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

Herson Esquivel-Vargas
University of Twente
Enschede, The Netherlands
h.esquivelvargas@utwente.nl

Marco Caselli
Siemens AG
Munich, Germany
marco.caselli@siemens.com

Andreas Peter
University of Twente
Enschede, The Netherlands
a.peter@utwente.nl

ABSTRACT

Specification-based intrusion detection (SB-ID) is a suitable approach to monitor Building Automation Systems (BASs) because the correct and non-compromised functioning of the system is well understood. Its main drawback is that the creation of specifications often require human intervention. We present the first fully automated approach to deploy SB-ID at network level. We do so in the domain of BASs, specifically, the BACnet protocol (ISO 16484-5). In this protocol, properly certified devices are demanded to have technical documentation stating their capabilities. We leverage on those documents to create specifications that represent the expected behavior of each device in the network. *Automated* specification extraction is crucial to effectively apply SB-ID in volatile environments such as BACnet networks, where new devices are often added, removed, or replaced. In our experiments, the proposed algorithm creates specifications with both precision and recall above 99.5%. Finally, we evaluate the capabilities of our detection approach using two months (80GB) of BACnet traffic from a real BAS. Additionally, we use synthetic traffic to demonstrate attack detection in a controlled environment. We show that our approach not only contributes to the practical feasibility of SB-ID in BASs, but also detects stealthy and dangerous attacks.

CCS CONCEPTS

• **Security and privacy** → **Intrusion detection systems**; *Network security*;

KEYWORDS

BACnet, building automation systems security, specification-based intrusion detection, automatic specification extraction

1 INTRODUCTION

Building Automation Systems (BASs) monitor and control physical processes in modern buildings. Typical deployments automate services such as heating, ventilation, and air conditioning, but can be extended to lights, alarms, elevators, physical access control, and many others. Unlike standard IT systems, BASs are capable of influencing the physical environment. The intrinsic capabilities of BASs open new possibilities for cyber-attacks and new challenges for cyber-defenses. On the offensive side, it has been shown how smart lights can be misused to trigger seizures in people suffering from photosensitive epilepsy or to exfiltrate data from highly secure office buildings [25]. In another example from 2014, the heating system is considered the entry point of the *Target* hack where the attackers got access to personal information about 70 million customers and 40 million credit card numbers [23]. In one of the most recent cases from 2017, attackers remotely locked all the doors in a hotel until a ransom was paid [5].

Protecting BASs has been proven to be hard since most systems have no built-in security features. In this context, attack detection becomes an essential approach to secure BASs. Diverse Intrusion Detection Systems (IDSs) have been proposed in literature [2]. A particular one compares rules that encode *expected behavior* with the *actual behavior* of the system. This is called specification-based intrusion detection (SB-ID) [26]. SB-ID has been successfully applied in critical infrastructures where the correct functioning of the system has been summarized as rules [4, 18]. The main drawback of SB-ID is that rules (a.k.a. specifications) often involve intensive manual labor.

Attacks on BASs manifest as unauthorized commands to vulnerable devices. Looking for such commands at the host level is unfeasible due to the heterogeneity of devices in BAS networks, running mostly closed proprietary software. An alternative is to monitor the network traffic, closely looking at the source and destination of BAS commands. This approach is passive, easy to deploy, and scalable.

In this work we develop the first fully automated approach to deploy SB-ID at the network level. We focus on each device's documented capabilities and monitor the network looking for inconsistencies in the observed behavior. Incoming and outgoing traffic is analyzed. Anomalies on incoming traffic (e.g., asking for capabilities that a device does not have) could be a sign of an ongoing attack, whereas anomalies on outgoing traffic (e.g., showing capabilities that the device is not supposed to have) could be a sign of an already compromised device.

