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

Hui Lin[1], Adam Slagell[2], Catello Di Martino[1], Zbigniew Kalbarczyk[1], Ravishankar K. Iyer[1]
[1]Coordinated Science Laboratory, University of Illinois at Urbana-Champaign,
1308 W. Main Street, Urbana, IL, 61801
[2]National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign,
1205 W. Clark Street, Urbana, IL, 61801
[1]{hlin33, dimart, kalbarcz, rkiyer}@illinois.edu, [2]slagell@illinois.edu

ABSTRACT

When SCADA systems are exposed to public networks, attackers can more easily penetrate the control systems that operate electrical power grids, water plants, and other critical infrastructures. To detect such attacks, SCADA systems require an intrusion detection technique that can understand the information carried by their usually proprietary network protocols.

To achieve that goal, we propose to attach to SCADA systems a specification-based intrusion detection framework based on Bro [7][8], a runtime network traffic analyzer. We have built a parser in Bro to support DNP3, a network protocol widely used in SCADA systems that operate electrical power grids. This built-in parser provides a clear view of all network events related to SCADA systems. Consequently, security policies to analyze SCADA-specific semantics related to the network events can be accurately defined. As a proof of concept, we specify a protocol validation policy to verify that the semantics of the data extracted from network packets conform to protocol definitions. We performed an experimental evaluation to study the processing capabilities of the proposed intrusion detection framework.

Categories and Subject Descriptors

K.6.5 [**Security and Protection**]

General Terms

Security

Keywords

SCADA, DNP3, Bro, specification-based intrusion detection system

1. INTRODUCTION

SCADA (Supervisory Control And Data Acquisition) systems monitor and control geographically distributed assets found in power grids, water plants, and other critical infrastructures. Modern SCADA systems are increasingly adopting Internet technology to boost control efficiency. Exposing such control systems to public networks increases the risk of attacks and failures inherited from the commodity network infrastructure.

What makes things even worse is that many companies operating critical infrastructures lack sufficient protections against failures caused by accidental events and malicious attacks. Consequently, industrial control operations are subject to serious cyber threats, and not just in theory. For example, in 2011, an attacker penetrated the control system of a water plant in Texas; in a similar 2012 incident, an intruder broke into a company operating gas pipelines.

The major challenge of applying traditional intrusion detection systems (IDSes) is that they usually lack sufficient capabilities to investigate network traffic based on unique proprietary protocols found in SCADA systems. This drawback prevents in-depth analysis of network activities, making traditional IDSes blind to attacks specific to SCADA systems.

In this paper, we propose a specification-based intrusion detection framework to provide high visibility of the semantics of the data carried by the proprietary network protocols. Specifically, we adapted Bro [7][8], a real-time network traffic analyzer, to integrate parsers of proprietary network protocols, such as DNP3, used in electrical power grids [3]. The built-in parsers generate network events related to SCADA systems, which are further analyzed to detect violations of defined security policies using the proposed intrusion detection framework.

Furthermore, we specify a protocol validation policy to maintain appropriate communication patterns defined by the DNP3 protocol. Abnormal communication patterns, which can be caused by malformed or replayed network packets, may indicate device failures, system misconfigurations, denial-of-service attacks, or malicious operations that put control environments into unstable states. The main purpose of proposing this policy is to demonstrate that the proposed intrusion detection framework is able to analyze the SCADA-related semantics from DNP3 network traffic. Other scenario-specific policies can be similarly specified and applied in various SCADA systems. The DNP3 parser and the proposed policy that we have built will be included in Bro's source code repository [8].

2. Related Work

Traditional signature-based intrusion detection techniques are not widely used in control environments, because little analysis of real attacks is available to public. Instead, anomaly-based intrusion detection techniques were initially used in the area. The work in [5] uses destination host addresses, port numbers, and other attributes of network packet headers to detect abnormal network traffic in SCADA systems. However, that technique is not effective in detecting malicious control operations hidden in the network payload.

