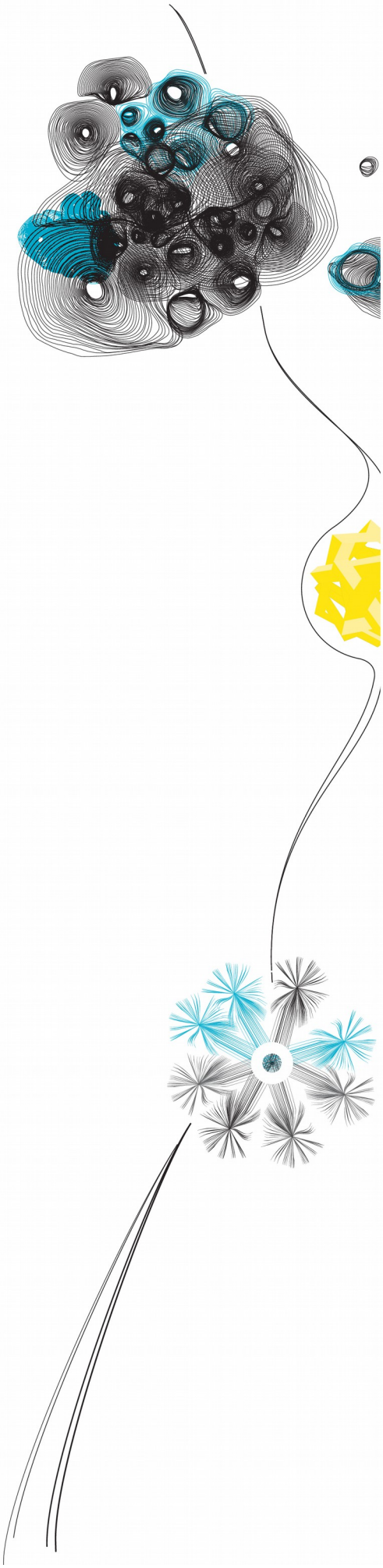


Master Thesis:

“Automatic Deployment of Specification-based Intrusion Detection in the BACnet Protocol”

Herson Tobías Esquivel Vargas

Graduation committee:
Dr. Andreas Peter
Dr. Anna Sperotto

Faculty of Electrical Engineering,
Mathematics & Computer Science

UNIVERSITY OF TWENTE.

September, 2016

Automatic Deployment of Specification-based Intrusion Detection in the BACnet Protocol

Herson Tobías Esquivel Vargas
Faculty of Electrical Engineering, Mathematics and Computer Science
University of Twente
The Netherlands

Abstract

Specification-based intrusion detection has high detection rates and low *false positives* rates. Its main drawback is that the creation of specifications (rules) often require human intervention. We present a data-mining approach that reads documents and automatically extracts rules from them. Specifically, we work in the field of Building Automation Systems (BAS) using the BACnet protocol (ISO 16484-5). The input documents are provided by manufacturers of BACnet devices. These documents state the capabilities of every device, therefore the extracted rules represent the expected behavior of the devices. In our experiments, the proposed algorithm creates rules with a 94.5% of concordance with the documents, on average. We tested our automatically generated rules in a real BACnet network. Non-standard undocumented capabilities were detected for three kinds of devices during the evaluation period.

1 Introduction

Processes automation has been one of the main applications in the ubiquitous computing field, specially for homes, buildings and industries [8, 14, 27]. The implementation of a Building Automation System (BAS) is an important step in the pursue of achieving those smart environments. Different solutions have emerged in the BAS context: LonTalk (ISO 14908), BACnet (ISO 16484-5) and KNX (ISO 14543-3) are competing protocols whose market share is difficult to assess [22]. Independently of the chosen standard, the growing trend in building automation is a palpable fact [17]. Since BACnet is vendor-neutral yet implemented by more than 800 manufacturers [13], this paper is focused exclusively on this protocol.

Although initially isolated, BAS devices became interconnected in order to ease management and provide new functionality. With the advent of the Internet of Things (IoT) new functional requirements pushed towards BAS networks being connected to the Internet. Internet exposed networks and devices allow efficient access for system administrators without physical location constraints. Unfortunately, remote Internet access also enables attackers to access and possibly cause damage to BAS from remote locations. The close interaction between

BAS actuators (elevators, alarms, lights, etc.) and people living or working in smart buildings, makes cyber-attacks a life threatening menace.

Different reports state conditions that increase the risk of attacks in BAS: the firmware is rarely updated in BAS devices [13], shared physical infrastructure with Internet connected LANs [9], easy access to exposed devices via IoT search engines like *Shodan*[1] [28] and the absence of authentication, authorization and encryption in BACnet [10].

Defining a threat model is the first step in the challenging task of securing a system. Back in 1995, BACnet's threat model dealt mostly with *insiders*. One of the BACnet creators wrote about "a disgruntled system operator who would do whatever damage he/she intended by means of an operator workstation" [19]. The massive connectivity we have nowadays has drastically changed the original threat model. The addition of *outsiders* as potential threats makes it a much more complex scenario.

Prevention of attacks is the way in which BACnet deals with security threats. The so-called *BACnet Security Architecture* has been integrated into the protocol with the aim to ensure confidentiality, integrity and authenticity of BACnet messages. If correctly implemented, those security features can prevent most of the attacks targeting BACnet [9]. However, BACnet compliant devices must adhere only to a small subset of mandatory clauses of the standard. Worrisomely, the *BACnet Security Architecture* is *not* mandatory and most devices do not implement such features [13, 28]. Detection of attacks becomes then one essential approach to secure BAS.

1.1 Intrusion Detection

Intrusion detection is "the process of monitoring the events occurring in a computer system or network and analyzing them for signs of possible incidents, which are violations or imminent threats of violation of computer security policies, acceptable use policies, or standard security practices" [23]. Literature identifies three types of *Intrusion Detection Systems* (IDSs): misuse detection, anomaly detection and specification-based detection [25]. Misuse detection, also known as signature-based detection, is done by comparing audit trails (e.g. network traffic) against a database of patterns that univocally identify attacks. Anomaly detection first observes what normal behavior is in a particular system and then scrutinizes audit trails looking for deviations from the norm in that particular setting. Finally, specification-based detection compares audit trails against rules that describe benign behavior (regardless of previous observations). Deviations from those rules are flagged as potential malicious activities. The three types of IDSs have advantages and disadvantages that must be carefully evaluated in the context where the IDS is going to be deployed.

Specification-based and anomaly-based intrusion detection share the capability to detect previously unknown attacks; however, specification-based detection considerably reduces the amount of false alarms triggered in anomaly-based detection [24]. On the other hand, rule generation is a costly process, which is considered the main drawback of specification-based intrusion detection [2].

Specification-based intrusion detection has been applied to a small set of computer network protocols including TCP, IP, AODV, among others [24, 20].

[1] https://shodan.io/

The rules for those protocols are extracted from openly available documents such as *Request For Comments* (RFC). Even though RFCs are mostly plain text descriptions in natural language (English), it is possible to extract formal and unambiguous rules from them. BACnet networks could be protected using specification-based intrusion detection using the same approach. Moreover, BACnet devices come with specific documentation about their capabilities. These documents are called "Protocol Implementation Conformance Statements" (PICS). Taking advantage of the availability of PICS, much more fine-grained rules can be generated. Our paper consists in *automated rule generation* from PICS to deploy specification-based intrusion detection in BACnet networks.

1.2 Contribution

Medium to large BACnet networks can easily contain hundreds of devices. Furthermore, BACnet networks are specially prone to change; devices are continuously added, replaced and removed. The fact that every device model has its own PICS makes it unfeasible to manually extract a complete set of rules.

PICS interpretation is hard to automate because of the several different formats vendors use. Different table styles, English words and even Unicode symbols are employed to enumerate device's capabilities in PICS. To overcome the formatting problem, our approach first looks at network traffic to identify BACnet devices and their capabilities. Combining the information observed in the traffic with the information written in the PICS, the algorithm interprets the PICS document. The output of the process are rules to be used in specification-based intrusion detection.

In summary, the main contributions of this paper are:

- A data-mining approach to interpret documents using network traffic, whose applicability is *not* limited to the BACnet protocol.
- A direct application of our data-mining approach that automatically extracts rules for specification-based intrusion detection in BACnet networks.
- An evaluation of the automatically generated rules in a real BACnet environment.