Our work is inspired by the research presented in [6] but differs in several fundamental aspects. The core difference regards the automated interpretation of each device's technical documentation: [6] essentially searches for specific occurrences of text strings in a predetermined order. Because of the very heterogeneous documentation, this approach gives no search results in many documents, implying no information extraction. In documents that follow the predetermined structure and format, the approach extracts the right information but is unable to interpret it correctly (e.g., does the occurrence of a capability name mean that the capability is supposed to be implemented in a certain device or not?). This ambiguity remains unsolved in [6] and produces a severe amount of manual interpretation. Our approach, on the other hand, solves both described limitations of [6] by interpreting the documentation using the available network traffic to map monitored (and hence implemented) information to parts of the documents which, in a second step, allows us to interpret the complete documents. Our experimental results show that with this approach, we can correctly and non-ambiguously interpret documents with different formats in an automated way. Further details on the differences of our approach with [6], such as on the design, implementation, and evaluation, can be found in Sect. 9.

Our approach is applied on devices that communicate using the BACnet protocol (ISO 16484-5). We leverage on publicly available technical documentation about BACnet devices to *automatically* extract expected behavior rules. The process is done in two steps. First, a *subset* of the devices' capabilities is observed in the network traffic. Second, based on the previous observations, our algorithm extracts *all* the devices' capabilities from the technical documents.

Our results show that the proposed algorithm can extract device behavior rules with high precision and recall. Furthermore, the extracted specifications allow the detection of a variety of dangerous attacks, some of them consisting of a single network packet.

In summary, we propose a novel approach to automatically deploy SB-ID in BACnet networks. Moreover, we identified a subset of attacks that can be detected with our specifications. Finally, we implemented a prototype and evaluated our approach with real and synthetic BACnet traffic.

Overall, we consider this work as a step forward in the challenging task of securing BASs. Hereafter the paper is organized as follows. Section 2 provides background information regarding the BACnet protocol. Section 3 and Sect. 4 define our system and threat model, respectively. The design of our solution is presented in Sect. 5. The implementation details are described in Sect. 6, followed by the evaluation in Sect. 7. We discuss our results in Sect. 8. A summary of related work is presented in Sect. 9. Final remarks and conclusions are presented in Sect. 10.

2 BACNET PROTOCOL OVERVIEW

BACnet (ISO 16484-5) is a vendor-neutral communications protocol for Building Automation and Control Networks [1]. BACnet dictates a set of rules that governs how devices should communicate. Thanks to its standardization, BACnet devices can interoperate regardless of the manufacturer. More than an application layer protocol, BACnet is a four-layers protocol stack. The physical and data link layers allow the use of different protocols in order to fit diverse environments. The network layer allows the interconnection of two or more BACnet networks. The application layer is in charge of the actual data exchange among BACnet devices. BACnet *objects*, *properties* and *services* play a key role in the application layer.

2.1 BACnet Objects and Properties

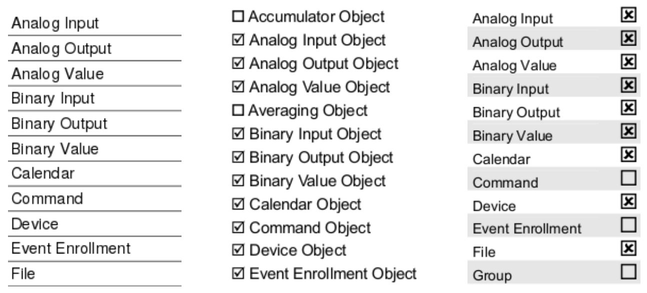
BACnet organizes data using an object-oriented approach. BACnet objects are comprised of a subset of BACnet properties that store data. The standard defines 60 object types to satisfy the most frequent needs in BASs. The BACnet standard dictates which properties are optional, required and writable for each object. Moreover, the standard declares the possibility to implement proprietary —vendor specific— objects and properties in BACnet devices. There is only one mandatory object for all BACnet devices: the *device* object. It contains several properties such as *vendor-identifier*, *model-name*, *system-status*, among others. Since the applicability of standard objects vary depending on the purpose of the device, it is not required to implement all 60 standard objects.

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.