The work in [2] detects intrusions based on violations of defined models characterizing the knowledge specific to the Modbus protocol. Modbus is another proprietary protocol used in SCADA systems [9]. Compared to Modbus, many other proprietary protocols, such as DNP3, are much more complex and contain more diverse semantics. Work presented in [1] applies a specification-based technique to the advanced metering infrastructure (AMI), which is a very different wireless communication environment. Both [1] and [2] emphasize the design of system models or specifications and their formal verification. Although our work also proposes a protocol validation policy for DNP3, we focus on the design of an applicable framework that can be used in real SCADA systems to provide various runtime semantic analyses.

3. DNP3 ANALYZER

In this section, we present three main components of the *DNP3 analyzer*, the proposed intrusion detection framework based on Bro [7][8].

Bro is a real-time network traffic analyzer widely used in forensic analysis, intrusion detection, and other network-related analysis. The modifications that we made to adapt Bro into SCADA systems are highlighted in Figure 1. We built a new parser of the DNP3 protocol to generate SCADA system-specific events. The semantics related to each event were delivered into the corresponding event handler. To analyze the semantics, we implemented the protocol validation policy by defining event handlers in terms of Bro scripts. The policy script interpreter executed the scripts to produce analysis results, such as alerts on abnormal network activities.

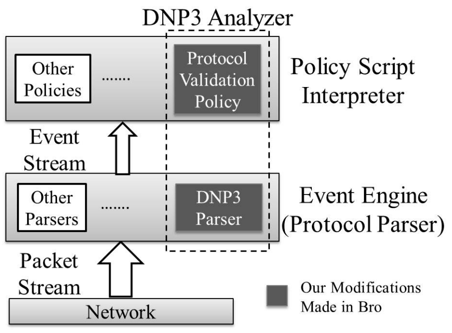
DNP3 Analyzer
Other Policies
....
Protocol Validation Policy
Policy Script Interpreter
Event Stream
Other Parsers
....
DNP3 Parser
Event Engine (Protocol Parser)
Packet Stream
Network
Our Modifications Made in Bro

Figure 1: DNP3 Analyzer Based on Bro

3.1 DNP3 Parser

The main responsibility of the network packet parser is to decode byte streams into meaningful data fields according to the protocol definition. The main body of Bro is written in C++. The DNP3 parser, however, exploits a compiler-assisted tool named *binpac* to shorten the development period and to ensure logical correctness [6].

To design the new parser, we represented the syntax of the DNP3 protocol by the *binpac* scripts, which are specifically designed to represent the hierarchical structure of a network protocol. With the help of the *binpac* compiler, the *binpac* scripts were automatically translated into C++ and integrated into Bro.

3.2 Event Handlers

Event handlers are used to analyze network events generated from parsing of each DNP3 network packet. The semantic information related to each event is extracted during parsing. For example, a *dnp3_crob* (Control Relay Output Block) event is generated by the DNP3 parser if an operation to control relay outputs is found within a DNP3 request. The parameters associated with this operation, such as the type and the duration of the operation, are extracted from the packet and delivered to the corresponding event handler.

A declaration of an event handler, including its name and arguments, provides an interface between the DNP3 parser and the policy script interpreter. During the parsing at runtime, the value of each argument is updated by the semantic information related to this event. We declared and associated an event handler with each type of data field defined in the DNP3 protocol; thus the DNP3 analyzer can cover all semantic information from any type of DNP3 network packet. Although the declarations of event handlers are fixed, their definitions are left to be implemented in terms of Bro scripts written by security experts. In specific operational contexts such as operations in power grids, system policies can be dynamically adjusted by including definitions of different event handlers.

3.3 Protocol Validation Policy

Other than defining the hierarchical structure of a network packet, the DNP3 protocol introduces additional requirements regarding network traffic. Specifically, dependencies between data fields within a single network packet are defined, and certain communication patterns between different network packets have to be maintained. The purpose of this policy is to use intra- and inter-packet validation to ensure observance of such requirements.

3.3.1 Intra-Packet Validation

The intra-packet validation is used to ensure dependencies between different data fields within a single network packet. Similar to teardrop attacks, malformed network packets can be used to directly perform denial-of-service attacks. During our experiment, such an attack occurred; malformed DNP3 network packets crashed Wireshark [10], an open source network traffic monitor.