1.3 Organization

We first discuss the BACnet protocol in more detail Section 2. A literature review is presented in Section 3. Section 4 describes the system overview. The methodology and results are described in Sections 5 and 6. Our results are discussed in Section 7, followed by conclusions and future work in Section 8.

2 BACnet Protocol Overview

BACnet is a vendor-neutral communications protocol for Building Automation and Control Networks. Since 2003, BACnet became the ISO 16484-5 standard protocol. It is also an ANSI and ASHRAE standard. Originally created to

support only heating, ventilation, and air conditioning (HVAC), BACnet was extended to support additional functionality like access control and lighting [10]. BACnet dictates a set of rules that governs how BACnet compliant devices should communicate. Thanks to the standardization, BACnet devices are capable to interoperate regardless of the manufacturer.

BACnet was inspired by the Open Systems Interconnection (OSI) model (ISO-7498). Four layers, each of them providing services to the next one, are part of the standard (Figure 1). The *physical* and *data link* layers allow the usage of different protocols that offer advantages and disadvantages in terms of cost, performance and type of transmission. The *BACnet network* layer allows the interconnection of two or more BACnet networks. BACnet has its own routing protocol implemented in the *network* layer. On top of the BACnet protocol stack is the *application* layer on charge of the actual data exchange between BACnet devices. The information is properly encoded and encapsulated in messages called Application Protocol Data Units (APDU). The details on how to create such messages are also part of the BACnet standard. BACnet *objects* and *services* play a key role in BACnet's *application* layer.

2.1 BACnet objects

Similar to the "object" concept in Object Oriented Programming, BACnet devices store data within objects. Objects have a *type* and a set *properties*. This abstraction hides low level details such as operating system or storage mechanisms and is crucial to exchange data among heterogeneous BACnet devices.

The BACnet-2012 standard defines 54 BACnet objects to satisfy the most frequent needs in Building Automation Systems. The standard also dictates which BACnet properties are mandatory for each object. Other properties are considered optional so manufacturers are free to implement them or not. Furthermore, the standard declares the possibility to implement proprietary –vendor specific– objects and properties in BACnet devices.

A BACnet temperature sensor, for example, will very likely contain an object to store its readings from the field. In this case, the most appropriate object type is *analog input*. Some of the properties that can be found in the *analog input* object are the *object-name* (e.g. "Office #1"), *present-value* (e.g. "23"), *upper-limit* (e.g. "35"), *lower-limit* (e.g. "10"), *units* (e.g. "Celcius"), and *out-of-service* (e.g. "False").

Most properties are readable, some are writable (*object-name*) and others

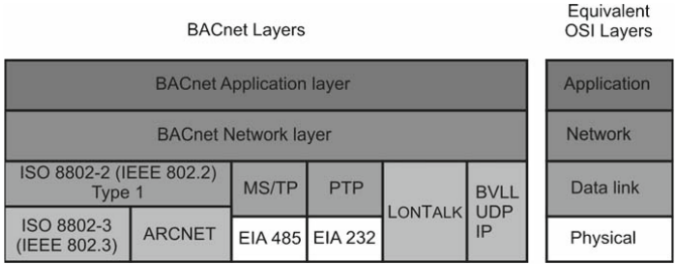


Figure 1: BACnet layers and their OSI equivalent [18].

are writable under certain conditions (e.g. *present-value* is writable only if *out-of-service* is "True") [18]. The access type (read, write or conditional write) for every property is also defined in the standard.

There is a broad variety of BACnet objects: *analog input*, *analog output*, *binary input*, *binary output*, *calendar*, and *device* are some of the most popular. The *device* object in particular, is the only mandatory object in a BACnet device. It contains properties such as *vendor-identifier*, *model-name*, *protocol-object-types-supported*, *max-apdu-length-accepted*, among others. Since the applicability of standard objects vary depending on the purpose of the device, it is not required to implement all 54 standard objects. Clause 24 of the BACnet standard ("Network Security Architecture") defines a *network security* object intended to solve the problem of unauthenticated plain-text messages; however, this clause is not mandatory and there are not known devices implementing the *network security* object yet [19].

It is important to distinguish between object *types* and object *instances*. Object *types* are templates that will be filled in with data when *instances* are created. BACnet devices that implement some object *types* must be able to store at least one *instance* of them. Some devices contain a fixed amount of *instances* by default, whereas others allow to create and delete *instances* dynamically. It is also possible that devices implement some object *types* but do not have any *instances* of them.

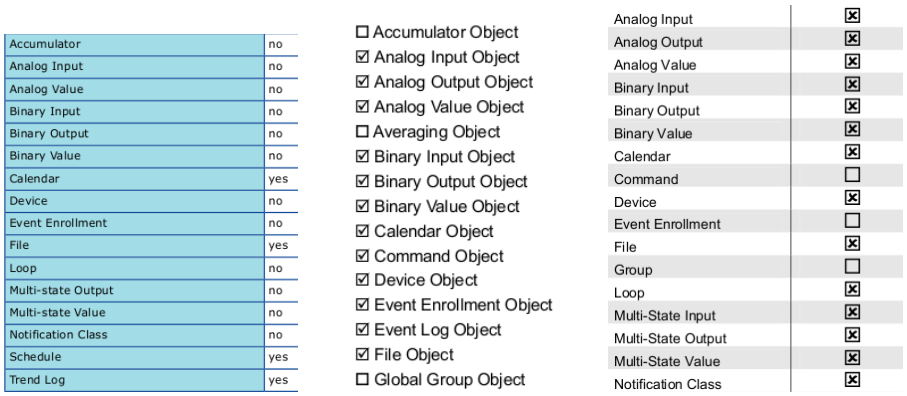
2.2 BACnet services

BACnet services refer to the capabilities that BACnet devices have. There are services to create and delete BACnet objects, services to read and write properties one at a time and several properties at once, services to receive notifications of changes, and many others. Some services are *confirmed*, which means that an acknowledgment message is expected, whereas other services are *unconfirmed*.

Read and write operations are *confirmed* services. The observation of BACnet traffic allows to identify both successful operations and errors. Common error codes are *write-access-denied* for attempts to write non-writable properties; *unknown-property* for properties not available within a specific object; and *unknown-object* for attempts to access object *instances* not available in the device, even though the object *type* could be implemented.

2.3 BACnet PICS

BACnet networks are composed of devices with very different functions. Field devices, controllers, routers, servers and workstations are some of them. Sensors typically report their readings (e.g. temperature) to BACnet controllers. Controllers can instruct actuators to perform actions (e.g. turn air conditioning on). Sensors and actuators are often called "field devices". BACnet routers interconnect different BACnet networks. The servers are often used for long-term storage of the logs generated by BACnet devices. Finally, workstations allow BACnet administrators to monitor and configure all the BACnet devices in the network. All these devices must be compliant with the BACnet standard: they communicate using BACnet services and exchange information stored in BACnet objects.



(a) Priva. (b) Kieback&Peter. (c) Siemens.

Figure 2: Excerpt of object listings in different PICS.

It is clear that different devices require different BACnet objects and services to execute their functions. In order to specify which BACnet objects, properties and services are actually implemented in *every* BACnet device, vendors must issue a document called "Protocol Implementation Conformance Statement" (PICS). Figure 2 shows how different vendors list implemented BACnet objects in PICS. Figure 3 shows examples describing specifically the *analog input* object. All figures show different styles and symbols to describe similar information.

2.4 BACnet Attacks

The BACnet standard is comprised of 4 layers, as shown in Figure 1. The two bottom layers allow to choose among different protocols. On top, BACnet implements its own routing (i.e. network) protocol and the BACnet application protocol. While our paper focuses on the application layer, it is important to note that any attacks on the underlying protocols will affect the upper layers as well.

BACnet nowadays controls services like HVAC, lighting, water, life-safety (e.g. fire alarms, foam system), and security (e.g. CCTV, physical access control). Furthermore, there are Working Groups[2] for elevators (EL-WG) and smart grids (SG-WG), which will lead to the addition of these services in the near future. As stated in [7], the "integration of security-critical services demands the underlying control system to be reliable and robust against malicious manipulations".

