

While the DNP3 analyzer is proved to be effective and efficient in the simulated laboratory environment, future works with real network traffic and environments are required. Currently, the search for an industrial collaborator is in progress.

Acknowledgment

This material is based upon work supported by the Department of Energy under Award Number DE-OE0000097. This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

References

[1] Bro intrusion detection system. http://bro-ids.org/.

[2] Communication Protocol Test Harness. http://www.trianglemicroworks.com/documents/Protocol_Test_Harness_Fact_Sheet.pdf.

[3] Legacy SEL-351S Protection and Breaker Control Relay. https://www.selinc.com/LegacySEL-351S/.

[4] Power system state estimation using wls. http://www.mathworks.com/matlabcentral/fileexchange/6101-power-system-state-estimation.

[5] SEL-421 Protection, Automation, and Control System. https://www.selinc.com/SEL-3530/.

[6] SEL3530Real-Time Automation Controller (RTAC). https://www.selinc.com/SEL-3530/.

[7] Vulnerability analysis of energy delivery control systems. Technical report, Idaho National Laboratory, Sep 2011.

[8] A. Abur and A.G. Exposito. *Power system state estimation: theory and implementation*, volume 24. CRC, 2004.

[9] J. Bigham, D. Gamez, and N. Lu. Safeguarding scada systems with anomaly detection. *Computer Network Security*, pages 171–182, 2003.

[10] R.B. Bobba, K.M. Rogers, Q. Wang, H. Khurana, K. Nahrstedt, and T.J. Overbye. Detecting false data injection attacks on dc state estimation. In *Preprints of the First Workshop on Secure Control Systems, CPSWEEK 2010*, 2010.

[11] L. Briesemeister, S. Cheung, U. Lindqvist, and A. Valdes. Detection, correlation, and visualization of attacks against critical infrastructure systems. In *Privacy Security and Trust (PST), 2010 Eighth Annual International Conference on*, pages 15–22. IEEE, 2010.

[12] E. Byres, D. Leversage, and N. Kube. Security incidents and trends in scada and process industries. *The Industrial Ethernet Book*, 39(2):12–20, 2007.

[13] A. Carcano, I. Fovino, M. Masera, and A. Trombetta. State-based network intrusion detection systems for scada protocols: a proof of concept. *Critical Information Infrastructures Security*, pages 138–150, 2010.

[14] A. Carcano, I.N. Fovino, and M. Masera. Modbus/dnp3 state-based filtering system. In *Industrial Electronics (ISIE), 2010 IEEE International Symposium on*, pages 231–236. IEEE, 2010.

[15] K. Curtis. A dnp3 protocol primer. *DNP Users Group,(www. dnp. org/files/dnp3_primer. pdf)*, 2000.

[16] H. Dreger, A. Feldmann, V. Paxson, and R. Sommer. Operational experiences with high-volume network intrusion detection. In *Proceedings of the 11th ACM conference on Computer and communications security*, pages 2–11. ACM, 2004.

[17] M. Fabro and T. Nelson. Control systems cyber security: Defense-in-depth strategies. Technical report, Technical report, Idaho National Laboratory (INL), 2007. 2.2, 2007.

[18] Nicolas Falliere, Liam O Murchu, , and Eric Chien. W32. stuxnet dossier. *White paper, Symantec Corp., Security Response*, 2011.

[19] Dina Hadziosmanovic, Damiano Bolzoni, Pieter Hartel, and Sandro Etalle. Melissa: Towards automated detection of undesirable user actions in critical infrastructures. In *European Conference on Computer Network Defense, EC2ND 2011*, pages 41–48. IEEE Computer Society, 2011.

[20] O. Kosut, L. Jia, R.J. Thomas, and L. Tong. Malicious data attacks on smart grid state estimation: Attack strategies and countermeasures. In *Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference on*, pages 220–225. IEEE, 2010.

[21] Y. Liu, P. Ning, and M.K. Reiter. False data injection attacks against state estimation in electric power grids. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 21–32. ACM, 2009.

[22] M. Majdalawieh, F. Parisi-Presicce, and D. Wijesekera. Dnpsec: Distributed network protocol version 3 (dnp3) security framework. *Advances in Computer, Information, and Systems Sciences, and Engineering*, pages 227–234, 2006.

[23] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Computer Networks*, 31(23-24):2435 – 2463, 1999. "".

[24] 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, 2002.

[25] Keith Stouffer, Joe Falco, and Karen Kent. Guide to supervisory control and data acquisition (scada) and industrial control systems security. 2006.

[26] Robert J. Turk. *Cyber incidents involving control systems*. Idaho National Engineering and Environmental Laboratory, 2005.

[27] Alfonso Valdes and Steven Cheung. Communication pattern anomaly detection in process control systems. In *Technologies for Homeland Security, 2009. HST'09. IEEE Conference on*, pages 22–29. IEEE, 2009.

[28] Alfonso Valdes and Steven Cheung. Intrusion monitoring in process control systems. In *System Sciences, 2009. HICSS'09. 42nd Hawaii International Conference on*, pages 1–7. IEEE, 2009.