A DNP3 network packet consists of different data fields, such as the object type and the function code. The structure of some data fields depends on the value of others. For example, the “length” field in the link layer header defines the length of the following payload field. As a result, we should verify that the value of the “length” field is consistent with the real payload length.

During the validation process, the value ranges for certain data fields are also analyzed, because out-of-bound values can be used to detect attacks. For example, the DNP3 protocol uses an 8-bit integer to represent the function code, and 37 out of 256 possible combinations are defined. However, in a real control system, only a subset of the 37 values are supported. So a DNP3 request with an abnormal function code may indicate a reconnaissance scan from an adversary.

3.3.2 Inter-Packet Validation

In addition to defining rules for data fields within a network packet, DNP3 defines communication patterns between different packets. For example, an “OPERATE” packet is almost always issued right after a “SELECT” packet to control remote field devices chosen by the previous “SELECT” packet.

The unmatched requests and responses are often the result of denial-of-service attacks or replay attacks, such that an adversary can flood a communication channel with previously transmitted network packets in an attempt to unexpectedly repeat certain operations. The DNP3 analyzer can maintain states from the parsed network packets. Based on the states, the incoming packets are further correlated and analyzed to guarantee appropriate communication patterns.

4. EXPERIMENTAL EVALUATION

The DNP3 analyzer, which includes the DNP3 parser and the sample protocol validation policy, is evaluated in this section.

4.1 Evaluation of the DNP3 Parser

First, we focus on robustness evaluation of the DNP3 parser. An unexpected hanging of a parser would prevent it from analyzing DNP3 packets. As a result, the DNP3 analyzer would fail to detect potential attacks.

The DNP3 parser is constructed by the *binpac* scripts, which express the structure of a network protocol following a certain BNF grammar [6].

Table 1: A Part of the DNP3 Parser in the *binpac* Scripts

```
type Dnp3_Request = record {
  app_header : Dnp3_App_Req_Header ;
  data : case ( app_header.function_code ) of {
     0x01 -> read_requests :  Read_Req_Object [ ] ;
     0x02 -> write_requests : Write_Req_Object [ ] ;
         ......
     };
};
type Dnp3_App_Req_Header = record {
     application_control : uint8;
     function_code : uint8;
} &length = 2 ;
```

Table 1 shows a part of the *binpac* scripts that represent a DNP3 request. In *binpac*, a *record* data structure, which is a user-defined composite type, describes a production rule in a BNF grammar. For example, the "Dnp3_App_Req_Header" record can be regarded as the following production rule:

Dnp3_App_Req_Header ::= *application_control function_code*

After the "app_header" of the type "Dnp3_App_Req_Header" has been defined, the "data" part can be defined by different new *record* types, whose internal structure is varied according to the function code field in the "app_header" (implemented by a "case" statement). Similarly, whenever defining a new field inside the "data" part, we explicitly made a new *record* type for this field instead of using predefined ones (even if this new field has the same structure as the predefined ones). As a result, the DNP3 parser avoids using recursive production rules, such as production rules with the form of $A ::= Ax$ or $A ::= Bx$; $B ::= A$.

We evaluated the DNP3 parser using a sample traffic trace collected from a real electrical power grid located in Ohio. We then evaluated it further using malformed synthetic network traffic with the protocol validation policy. The latter experiment is described in the next section.

4.2 Evaluation of Protocol Validation Policy

In this paper, the protocol validation policy is specifically defined based on the context of SCADA systems operating electrical power grids. Its implementation includes the definitions of three event handlers: *dnp3_app_request_header*, *dnp3_app_response_header*, and *dnp3_object_header*. These event handlers extract values of the function code, the object type, and other semantic information from the DNP3 request/response headers and object headers.

For example, an object with the group number 12 and variation number 1 describes a *CROB (Control Relay Output Block)* object. This type of object can only be initiated by requests with function codes 3, 4, 5, and 6. Consequently, we included the following Bro scripts in the policy to validate this rule.

```
if ( ( Obj_Type == 0x0c01 )  &&
        ( ( FunCode < 0x03 )  ||  ( FunCode > 0x06 ) ) )
    ALERT ....;
```

Other rules defined by the DNP3 protocols were verified through similar scripts. The implementation details will be included in Bro's source code repository [8].