Since BACnet is essentially a computer protocol involving a BACnet network and BACnet devices, it is possible to classify BACnet attacks in the same way as in "regular" TCP/IP computer networks. Other attacks, known as process control subverting attacks, are specific to the BAS context because they interfere with physical processes using BAS actuators.

Denial of Service. DoS attacks "prevent legitimate users of a service from using that service"[26]. Researchers have compromised the availability of BACnet devices by means of malicious messages [13]. In standard TCP/IP

[2]http://www.bacnet.org/WG/index.html

(a) Delta Controls Inc - enteliBUS CPU Engine

Object-Type	Supported	Dynamically Creatable	Dynamically Deletable	Optional Properties Supported	Writable Properties
Analog Input	☑	☑	☑	Description, Device Type, Reliability, Min Pres Value, Max Pres Value, Update Interval, Resolution, COV Increment, Notify Type, Acked Transitions, Event Time Stamps	Object Name, Description, COV Increment, Present Value (conditional), Out of Service, Reliability (conditional), Time_Delay, Notification_Class, High_Limit, Low_Limit, Deadband, Limit_Enable, Event_Enable, Notify_Type

(b) Siemens - Desigo V4

Analog Input	Optional properties supported	Writable properties	Property range restrictions
		Present_Value	
	Description		
	Reliability		
		Out_Of_Service	
	Resolution		
	COV_Increment		
	Time_Delay	Time_Delay	
	Notification_Class		
	High_Limit	High_Limit	
	Low_Limit	Low_Limit	
	Deadband	Deadband	0 .. maxReal
	Limit_Enable		
	Event_Enable	Event_Enable	
	Acked_Transitions		
	Notify_Type		
	Event_Time_Stamps		
	Profile_Name		

(c) Priva BV - TC BACnet router

Analogue Input object

Properties	Optional/ required	Read / Write
Object_Identifier	R	R
Object_Name	R	R
Object_Type	R	R
Present_Value	R	R/W*
Description	O	R
Status Flags	R	R
Event State	R	R
Reliability	O	R
Out_of_Service	R	R/W
Units	R	R
COV_Increment	O	R/W

* Writeable if out-of-service is TRUE

Figure 3: Analog-input object representation in different PICS.

networks, there are usually several paths between any two hosts (the network is a graph). BACnet's simplistic routing protocol allows only one path (the network is a tree). A successful attack on a vulnerable BACnet router will disable not only the device itself, but also the network segments connected through it. Thus, the proportion of the attack is magnified depending on the logical location of the vulnerable device in the network.

Exploitation attacks. This kind of attacks exploit flaws in software usually via unsanitized input. The outcome of the attack could be a DoS, gaining access to the system or execution of arbitrary code. BACnet software, either in embedded devices or at the operator's workstation, is susceptible to exploitation attacks. Examples of exploitation attacks in BACnet can be found in Mitre's Common Vulnerabilities and Exposures database[3]. Firmware updates in BACnet devices are rarely applied [28], which increases the chances of successful exploitation attacks.

Snooping attacks. Snooping attacks assume that the attacker has access to the network. Snooping attacks are possible because current BACnet implementations do not use encryption. Even with encryption in place, side-channel attacks have been successful enough to perform snooping attacks in other protocols [5]. Similar techniques could be used for BACnet. Unauthorized information gathering from a BAS is not only a security problem but also a privacy sensitive issue. From a security point of view, snooping attacks allow intruders to enumerate and fingerprint devices, which is usually the first step before launching exploitation attacks. On the privacy side, information about human activities can be easily collected and inferred from the BACnet traffic: presence or absence in rooms, temperature and lighting preferences, tracing people's paths through badge locked doors, etc.

Process control subverting. Process control subverting attacks [3] are particularly important because they combine digital actions and physical consequences. Attackers can gain access to BACnet software through exploitation attacks and then modify the automation and control processes set up in the building. Taking into account the broad set of services controlled by BACnet, life threatening situations can occur (e.g. people being stuck in elevators). It is clear that attacks on BACnet are grave, however, simultaneous execution of BACnet *and* physical attacks can be catastrophic (e.g. disabling fire alarms, water supply, locking doors, turning the lights off and then setting a fire in the building).

3 Literature Review

Specification-based intrusion detection has been applied in very diverse contexts. This detection approach was originally presented in [15] as an alternative to IDSs encoding *misuse behavior*. Their research aimed at detecting malicious activities executed by vulnerable privileged software running on UNIX systems. Using a *Policy Specification Language* the authors were able to formalize normal behavior, and therefore note when exploited applications diverted from its normal operations.

In specification-based intrusion detection, rules are often extracted from unstructured sources of information. This is considered difficult and requires hu-

[3] https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=BACnet

man intervention. Several researchers have used textual descriptions of network protocols to *manually* extract rules [12, 20, 24]. For most protocols, including BACnet, there are a few documents (RFCs, IEEE or ISO standards) describing them; however, actual implementations do not strictly adhere to these documents [1, 6]. As a consequence of slightly different protocol implementations, specification-based intrusion detection must use loose rules, otherwise there is a risk of triggering false alarms [24]. The drawback of loose rules is that some attacks might not be detected (false negatives). Some researchers have found that adding anomaly-based detection can help to overcome this problem [25, 24] but the complexity of the system increases.

In the specific case of the BACnet protocol, PICS represent additional sources of information describing the list of objects (with their corresponding properties) implemented in each device. The BACnet standard demands a strict match between device's capabilities and PICS, which allows us to create tight rules from PICS without facing bad detection problems. Since rules extraction from possibly hundreds of PICS is not feasible for real world BACnet deployments, our research aims to automate this process. Following this approach, IDS alerts mean a deviation from the expected behavior described in the PICS. Likewise, non-alerts mean adherence to the PICS, which according to the standard should *always* be the case [19].

The location of the IDS defines the audit data to be scrutinized. While network IDSs are usually placed in network gateways with the intention to capture all incoming and outgoing traffic, host-based IDSs monitor only local events. Sekar *et al.* [24] created a network specification-based IDS with a Finite State Machine modeling the TCP/IP traffic from a router's point of view. In [12] a similar IDS was proposed for the ZigBee (IEEE 802.15.4) protocol. In their proposal the IDS was located in the ZigBee Personal Area Network (PAN) coordinator, which is on charge of forwarding messages among PAN devices. Since our particular interest lies in BACnet network traffic, BACnet routers and switches are considered ideal locations to apply our automatically generated rules.

Although our paper focuses on *detection* some efforts have been made in *prevention*. Kaur *et al.* [13] developed a traffic normalizer that either drops or fixes (whenever possible) malformed BACnet packets. The poor input handling in BACnet devices can make them fail on unexpected messages [13]. Their approach does not deal with specific details of every device and instead only considers general aspects of the protocol like header fields length or valid flags status.

The idea of a training phase in specification-based intrusion detection is mentioned in [4] in the context of mobile phones. Mobile malware often send SMS messages or dial numbers without the users knowledge and consent. The authors idea was to create a framework capable to differentiate between *software* generated actions and *user* generated actions. During the training phase the system computes hashes of running applications in order to notice future tampering on those applications. The training phase is not directly related with the process of rule generation. We also consider a training phase in which we collect BACnet traffic that will help in the correct interpretation of PICS.

A common characteristic among all the aforementioned papers is the requirement of creating *manual* specifications. In [25], Stakhanova *et al.* automate this tedious and error-prone process using a hybrid specification/anomaly

based IDS. The IDS detects wrong behavior of GNU/Linux application programs (e.g. *cat, mount*) using the sequence of *system calls* they make. The anomaly-based subsystem, supported by a machine learning classifier (SVM), automatically populates with rules the specification-based subsystem. Rules in our system are automatically generated from trusted sources (PICS written by manufacturers) explicitly dictating the right behavior of devices.

Semi-automated rules creation in Networked Control Systems (NCS) is analyzed in [3]. NCSs like Industrial Control Systems, In-Vehicle Networks and Building Automation Systems, often have a variety of documentation that can be used to generate rules. The authors' proof of concept was done with the BACnet protocol, considering not only PICS but also configuration files as sources of information. Our system completely automates the process of rules creation using a novel data-mining approach that interprets PICS by means of network traffic previously analyzed during a training phase.

4 System Overview

Our research aims to automatically generate rules to be used in specification-based intrusion detection. Rules are extracted from PICS written by manufacturers. PICS are published by *BACnet Testing Laboratories* that certify devices as BACnet compliant[4].