2.2 BACnet Services

BACnet services refer to specific actions that devices can handle. *Read Property*, *Write Property*, *Reinitialize Device*, and *Atomic Write File* are some examples of BACnet services. BACnet defines 41 standard services. Depending on the device's purpose, only a subset of them has to be implemented.

Services typically involve two roles, clients that send requests, and servers that reply. In order to clarify which services can be sent or received by different devices, the BACnet standard defines the BACnet Interoperability Building Blocks (BIBBs). BIBBs are often mentioned as acronyms with three components: *type*, *task*, and *capability*. The *type* refers to 5 broad BIBB categories. The *task* specifies the purpose of the BIBB. The *capability* states whether the device acts as a client or a server, denoted as “A” or “B”, respectively. *DS-RP-A* is a BIBB example that stands for Data Sharing (type), Read Property (task), and client role (“A” capability). Devices implementing DS-RP-A can send Read Property queries to other devices. A device with the complementary BIBB, namely DS-RP-*B*, would be able to reply such a query.



Analog Input
Analog Output
Analog Value
Binary Input
Binary Output
Binary Value
Calendar
Command
Device
Event Enrollment
File

(a)

☐ Accumulator Object
☑ Analog Input Object
☑ Analog Output Object
☑ Analog Value Object
☐ Averaging Object
☑ Binary Input Object
☑ Binary Output Object
☑ Binary Value Object
☑ Calendar Object
☑ Command Object
☑ Device Object
☑ Event Enrollment Object

(b)

Analog Input	☒
Analog Output	☒
Analog Value	☒
Binary Input	☒
Binary Output	☒
Binary Value	☒
Calendar	☒
Command	☐
Device	☒
Event Enrollment	☐
File	☒
Group	☐

(c)

Figure 1: Excerpts from different PICS stating which object types are implemented. PICS (a) lists only the implemented object types, whereas PICSs (b) and (c) use specific Unicode characters to denote which of them are implemented.

There are 146 BIBBs in the 2016 version of the standard, some of which are composed of multiples services at once.

2.3 BACnet PICS

BACnet networks are comprised of devices with very different functions. Sensors, actuators, operator workstations, controllers and others, are device profiles typically found in BACnet deployments. Vendors must issue a document called *Protocol Implementation Conformance Statement* (PICS), that specifies which BACnet objects, properties and BIBBs are actually implemented per device.

The BACnet standard is strict about the contents that PICSs must have, but lax in the document layout that PICSs should use. Object listings from different PICSs are shown in Fig. 1. All of them describe the same kind of information using distinct table layouts. BACnet properties are also described in PICSs, however, more complex table layouts are used because they have additional features such as obligatoriness, writability, conditional existence, among others.

3 SYSTEM MODEL

In this paper, we consider a typical BACnet network consisting of sensors and actuators, which are managed by BACnet controllers and are interconnected by BACnet routers (see Fig. 2). The operator workstation is occasionally on-line and has a special BACnet software to perform configuration and process monitoring tasks. Operator workstations, routers and controllers are typically interconnected by the corporate LAN (TCP/IP). They are known as BACnet/IP devices. Sensors and actuators, on the other hand, are usually connected to their controllers using different protocols such as EIA-232. Still, their communication can be observed on the LAN because controllers forward messages to/from them. The theoretical limit of BACnet devices in the network is about 2^{22}. All the components in Fig. 2 can potentially communicate with each other through BACnet messages, using BACnet objects, properties, and services.

We assume the availability of PICS for all the devices in the network. This is a realistic assumption since the BACnet standard states that *"all devices conforming to the BACnet protocol shall have a Protocol Implementation Conformance Statement (PICS) that identifies all of the portions of BACnet that are implemented"* [1]. Moreover, PICS must contain at least the supported object types, properties, writable properties and BIBBs. PICS are published as PDF files in the BACnet International website [3].

Standard network monitoring tools can be used to observe BACnet traffic. We consider BACnet networks with no authentication and no encryption. This is due to the fact that the *Network Security* clause of the standard is optional and uncommon in BACnet devices [20]. We assume that it is possible to collect BACnet traffic prior to the deployment of the IDS. Traffic collection is typically granted to authorized parties in the organization such as to network security staff. The collected traffic must contain information about all the devices in the network under normal operative conditions.