We evaluated the implementation of the policy in a simulated SCADA testbed (Figure 2). A Windows XP workstation simulated a control center that collected measurement data from a field site and issued operations to it. The field site was simulated by a data aggregator and a relay. The data aggregator worked as a mediator that aggregated measurement data from the relay and forwarded an operation from the control center to this relay. The relay monitored the status of an electrical transmission line. The monitor machine was a separate commodity workstation in which the proposed DNP3 analyzer ran independently without affecting the operations of the control center and the field site. All the components were connected to a network switch. The switch was configured such that all network traffic was "mirrored" to the monitor machine.

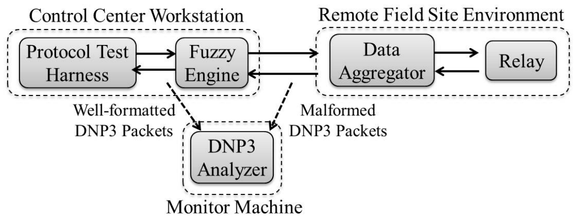

Figure 2: Simulated SCADA Testbed

We used Protocol Test Harness [11], the software running in the control center, to generate DNP3 network packets of different structures. A "Fuzzy Engine" is a self-developed program based on the TCP/IP socket. In each round of communication, the "Fuzzy Engine" replaced each byte of the generated packet with a random value. As a result, the control center issued both well-formatted and malformed packets to the data aggregator (Figure 2). Corresponding error detection codes (CRC values) were recalculated to simulate modifications made by an attacker.

4.2.1 Robustness Evaluation

For comparison, both Wireshark [10] and our DNP3 analyzer were used to monitor the testbed at runtime. Notably, the two tools handled malformed network packets differently. In one of our experiments, Wireshark looped for more than three hours when processing a malformed packet that is shown in Figure 3. The 5th byte of the packet represented the *qualifier field* that defines the hierarchical structure of the remaining part of the packet. After it was replaced with the value 0x09, Wireshark hung for over three hours. Although it is not clear what exactly caused the loop, we suspect that the injected errors resulted in the misuse of a loop statement or a recursive procedure.

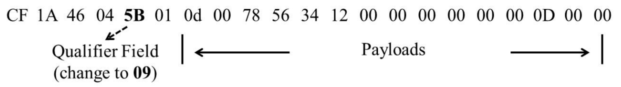
CF 1A 46 04 5B 01 0d 00 78 56 34 12 00 00 00 00 00 00 0D 00 00
Qualifier Field
(change to 09)
Payloads

Figure 3: The DNP3 Network Packet that Crashes Wireshark

Our proposed DNP3 analyzer did not introduce such unreliable behavior during any of our experiments. The DNP3 parser avoids using recursive production rules in its implementation. The protocol validation policy is implemented by less than 400 lines of Bro scripts. Consequently, we can easily verify that the policy scripts avoid loop statements and recursive function calls.

4.3 Performance Evaluation

As the DNP3 analyzer is used to analyze industry control environments passively, it must process network packets in real-time to provide useful detection results.

In this section, we evaluate the throughput of the DNP3 analyzer in terms of the number of packets processed per second. We used the experimental setup shown in Figure 2 to generate a 1 GB packet trace. The packet trace contained both well-formatted and malformed DNP3 network packets along with the TCP packets needed to open and close communication sessions. The whole trace included a total of 3,789,120 DNP3 packets.

The DNP3 analyzer processed the packet trace off-line on the monitor machine. The purpose of the off-line analysis is to evaluate the ultimate processing capabilities of the DNP3 analyzer. The analysis results can suggest how the proposed DNP3 analyzer might fit into real SCADA systems.

The monitor machine was a VMware virtual machine with a single logical processor with two 3.07GHz cores and a 1GB RAM. During the processing, we ran the monitor machine exclusively without starting other virtual machines in the same host to avoid possible interferences. We performed 10 experimental runs to measure the average execution time.