We assume the possibility to collect BACnet traffic in pcap format from the same network where the IDS is going to be deployed. Furthermore, we assume the existence of a local repository storing the PICS of all the devices in the network.

The Analysis Engine shown in Figure 4 is on charge of the rule extraction process. The procedure starts with a **training phase** whose input is BACnet traffic previously collected from the network. The observation of BACnet traffic allows the Analysis Engine to keep record of active BACnet devices and their features like present and absent BACnet objects and properties (Figure 4, label 1). This information is extracted using the BACnet protocol parser provided by [3]. It is required that the traffic used during the training phase contains only

[4]http://bacnetinternational.net/btl/

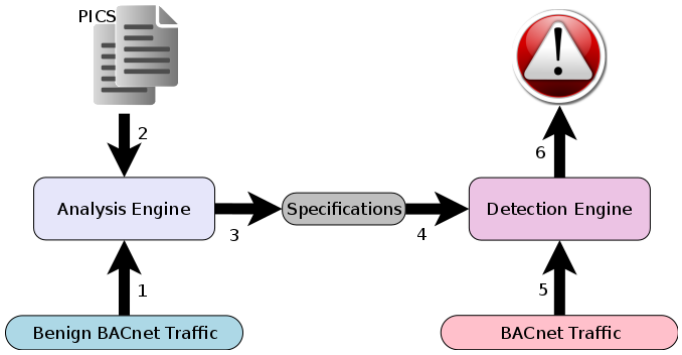


Figure 4: System overview.

legitimate BACnet dialogues. Malicious BACnet messages could mislead the Analysis Engine to wrong conclusions (e.g. false observations). At the end of the training phase, the Analysis Engine automatically matches each device observed during the training phase with one of the PICS stored in local repository.

The **PICS interpretation phase** leverages on the training phase to make sense out of tables, words and symbols commonly found in PICS (Figure 4, label 2). If a PICS, for example, contains all the implemented properties next to a Unicode symbol (e.g. ☑), while the non-implemented properties are next to a different symbol (e.g. ☒), then the Analysis Engine can compare its observations during the training phase with the two sets of properties in the PICS. The set of properties tagged as "☑" will have several elements in common with the set of observed properties, whereas the set of properties tagged as "☒" will not. In this way the Analysis Engine is capable to discern among the different sets of properties and deduce that the "☑" symbol means "implemented properties". PICS reading automation is the basis to create rules from the PICS' content.

The output of the Analysis Engine are *specifications*, also known as rules, stating allowed BACnet objects and properties per device (Figure 4, label 3). The access type of the properties –read or write– is also part of the rules.

A Detection Engine takes the automatically generated rules as input (Figure 4, label 4). Once the rules are loaded, the Detection Engine verifies concordance between BACnet packets and rules (Figure 4, label 5). It is worth noting that the rules constitute a *white list* of allowed BACnet objects, properties and access types. Messages to or from identified devices containing BACnet objects, properties or access types *not* mentioned in the rules will be considered security violations (Figure 4, label 6), otherwise the messages are considered benign.

5 Method

A detailed description of the tasks executed by the Analysis Engine followed by our experimental results are described in Section 5.1 and Section 5.2. The *rules creation* process is described in Section 5.3.

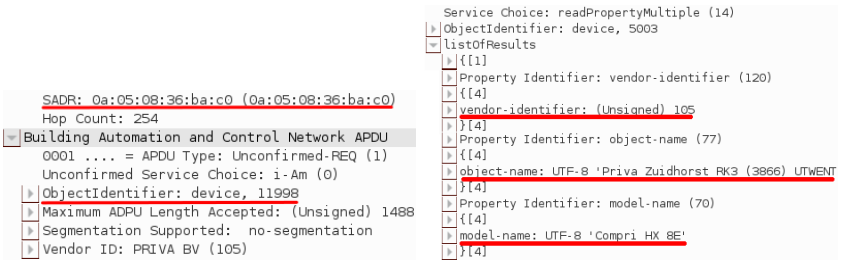
5.1 Training Phase

The observation of BACnet traffic has a twofold purpose. First, to create an inventory of legitimate BACnet devices in the network. Second, to keep record of present and absent BACnet objects and properties per device. The access type (read or write) for every property is also logged.

BACnet devices. The BACnet standard defines a service that allows devices to advertise themselves in the network. The service is called "I-Am". "I-Am" messages contain information that univocally identify BACnet devices: the BACnet MAC address and the device ID. Figure 5a shows an excerpt of an "I-Am" message. Moreover, queries on the *device* object contain very detailed information (BACnet properties) such as *model_name, vendor_identifier* and *object_name* (Figure 5b). Those properties are used at a later stage, as *keywords* to match the correct PICS for every device.

PICS stored in a local repository are indexed using Apache Solr[5]. This tool consists of a web server that allows searches on indexed documents using

[5]https://lucene.apache.org/solr/



(a) I-Am message. (b) Device's properties.

Figure 5: Messages used to identify devices.

keywords. The goal of the device identification process is to group all the devices by PICS.

In our implementation at the University of Twente, the local PICS repository consists of 6 different files. During the training phase we used 5 days (2 gigabytes) of BACnet traffic. We discovered 223 out of the 225 devices listed by the BACnet administrators. Discovered devices were matched with one of the PICS available in the repository. A manual verification confirms that BACnet compliant devices were correctly matched with their corresponding PICS. A breakdown of the amount of devices per PICS is shown in Table 1.

Those devices were identified using the first 160.000 BACnet packets of the training data, which in our particular network represents roughly one hour (20 MB) of traffic (see Figure 6a).

# Devices	Vendor	PICS file
2	Delta Controls	PICS-for-enteliBUS-CPU-Engine-340.pdf
2	Priva	BACnet-PICS-Blue-ID-S10-Controller.pdf
6	Siemens	DESIGO-PX-Router.pdf
6	Kieback&Peter	PICS-EN-DDC4400.pdf
9	Siemens	DESIGO-PX-Station.pdf
198	Priva	PICS-TC-BACnet-Router.pdf

Table 1: Devices grouped by PICS.

BACnet objects and properties. Besides device identification, it is also crucial to learn as many BACnet object types –and their properties– as possible. To do so we use a technique originally described in [3] as "BACnet address linking". It consists in looking for two features in every BACnet packet: (1) the BACnet source address; and (2) BACnet objects and properties. Then, it is possible to link the objects and properties to a specific device ID using the source address (Figure 5a). Since in our system devices are already matched with PICS, there is an indirect link between those objects and properties with their corresponding PICS.

BACnet devices are regularly queried for their properties. The fact that a response contains the current value of a particular property proves that the respondent device actually has the attribute and the object it belongs to, otherwise an error would have been sent. Moreover, the *device* object has a mandatory

property called *protocol-object-types-supported*, which is also commonly queried. This property contains all the implemented and not implemented object types in the device.

The access type (read or write) of properties can also be confirmed by scrutinizing network traffic. Write requests in the BACnet protocol must be confirmed by the recipient. Acknowledgment messages after write requests, allow us to identify the set of writable properties. Similarly, error messages offer the possibility to identify non-writable properties.

Using the data available at the University of Twente we were able to discover 3.933 properties (Figure 6c). Our system also found 5.566 objects. Figure 6b shows the difficulty in finding new objects after 100.000 BACnet packets (approximately 40 minutes of traffic in our setting). The access type of properties is also logged by the system, however only 2 BACnet properties were correctly linked with one of the discovered devices. The abnormally low amount of write requests in our training traffic is due to a reorganization process carried out in the network at the time of collecting the traffic.

5.2 PICS Interpretation Phase

While PICS contents are clearly specified in the standard, PICS format is not standardized. Tabular information arrangements are commonly employed in PICS. Since there are several ways to accommodate BACnet objects and properties in tables, our approach defines an extendable set of templates from which it is capable to extract information.

PICS are usually provided in PDF format, however, our prototype works with PICS converted to spreadsheets in XML format. PDF parsing [21] is beyond the scope of this paper, however, PICS files conversion can be automated with tools such as *Adobe Reader Pro*, which we have used in our work.