4 THREAT MODEL

Attackers are assumed to be capable of sending messages to BACnet devices and getting replies from them. Logical access to BACnet devices can be obtained locally or remotely. Since BACnet networks typically use the corporate LAN, attackers can interact with BACnet devices within the network. Moreover, search engines like Shodan allow attackers to find Internet exposed BACnet devices in remote locations [27].

The adversary is also capable of manipulating the devices' firmware. Firmware tampering can occur, for example, by compromising the operator's workstation, gaining physical access to the control room where BACnet devices are located, or intercepting devices during shipping [9]. A compromised BACnet device is also considered an attacker in the local network.

We assume that the network monitor cannot be compromised. This is a common assumption since network security devices are often in a different address space and therefore, unreachable from the monitored network where the attacker resides. Similarly, we assume that the data source of the IDS (e.g., switch with a mirroring port) is not compromised and delivers data without packet loss.

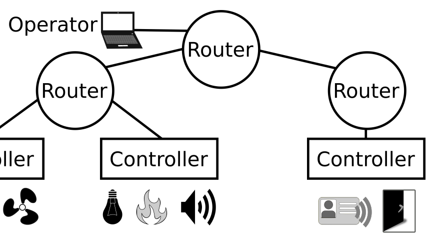


Figure 2: Schematic overview of BACnet network components. Controllers manage sensors and actuators. Internetwork communication relies on BACnet routers.

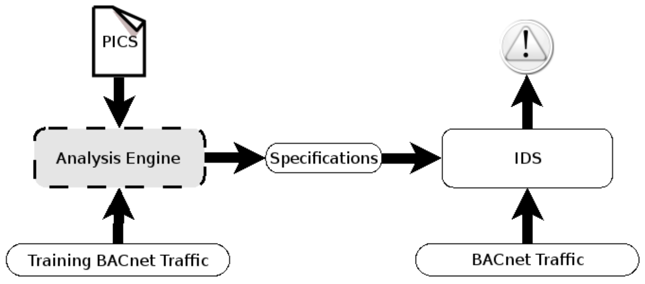


Figure 3: Overall system architecture.

BACnet Attacks. The threats described before materialize in specific attacks such as backdoors, active device fingerprinting, and denial of service (DoS).

A firmware backdoor could allow the attacker to establish a hidden communication channel with the affected device. The possibilities of the attacker from this point on are diverse: lateral movement to other BAS devices or networks, divert BAS processes (e.g., changing set points), sniff network traffic, data exfiltration, etc. There are precedents of attacks on IT infrastructure via compromised BASs [16, 23]. Such backdoors could be implemented as proprietary or standard BACnet elements in order to disguise the traffic as benign.

On the other hand, during active device fingerprinting the attacker sends a predefined set of messages to the targeted device. The device's response is used to deduce specific details such as manufacturer, model and firmware version. This information is useful to find exploits and launch further attacks.

As our last example, valid services such as *Reinitialize Device* or *Device Communication Control* can be used in DoS attacks. The former causes devices to reboot while the latter instructs them to stop initiating and optionally stop responding to incoming messages [1].

Rather than extensively listing BACnet attacks, our goal is to illustrate diverse attack instances to realize the potential impact on BASs. We implement and provide more details on these three attacks in Sect. 7.4.

5 DESIGN

We start with a system overview in Sect. 5.1. Details of our approach to automatically extract specifications are presented in Sect. 5.2. Finally, Sect. 5.3 briefly describes our attack detection process.

5.1 Overview

Our goal is to develop an approach to automatically generate rules to be used in specification-based intrusion detection. The rules we extract in this paper represent valid device behavior in BACnet networks. Once the rules have been extracted, they are loaded into the IDS. Finally, the IDS monitors the network looking for rule violations in the traffic. Figure 3 shows the overall architecture of our system.