Table 2 presents the throughputs of the DNP3 parser (first row) and the DNP3 analyzer with the protocol validation policy (second row). With the policy loaded, the DNP3 analyzer processed approximately 30% less network traffic every second. The reason is that the protocol validation policy performed intense analysis on almost all fields of each DNP3 network packet and generated a large number of alerts from malformed packets. Even under those circumstances, more than 9000 DNP3 network packets were processed every second. In an industrial control environment such as the power grid, legacy devices usually issue one or two DNP3 network packets every second [4]. Based on those figures, we anticipate that the proposed DNP3 analyzer can monitor a field site consisting of 4500 to 9000 devices. When more DNP3 analyzers are distributed into different host machines to form a monitor cluster, a larger-scale control environment can be monitored. Furthermore, it is possible to design an intrusion prevention system based on the proposed DNP3 analyzer to stop malicious operations.

Table 2: Throughput for the DNP3 Analyzer

Evaluation Target	Throughput (packets/second)
DNP3 Parser	13950
DNP3 Parser & Policy	9427

5. CONCLUSIONS

In this paper, we propose a DNP3 analyzer that is an intrusion detection framework based on Bro. With the help of the DNP3 parser, the analyzer is able to observe all DNP3 events related to SCADA systems. A sufficient number of event handlers are declared and associated with data fields in network packets to cover all semantic information carried by DNP3 network traffic.

Based on the extracted semantic information, we can accurately design security policies to perform analysis. As a proof of concept, we proposed a protocol validation policy that ensures that network traffic follows predefined communication patterns. The proposed DNP3 analyzer was evaluated in scenarios involving both well-formatted and malformed network packets, the latter of which triggered intense analysis and a large amount of alerts. Based on that "worst-case" experiment, we believe that the proposed DNP3 analyzer holds promise to work in real SCADA systems.

In future work, we plan to apply the DNP3 analyzer to detect well-formatted control operations from malicious users. In order to reveal the purpose of the suspicious network traffic, we must carefully select and correlate semantics related to different control operations as well as host activities in the control center.

6. ACKNOWLEDGMENTS

This material is based upon work supported in part by the Department of Energy under Award Number DE-OE0000097, by the National Science Foundation under Grant No. OCI-1032889, by Infosys Limited, and by the Boeing Company. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or other sponsors.

7. REFERENCES

[1] Berthier, R. and Sanders, W. H. 2011. Specification-based intrusion detection for advanced metering infrastructure. In *Proceedings of 2011 IEEE 17th Pacific Rim International Symposium on Dependable Computing* (Pasadena, CA, USA, Dec. 12 - 14, 2011), 184-193.

[2] Cheung, S., Dutertre, B., Fong, M., Lindqvist, U., Skinner, K., and Valdes, A. 2007. Using model-based intrusion detection for SCADA networks. In *Proceedings of the SCADA Security Scientific Symposium 2007* (Miami Beach, FL, USA, Jan. 24 - 25, 2007), 127-134.

[3] Curtis, K. 2000. A DNP3 protocol primer. Technical report. DNP User's Group.

[4] Heine, E., Khurana, H., and Yardley, T. 2011. Exploring convergence for SCADA networks. In *Proceedings of 2011 IEEE PES Innovative Smart Grid Technologies* (Hilton Anaheim, CA, USA, Jan. 17 - 19, 2011), 1-8.

[5] Linda, O., Vollmer, T., and Manic, M. 2009. Neural network based intrusion detection system for critical infrastructures. In *Proceedings of International Joint Conference on Neural Networks, 2009* (Atlanta, GA, USA, June 14 - 19, 2009), 1827-1834. IJCNN 2009.

[6] Pang, R., Paxson, V., Sommer, R., and Peterson, L. 2006. Binpac: A yacc for writing application protocol parsers. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement* (New York, NY, USA, Oct 25 - 27, 2006), 289-300. IMC '06.

[7] Paxson, V. 1999. Bro: A system for detecting network intruders in real-time. *Computer Networks*, 31, 23 (Dec. 1999), 2435-2463.

[8] The Bro Project. 2012. Bro Network Security Monitor. http://bro-ids.org.

[9] The Modbus Organization. 2006. Modbus messaging on TCP/IP implementation guide v1.0b 2006. http://modbus.org.

[10] The Wireshark Foundation. 2012. Wireshark. http://wireshark.org/.

[11] Triangle MicroWorks, Inc. 2012. Communication Protocol Test Harness. http://trianglemicroworks.com.