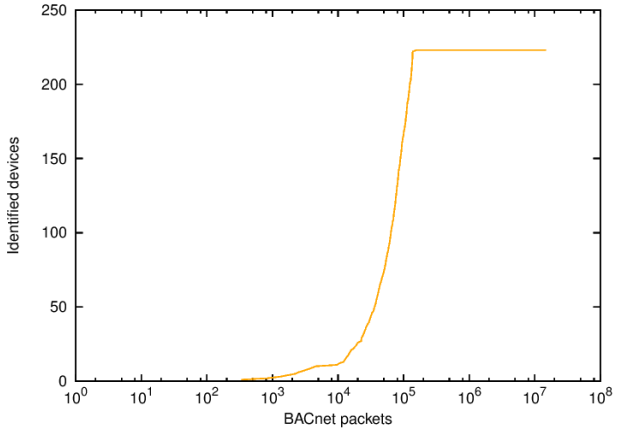
PICS reading automation relies in the correct spelling of BACnet objects and properties. However, typos and abbreviations are commonly found in PICS. In order to overcome this problem we implemented a typos correction routine based in the Levenshtein distance [16]. The Levenshtein distance counts the minimal number of insertions, deletions and substitutions to edit one string into another.

Sections 5.2.1 and 5.2.2 show how our algorithm extracts implemented BACnet objects and properties from PICS.

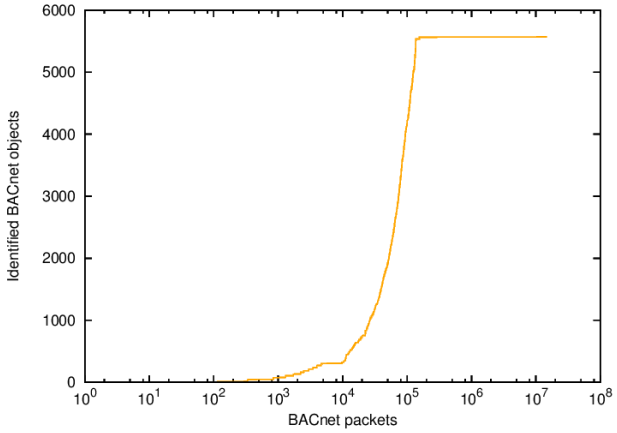
5.2.1 Extraction of Implemented BACnet Objects

Most PICS specify which BACnet objects are implemented by means of tables as those shown in Figure 2. Human understanding of language and symbols makes it easy to realize which objects are implemented and which are not. Our approach to automatically *"understand"* such tables consists in creating templates that the system can identify when reading the PICS. Figure 7 shows an abstract version of the tables in Figure 2, where orange circles denote BACnet objects. The template consists of an undetermined number of objects, one per row, but all of them in the same column. Furthermore, arbitrary texts are allowed in the columns to the right and left of the object.

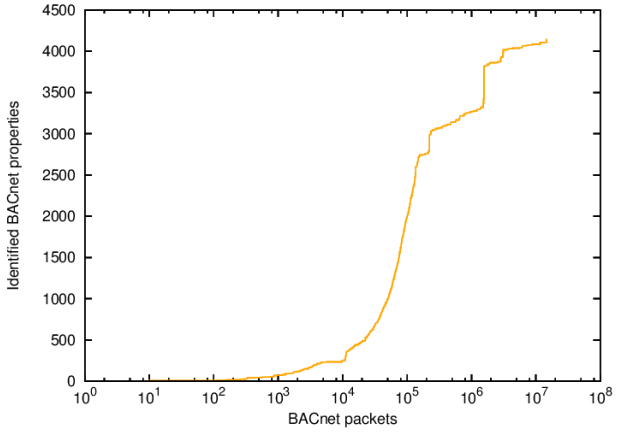
The interpretation process creates groups of objects based on the texts found on each object's side columns. Using Figure 2a as an example, the algorithm



(a) Identified BACnet devices.



(b) Identified BACnet objects.



(c) Identified BACnet properties.

Figure 6: BACnet packets required during the training phase.

creates two sets:

$$yes = \{calendar, file, schedule, trend\ log, ...\}$$

$$no = \{accumulator, analog\ input, analog\ value, ...\}$$

These sets are called *reference sets*. Set *"yes"* and set *"no"* contain the implemented and not implemented object types, respectively. Several *reference sets* can be created depending on the number of columns and texts present in the table. Any sequence of one or more characters, including Unicode symbols, is used to create *reference sets*. Furthermore, the algorithm also creates a *reference set* with the union of all the others *reference sets*.

In order to identify which of the *reference sets* contains the implemented objects, the algorithm uses the (possibly incomplete) list of present objects that was collected during the training phase. It is worth recalling that present and absent objects are grouped by PICS. Comparing the similarity of the collected list, against all the *reference sets* allows the algorithm to choose the "winner" set. The Jaccard similarity coefficient is a simple yet effective scoring mechanism to compare sets [11]. Equation 1 defines Jaccard similarity, where P is the set of *present* objects observed during the training phase, and R_n is the n^{th} *reference set*.

$$Jaccard(P, R_n) = \frac{|P \cap R_n|}{|P \cup R_n|} \tag{1}$$

Since *absent* objects (denoted as A) are also collected during the training phase, we implemented a modified version of the Jaccard similarity to improve the accuracy of the algorithm. Equation 2 shows our *Jaccard' similarity*, that lies at the core of our extraction algorithm. $Jaccard'$ adds new terms to the $Jaccard$ similarity with the following effect:

1. If R_n is a suitable *reference set*, it should contain most (ideally all) of the objects in P, which means that $|P - R_n|$ must be equal or close to zero.

2. If R_n is a suitable *reference set*, it should contain few (ideally none) of the elements in A, which means that $|A \cap R_n|$ must be equal or close to zero.

Deviations from the ideal cases will increase the penalization (denominator) and cause a decrease in the overall scoring for R_n. On the other hand, the ideal cases imply that $Jaccard'(A, P, R_n)$ is identical to $Jaccard(P, R_n)$.

$$Jaccard'(A, P, R_n) = \frac{|P \cap R_n|}{|P \cup R_n| \cdot (|P - R_n| + |A \cap R_n| + 1)} \tag{2}$$

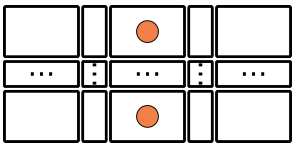


Figure 7: BACnet objects table template.

Results about the extraction of implemented objects using $Jaccard'$ are shown in Section 6.1.1 Table 2.

Object Extraction Summary. In order to extract the set of implemented object types from the PICS, a table template is created (Figure 7). The template is used to match similar tables in the PICS document. Once the table has been read, the algorithm creates as many *reference sets* as possible, and then ranks each of them using a modified version of the Jaccard similarity (Equation 2). The *reference set* scoring the highest similarity will be considered the complete set of implemented objects according to the PICS. A pseudo-code description is shown in Algorithm 1.

Algorithm 1 Implemented Objects Extraction Algorithm.

```
1: function OBJECT–EXTRACTION(PICS_i, Set_P_PICS_i, Set_A_PICS_i)
2:     Extract from PICS_i all reference-sets
3:     for each reference-set do
4:         similarity = jaccard'(Set_A_PICS_i, Set_P_PICS_i, reference-set)
5:     end for
6:     return reference-set with highest similarity
7: end function
```

5.2.2 Extraction of Implemented and Writable BACnet Properties

Extraction of implemented properties from PICS is very similar to the approach taken to extract objects. There are, however, two main differences: (1) the properties are grouped by object; and (2) different PICS use disparate table types to write the properties. Figure 8 shows three table templates from which our current implementation is capable to extract information, where orange circles represent BACnet objects and green circles represent BACnet properties. Linking the templates shown in figure 8 with the real examples shown in Figure 3, it is possible to find a match between Figures 3a-8a, 3b-8b and 3c-8c. From a sample of 786 PICS available for download[6], 73.28% of the PICS match with any of those three templates. All the PICS used in our infrastructure also belong to those templates.

Vendors commonly describe a few attributes about BACnet properties: access type, obligatoriness according to the standard, range restrictions and whether they are implemented or not. More or fewer attributes are shown in different PICS. Figure 3 shows how all these attributes about BACnet properties are differently encoded. Our main interest lies in the identification of implemented properties and their access type.