Looking closer at the specification extraction process, the *Analysis Engine* takes two inputs: (1) training BACnet traffic; and (2) PICS files. First, the *Analysis Engine* reads training traffic from the network in order to identify devices, group them by brand and model, and deduce which are *some* of their capabilities. During this process our approach should not interact with BACnet devices in order to avoid accidental disruption of the network. We refer to this property of our system as *passiveness.*

Second, using the information learned from the training traffic, our algorithm interprets PICSs and creates sets of capabilities for each device in the network. Automation of the PICS interpretation process is crucial because BACnet networks can be comprised of thousands of devices, each of them potentially described by its own PICS. We refer to this property of our system as *scalability.*

5.2 Architecture

Specification extraction is performed by the *Analysis Engine*. A more detailed view of the *Analysis Engine* is shown in Fig. 4. Its main functions are:

Device Fingerprinting. The *input* of the fingerprinting routine is the training BACnet traffic. The *output* is a list of tuples `[device_id, brand, model]`. The `device_id` is unique throughout the BACnet network. On the other hand, many devices can share the same brand and model.

Our system fingerprints BACnet devices applying two techniques described in [6]. The first technique is *"Device Object Analysis"*, which focuses on network packets containing properties of the *device* object. Those packets might contain all the required information at the same time. The second technique is *"BACnet Address Linking"*. In this case, we link one packet containing the `device_id` and the BACnet address (both of which are unique in the network), with another packet containing the BACnet address plus additional information like the brand and model. In this way, the BACnet address links the `device_id` with its corresponding brand and model.

It is worth noting that our system cannot extract specifications for devices not fully fingerprinted (i.e. incomplete tuples). No specifications for a subset of devices mean that the IDS will not be able to monitor malicious behavior for them. This condition imposes a security gap that BACnet administrators should be warned about. However, in practice devices can be successfully fingerprinted after analyzing *enough* training traffic. The example in Fig. 4 shows five devices successfully fingerprinted, depicted as geometric figures.

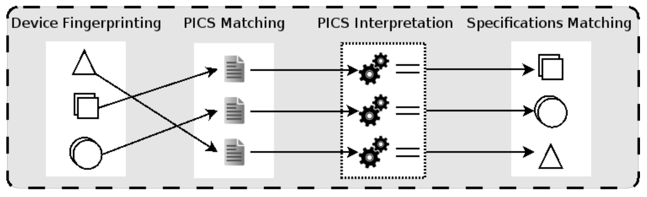


Figure 4: Analysis Engine components.

PICS Matching. The *inputs* of the PICS matching routine are the list of fingerprinted devices and the set of PICS that describe all the devices in the network. The *output* is a list of extended tuples `[device_id, brand, model, PICS_file]`.

PICS Interpretation. The *inputs* of the PICS interpretation process are the list of extended tuples and the training BACnet traffic. For every PICS file there are four *outputs*: (1) the set of implemented object types; (2) the set of implemented properties for each object type; (3) the set of writable properties for each object type; and (4) the set of implemented BIBBs.

The interpretation routine is the core of our specification extraction approach. In what follows, we will broadly refer to BACnet objects, properties, writable properties, and BIBBs as BACnet *elements*.

The first step is to classify the training traffic by device type. All the traffic sent by devices of the same brand and model is analyzed individually. Looking for confirmations in the traffic (e.g., read acknowledgments), the system identifies which BACnet elements are present for that type of devices. Similarly, looking for errors in the traffic the system deduces which elements are absent. We refer to the sets of present and absent elements as $\mathscr{P}$ and $\mathscr{A}$, respectively. The sets $\mathscr{P}$ and $\mathscr{A}$ are observations taken from the training traffic, which is collected during a limited time span. Hence, it is expected to get incomplete sets of implemented and non-implemented elements.