The algorithm to identify implemented properties consists in grouping the object's properties in different sets (*reference sets*). Then, using the properties observed during the training phase (set P), along with the set of absent properties (set A), the algorithm uses our modified Jaccard similarity (Equation 2) to choose the "winner" set. The *reference set* scoring the highest similarity will be considered the complete set of implemented properties according to the PICS.

Results about the extraction of implemented properties using $Jaccard'$ are shown in Section 6.1.2 Table 3.

[6] https://bacnetinternational.net/btl/

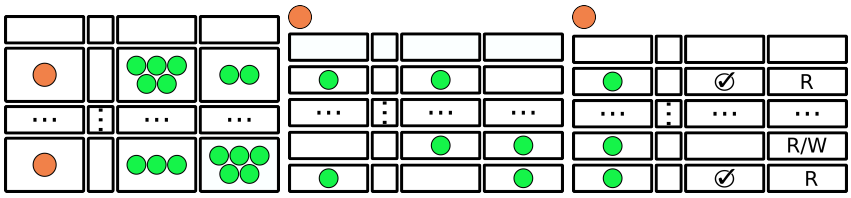


Figure 8: BACnet properties table templates.

Extraction of writable properties is done with the same algorithm but different parameters. If writable properties are specified in the PICS, one of the *reference sets* will contain those properties. Then, using the sets of writable and non-writable properties extracted during the training phase, the algorithm ranks the *reference sets* using our modified Jaccard similarity.

Reference sets creation. For those objects whose properties are described as in Figure 8a, we take advantage of the groups of properties already bounded within a cell. Each of those groups becomes a *reference set*. The table template shown in Figure 8b suggests a column-wise grouping. *Reference sets* are comprised of all the BACnet properties in the same column. Finally, properties arranged as in Figure 8c are grouped using the strings shown per row. In Figure 8c, for example, we would create a *reference set* containing all the properties marked with an "R", a different *reference set* with all the properties marked with an "R/W", and lastly a *reference set* with the checked (i.e. ☑) properties. Regardless of the table template, we also create a *reference set* with the union of all the others *reference sets*.

Property Extraction Summary. The interpretation of tables containing implemented properties is based on the observations carried out during the training phase. Present and absent properties for the specific BACnet object under analysis, come from the database created during the training phase. The *reference sets* are retrieved from the PICS according to the table templates. Finally, the algorithm ranks the *reference sets* using the modified version of Jaccard similarity shown in Equation 2. A pseudo-code description is shown in Algorithm 2.

Algorithm 2 Implemented Properties Extraction Algorithm.

1: **function** PROPERTY–EXTRACTION($Object_i$, $Set\_P_{Object_i}$, $Set\_A_{Object_i}$)
2: Extract from PICS all *reference-sets* for $Object_i$
3: **for** each *reference-set* **do**
4: similarity = jaccard'($Set\_A_{Object_i}$, $Set\_P_{Object_i}$, *reference-set*)
5: **end for**
6: **return** *reference-set* with highest similarity
7: **end function**

5.3 Automatic IDS Rule Creation

The goal of our research is to automatically create rules for specification-based intrusion detection. For every PICS, the system creates three sets of rules con-

taining implemented objects, implemented properties and writable properties. Since all the devices are grouped by PICS, it is straightforward to link every device with its corresponding sets of implemented objects, implemented properties and writable properties.

The rules consist in an exhaustive list of devices found in the network, along with the corresponding sets of valid objects and properties. Even though some PICS do not include mandatory properties, we added them to the rules in order to avoid false alerts. In the same way that rules were created in [3], any attempt to access an object or property is verified with the corresponding rules (Algorithms 3 and 4). Furthermore, any attempt to write a property is verified with the set of writable properties (Algorithm 5). For every device, all outgoing and incoming packets are verified against the rules.

Since rules for objects and rules for properties are created independently, contradictions can occur. Rules for objects are created based on object listings (Figure 2), while rules for properties are extracted from different tables (Figure 3). One of our PICS has typos in the objects list (see Section 6.1.1). As a consequence, 2 objects were not added to the list of valid objects. Later in the same PICS, the affected objects were correctly spelled in the properties tables. Thus, the 2 objects with their properties are considered as valid by the second rules set.

Our prototype uses Bro IDS[7] as a network traffic analyzer. Rules are generated using Bro's scripting language, however, the system could be adapted to create rules in other formats.

Algorithm 3 Object Rules Format.

1: **if** $BACnetObject \notin Device_{AllowedObjectTypes}$ **then**
2: Alert("Forbidden BACnet Object")
3: **end if**

Algorithm 4 Property Rules Format.

1: **if** $BACnetProperty \notin Device_{ObjectType_{AllowedProperties}}$ **then**
2: Alert("Forbidden BACnet Property")
3: **end if**

Algorithm 5 Writable Property Rules Format.

1: **if** $write(BACnetProperty) \notin Device_{ObjectType_{WritableProperties}}$ **then**
2: Alert("Forbidden BACnet Property writing")
3: **end if**

6 Results

The evaluation of our algorithm is done in two levels. First, we measured the effectiveness of our approach to create rules concordant with the PICS (Section

[7] https://www.bro.org/

6.1). Second, we applied the rules in our BACnet network to analyze their impact in a real BACnet environment (Section 6.2).

All the results presented in this section are based in the BACnet network located at the University of Twente.

6.1 Evaluation Using PICS

As we already showed in Table 1, 223 BACnet devices in our network are grouped into six different PICS. For each of those PICS we ran our algorithm to extract implemented objects, implemented properties and writable properties. In order to measure the accuracy of the extracted *elements* (i.e. objects and properties), we created a *ground truth* list by hand.

We will refer to the *elements* consistent with the PICS as *matches*. Extracted *elements* not in the PICS will be considered *mismatches*. Finally, not extracted *elements* will be denoted as *omissions*.

6.1.1 Implemented objects

Table 2 shows the results gotten after running the algorithm described in Section 5.2.1. Overall results show that 97.8% of the objects described in the PICS were correctly extracted and 2.2% were omissions, all of them in one single PICS. All the objects tagged as implemented by the algorithm were indeed implemented according to the PICS (no mismatches).

PICS	**# Objects**	**Matches**	**Mismatches**	**Omissions**
PICS-for-enteliBUS-CPU-Engine-340	19	19	0	0
BACnet-PICS-Blue-ID-S10-Controller	16	16	0	0
DESIGO-PX-Router	N/A	N/A	N/A	N/A
PICS-EN-DDC4400	22	22	0	0
DESIGO-PX-Station	18	18	0	0
PICS-TC-BACnet-Router	16	14	0	2
TOTAL	**91**	**89**	**0**	**2**

Table 2: Automatic object extraction performance.

The DESIGO-PX-Router PICS does not contain a table of implemented object types. Instead, there is a legend saying *"Only the device object is supported."*. This is clearly not compliant with our template and therefore the algorithm does not work.

The 2 omissions in the PICS-TC-BACnet-Router were caused by typos. The object *"trend log"* is written simply as *"trend"* and *"notification class"* written as *"notification"*. Even though we implemented a typos correction routine, the Levenshtein distance in these two cases is greater than the threshold set in our prototype (distance ≤ 2), therefore no fixing was applied. The algorithm uses pattern matching to identify the template (Figure 7) in the PICS. Object names must adhere to the standard, otherwise they will not be found.

6.1.2 Implemented properties

The results of the algorithm described in Section 5.2.2 are shown in Table 3. The algorithm extracted 94.0% of the properties implemented and shown in the PICS. There are also 2.9% of mismatches and 6.0% of omissions. It is important to recall that some PICS (e.g. Figure 3a) do not show all the implemented properties, but only the optional. Since our evaluation is based in the contents of the PICS, implicit properties are not taken into account.

PICS	**# Properties**	**Matches**	**Mismatches**	**Omissions**
PICS-for-enteliBUS-CPU-Engine-340	231	206	9	25
BACnet-PICS-Blue-ID-S10-Controller	241	241	0	0
DESIGO-PX-Router	7	7	1	0
PICS-EN-DDC4400	520	520	25	0
DESIGO-PX-Station	241	196	3	45
PICS-TC-BACnet-Router	237	218	5	19
TOTAL	**1477**	**1388**	**43**	**89**

Table 3: Automatic property extraction performance.

We identify three main causes of omissions and mismatches:

1. **Typos and abbreviations:** As we already pointed out in Section 6.1.1, typos can lead to problems while identifying BACnet objects and properties. Exact pattern matching fails in presence of typos or abbreviations. Even though the system is flexible about word separators (white space, underscore or dash), other variations are not taken into consideration. We use Levenshtein's distance to fix typos in PICS. The correction procedure is intended to fix minor typos in BACnet's objects and properties names. Words in the PICS are fixed if the Levenshtein distance to a BACnet object or property name is less or equal than 2. Words with greater Levenshtein distances are not modified.

2. **Table format misinterpretation:** The system identifies table templates (Figure 8) by means of keywords, which are BACnet objects and properties. The location of those keywords in the table can alter the internal representation of the table that is used at a later stage to generate the rules. A concrete example of the *table format misinterpretation* problem is shown in Figure 3a, where "Object-Type" is a column header but it is also a standard BACnet property name. Another cause of *table format misinterpretation* comes from the typos correction procedure. A high threshold for the Levenshtein distance causes that many words (e.g. English stop words) in the PICS become BACnet objects and properties. In this case, it is hard for the system to identify any of the 3 templates it is capable to extract information from.

3. **Wrong set selection:** Regardless of the table template being analyzed, a variable number of *reference sets* are created. Our modification on

the *Jaccard similarity* decides which of them is the set of implemented properties according to the PICS. Insufficient data during the training phase can lead to selection errors.

Typos and abbreviations cause 4.7% of the mismatches and 22.5% of the omissions. On the other hand, *table format misinterpretation* problems cause 95.3% of the mismatches and 77.5% of the omissions. Even though *wrong set selection* was not a problem with our 2GB training dataset, previous tests (not reported in this document) with fewer training data suffered from this problem.

6.1.3 Writable properties

In order to extract writable properties it is crucial to have enough write requests during the training phase. Due to special circumstances in the BACnet network where our training dataset was collected, all the write requests were directed to only two BACnet properties (analog-value → present-value and binary-value → present-value) in one device. Under these conditions we were not able to extract writable properties from any PICS.

Our observations extracting *implemented* properties suggest that at least one-third of the *writable* properties (for a single object) must be learned during the training phase in order to successfully extract all of them.

6.2 Rules Evaluation with BACnet Traffic

In this section we present our results after analyzing 5 days of real BACnet traffic with the automatically generated rules.

Matches, mismatches and omissions in the rules determine the performance of our approach when dealing with real BACnet traffic.

Omitted rules are BACnet objects and properties showed in the PICS as implemented that were not added to the sets of valid elements. Therefore, the IDS triggers false alerts whenever those elements are observed in the traffic.

Mismatches are BACnet objects and properties *not* showed in the PICS as implemented that were added to the sets of valid elements. Therefore, the IDS do not trigger alerts whenever those elements are observed in the traffic.

Finally, **Matches** are BACnet objects and properties showed in the PICS as implemented that were correctly added to the sets of valid elements. Therefore, the IDS do not trigger alerts whenever those elements are observed in the traffic. However, the IDS does trigger alerts whenever elements *not* showed in the PICS as implemented are observed in the traffic.

Our tests with real BACnet traffic allowed us to understand the effect of the matches, mismatches and omissions during the intrusion detection phase. Table 4 summarizes our results per PICS file. We counted how many unique alerts were triggered because of matches (M) and omitted (O) rules. Furthermore, we manually checked the traffic with Wireshark[8] in order quantify how many alerts were missing because of the mismatches (MM).

PICS-for-enteliBUS-CPU-Engine-340. Objects rules were fully concordant with the PICS. As a consequence, the count of alerts (for omissions) and non-alerts (for mismatches) was kept in zero. Matching rules, however, triggered one alert. Attempts to access a proprietary object raised the alarm.

[8] https://www.wireshark.org/

PICS	Objects rules			Properties rules		
	M	**MM**	**O**	**M**	**MM**	**O**
PICS-for-enteliBUS-CPU-Engine-340	1	0	0	13	0	1
BACnet-PICS-Blue-ID-S10-Controller	0	0	0	0	0	0
DESIGO-PX-Router	N/A	N/A	N/A	5	0	0
PICS-EN-DDC4400	0	0	0	0	0	0
DESIGO-PX-Station	1	0	0	14	0	4
PICS-TC-BACnet-Router	0	0	2	3	0	8
TOTAL	**2**	**0**	**2**	**35**	**0**	**13**

Table 4: Impact of the rules during the detection phase.

Properties rules triggered 13 alerts because of matching rules, 11 of them caused by proprietary properties and 2 caused by attempts to access standard properties not implemented in this kind of devices. The 25 omissions in properties rules caused only 1 alarm. A manual inspection in the traffic, confirmed that the 9 mismatches did not cause that important alarms were dismissed.

BACnet-PICS-Blue-ID-S10-Controller. Objects and properties rules were fully concordant with the PICS. Since this PICS did not suffer from omissions or mismatches in the generated rules, both counts remain in zero for objects and properties rules.

The traffic analyzed for this kind of devices contained only valid BACnet objects and properties. Therefore, the matching rules did not trigger any alerts either.

DESIGO-PX-Router. As it was explained in Section 6.1.1, no objects rules were generated by our approach for this specific PICS.

Properties rules, however, triggered 5 alerts because of matching rules: 2 of them caused by proprietary properties in the *device* object and 3 because of unimplemented standard properties in the same object. The mismatched property (*mode*), was not found in the traffic. No omissions in the automatically generated rules avoid false alarms.

PICS-EN-DDC4400. None of the rules triggered alarms on any of the 6 devices described by this PICS. The 25 mismatches for properties rules did not cause any problem during our evaluation period. None of those properties was observed in the traffic.

DESIGO-PX-Station. Objects rules triggered 1 alert caused by a proprietary object not listed in the PICS. The fact that objects rules do not have omissions avoids false alerts at detection time.

Properties rules reported 14 unique alerts caused by matching rules: 8 of them were caused by proprietary properties in standard objects, 3 because of standard properties in proprietary objects and the remaining 3 because of unimplemented standard properties. The 45 omissions caused only 4 unique alerts. After manually inspecting the traffic we concluded that the mismatches did not have a negative effect for this PICS.

PICS-TC-BACnet-Router. Table 2 shows 2 omissions in objects rules. Since both objects were observed during the real traffic analysis, the IDS triggered alarms for both of them.

For properties rules, 8 out 19 different possible alerts were triggered because

of the omissions. Three alarms were correctly raised after unimplemented properties were observed. A manual inspection of the traffic confirmed that the 5 properties mistakenly added as valid (mismatches), were not observed in the traffic.

In summary, a higher amount of alerts are triggered by properties rules. Manual inspections discarded that mismatches had a negative effect during our evaluation period.

Omitted rules caused 15 unique false alerts in total: 2 for objects and 13 for properties. Omitted rules should be minimized as much as possible to avoid false alarms. All the 37 alerts from matching rules are caused either by proprietary or unimplemented elements. It is important to clarify that asking for unimplemented elements is allowed by the BACnet standard and should not be considered as an attack. Instead, we consider it as an event of interest that could be a sign of a snooping attack.

7 Discussion

Our approach leverages on two assumptions: first, the availability of enough training traffic; second, a repository of PICS describing *all* the devices in the network.

The BACnet traffic required for training purposes must contain as much objects and properties as possible. More important than the amount of traffic is the diversity of information in it. Our own experiments suffered from monotonic traffic regarding write requests. Even though we observed several thousands write requests in our training dataset, all of them were directed to only two BACnet properties. Due to the scarce information available, the system was unable to extract rules about writable properties. Having control of the BACnet network during the traffic capture, would achieve good results in short time. BACnet operators could methodically execute read and write requests to at least one device of every kind. This ensures the gathering of enough information to properly interpret all the PICS.

The foundation of our algorithm is the interpretation of PICS. Automatically generated rules are as good as the PICS they come from. Typos and non-standard abbreviations lead to a considerable amount of omitted rules with subsequent false alarms during the detection phase.

Our PICS reading approach is based in a limited but extendable set of table templates. Our aim is to read most of the PICS available in the market. Although BACnet's official website offers a PICS template[9] for vendors, in practice most manufacturers use their own format. The variety of PICS difficults the process of information extraction. A fundamental constraint in our approach is that we identify different table templates by means of keywords. These keywords are standard BACnet objects and properties. Thus, our approach is unable to detect proprietary objects and properties from PICS.

Besides the PICS conditions (typos, tables, etc.), the file reading itself is a difficult task. PDF is optimized for information representation, however, extraction of information is hard. Our prototype reads an XML version of the PICS, which is generated by a third-party software tool. Nonetheless, the process is error prone and the format and contents of the original file might be

[9] http://www.bacnet.org/DL-Docs/135-2008-ANNEX-A-rev-2010-11-02.doc

distorted in the output file. Two of our PICS had to be manually fixed due to important missing information (BACnet object names) in the XML.

Automatic typos correction should be applied carefully. We ran the correction routine on those strings with a Levenshtein distance less or equal than 2. While a low threshold in the Levenshtein distance has a moderated effect in the PICS, a high threshold converts irrelevant words into BACnet objects and properties. If the contents of the PICS is severely damaged by the typos correction routine, the extracted rules suffer from omissions and mismatches.

While looking for the PICS of the 223 devices found in our network, we discovered that not all of them have one. According to the BACnet standard, *all* BACnet devices must have a PICS. After contacting the vendors, we realized that some devices are BACnet *aware* (they are capable to exchange BACnet messages) but not BACnet *compliant* (properly certified by the BACnet Testing Laboratories). Since BACnet *aware* devices are not obliged to satisfy any BACnet requirements, we found devices without *mandatory* properties. From a security point of view, networks composed of standard and non-standard devices, make it even harder to identify anomalous behavior.

The results during the detection phase are determined by the PICS interpretation process. A good interpretation leads to good rules and therefore relevant alerts. The fact that 89 omitted properties rules generated only 13 unique alarms in total, suggest that most of those omitted properties are not common in BACnet dialogues. We stress that this observation is only valid for BACnet networks with conditions similar to ours. Alerts because of proprietary elements are particularly important when they are not documented in the PICS. The discovery of undocumented capabilities become a valuable insight for BACnet operators as it was shown in [3].

8 Conclusions and Future Work

We present a data-mining approach to automatically interpret documents using network traffic. A direct application of our algorithm was used to generate rules for specification-based intrusion detection in the BACnet protocol. Our algorithm leverages on a modified version of the *Jaccard similarity* that enhances the interpretation of PICS based on the BACnet traffic analyzed during a training phase.

In our experiments at the University of Twente, the automatically generated rules show a high level of concordance with the PICS, however, there are some factors that can affect the results: the traffic in the training phase, the quality of the PICS (e.g. typos) and the correct interpretation of tables.

We tested our rules using 5 days of real BACnet traffic and discovered undocumented proprietary elements (objects and properties) about three types of devices.

The severity of attacks in BAS merits a careful security study. The fact that BAS protocols privilege functionality over security, increases the chance of attacks. Future BAS protocols should be designed with security as a core component. In the meantime, current BAS protocols should be methodically protected. The approach presented in this paper is a contribution in this direction.

Our current implementation[10] could be extended to take advantage of additional information provided in PICS. Annotations like "writable if *out-of-service* is TRUE" in Figure 3c, can be used to extract more complex rules. A different approach, possibly involving Natural Language Processing will be required.

References

[1] Microsoft exchange server 2007 rfc and support for standards. https://technet.microsoft.com/en-us/library/cc540463%28v=exchg.80%29.aspx. Accessed: 2016-03-22.

[2] R. Berthier and W. H. Sanders. Specification-based intrusion detection for advanced metering infrastructures. In *Dependable Computing (PRDC), 2011 IEEE 17th Pacific Rim International Symposium on*, pages 184–193. IEEE, 2011.

[3] M. Caselli, E. Zambon, J. Amann, R. Sommer, and F. Kargl. Specification mining for intrusion detection in networked control systems. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 791–806, Austin, TX, Aug. 2016. USENIX Association.

[4] A. Chaugule, Z. Xu, and S. Zhu. A specification based intrusion detection framework for mobile phones. In *Applied Cryptography and Network Security*, pages 19–37. Springer, 2011.

[5] S. Chen, R. Wang, X. Wang, and K. Zhang. Side-channel leaks in web applications: A reality today, a challenge tomorrow. In *2010 IEEE Symposium on Security and Privacy*, pages 191–206. IEEE, 2010.

[6] J. De Ruiter and E. Poll. Protocol state fuzzing of tls implementations. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 193–206, 2015.

[7] W. Granzer, F. Praus, and W. Kastner. Security in building automation systems. *IEEE Transactions on Industrial Electronics*, 57(11):3622–3630, 2010.

[8] S. N. Han, G. M. Lee, and N. Crespi. Semantic context-aware service composition for building automation system. *Industrial Informatics, IEEE Transactions on*, 10(1):752–761, 2014.

[9] D. Holmberg. Enemies at the gates. *BACnet Today, B24–B28*, 2003.

[10] D. G. Holmberg. Ashrae journal supplement-bacnet (r) today-secure messaging in bacnet (r). *ASHRAE Journal-American Society Heating Refrigerating Airconditioning Engineer*, 47(11):B23, 2005.

[11] P. Jaccard. The distribution of the flora in the alpine zone.1. *New Phytologist*, 11(2):37–50, 1912.

[10] Code available in https://github.com/SBIDS-BACnet/SBIDS-BACnet

[12] P. Jokar, H. Nicanfar, and V. Leung. Specification-based intrusion detection for home area networks in smart grids. In *Smart Grid Communications (SmartGridComm), 2011 IEEE International Conference on*, pages 208–213. IEEE, 2011.

[13] J. Kaur, J. Tonejc, S. Wendzel, and M. Meier. Securing bacnets pitfalls. In *ICT Systems Security and Privacy Protection*, pages 616–629. Springer, 2015.

[14] C. D. Kidd, R. Orr, G. D. Abowd, C. G. Atkeson, I. A. Essa, B. MacIntyre, E. Mynatt, T. E. Starner, and W. Newstetter. The aware home: A living laboratory for ubiquitous computing research. In *Cooperative buildings. Integrating information, organizations, and architecture*, pages 191–198. Springer, 1999.

[15] C. Ko, G. Fink, and K. Levitt. Automated detection of vulnerabilities in privileged programs by execution monitoring. In *Computer Security Applications Conference, 1994. Proceedings., 10th Annual*, pages 134–144. IEEE, 1994.

[16] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. In *Soviet physics doklady*, volume 10, page 707, 1966.

[17] J. Manyika, M. Chui, J. Bughin, R. Dobbs, P. Bisson, and A. Marrs. *Disruptive technologies: Advances that will transform life, business, and the global economy*, volume 12. McKinsey Global Institute New York, 2013.

[18] H. Merz, T. Hansemann, and C. Hübner. *Building Automation: Communication Systems with EIB/KNX, LON and BACnet*. Springer Science & Business Media, 2009.

[19] M. Newman. *BACnet The Global Standard for Building Automation and Control Networks*.

[20] C. Panos, C. Xenakis, P. Kotzias, and I. Stavrakakis. A specification-based intrusion detection engine for infrastructure-less networks. *Computer Communications*, 54:67–83, 2014.

[21] C. Ramakrishnan, A. Patnia, E. Hovy, and G. A. Burns. Layout-aware text extraction from full-text pdf of scientific articles. *Source Code for Biology and Medicine*, 7(1):1–10, 2012.

[22] T. Samuila, B. Orza, and A. Vlaicu. Open systems in building management: Lonworks vs. bacnet. *ACTA TECHNICA NAPOCENSIS: Electronics and Telecommunications*, 48(3):36–40, 2007.

[23] K. Scarfone and P. Mell. Guide to intrusion detection and prevention systems (idps). *NIST special publication*, 800(2007):94, 2007.

[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] N. Stakhanova, S. Basu, and J. Wong. On the symbiosis of specification-based and anomaly-based detection. *computers & security*, 29(2):253–268, 2010.

[26] W. Stallings. *Network security essentials: applications and standards*. Pearson Education India, 2007.

[27] M. Strassner and T. Schoch. Todays impact of ubiquitous computing on business processes. In *First international conference on pervasive computing*, volume 2002, pages 62–74, 2002.

[28] S. Wendzel, V. Zwanger, M. Meier, and S. Szlósarczyk. Envisioning smart building botnets. In *Sicherheit*, pages 319–329, 2014.