The interpretation process locates the complete list BACnet elements in the PICS. Since the list of standard BACnet elements is known in advance, it is feasible to pinpoint the elements in the document. Figure 5 shows four examples of typical table layouts that describe BACnet elements in PICS. Once the elements' location has been identified, our approach groups the elements using different criteria. Figures 5a, 5c, and 5d suggest a column-wise grouping. Similarly, Fig. 5b and Fig. 5c suggest a row-based grouping. Elements within the same cell (Fig. 5b) are also grouped together. Finally, for any of the layouts, we create groups of elements with similar nearby text. For example, elements next to a symbol (e.g., ☑), word (e.g., *"Implemented"*) or phrase (e.g., *"Supported Object Types"*). In what follows we will refer to the groups extracted from the PICS as *reference sets*. Several reference sets are extracted for each PICS, but only one of them will contain the set of *all* implemented elements.

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

Our way to find out which of the *reference sets* has all the implemented elements, is by comparing with our observations $\mathscr{P}$ and $\mathscr{A}$. The Jaccard coefficient is a simple yet effective scoring mechanism to quantify the similarity between two sets [11]. Equation (1) defines the Jaccard similarity with arguments $\mathscr{P}$ and R_i, where R_i represents the i^{th} *reference set* extracted from the PICS. Using the Jaccard coefficient it is possible to rank the reference sets according to their similarity

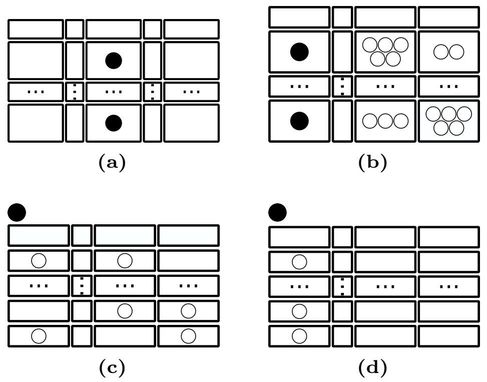


Figure 5: Table layouts commonly used in PICS. Dark and light circles represent BACnet objects and properties, respectively. Empty cells might contain arbitrary text.

to $\mathscr{P}$. A *reference set* similar enough to $\mathscr{P}$ is expected to be the set of all the implemented elements, however, this might not always be the case specially when $\mathscr{P}$ has few elements. To solve this problem we also use the set of absent elements $\mathscr{A}$.

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

Equation (2) shows our modified version of the Jaccard similarity that we call $Jaccard'$. $Jaccard'$ adds new terms to the $Jaccard$ similarity with the following effect:

(1) If R_i is a suitable *reference set*, it should contain most (ideally all) of the objects in $\mathscr{P}$, which means that $|\mathscr{P} \setminus R_i|$ must be equal or close to zero.
(2) If R_i is a suitable *reference set*, it should contain few (ideally none) of the elements in $\mathscr{A}$, which means that $|\mathscr{A} \cap R_i|$ 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_i.

We evaluate the effectiveness of this modified metric in Sect. 7.2.

Specifications Matching. The *inputs* for this routine are the list of extended tuples and the set of valid BACnet elements (objects, properties, writable properties, and BIBBs) for each PICS. The *output* are specifications of the form `[device_id, valid_BACnet_elements]`.

Since all the fingerprinted devices have already been matched with their corresponding PICS, it is straightforward to match each device with their specifications.

5.3 Attack Detection

As shown in Fig. 3, the specifications generated by the *Analysis Engine* are loaded into the IDS. The IDS verifies that audited devices are concordant with their PICS in terms of valid objects, properties and BIBBs. Deviations from the expected behavior are flagged as security events. It is worth noting that PICS violations do not imply violation of the standard. In fact, all the attacks we consider are comprised of completely valid BACnet packets.

Verification of BACnet objects and properties is unambiguous because PICS explicitly mention which of them are implemented or not. BIBBs verification is harder because the network communication is based on BACnet services that can be part of many BIBBs. The NIDS verifies that the observed service is part of at least one of the supported BIBBs. An alarm is raised otherwise.

6 IMPLEMENTATION

We implement our prototype using third-party software tools and custom scripts. The scripts are open-source and publicly available.[1]

Device fingerprinting is implemented using Bro [29]. Bro's output are log files in plain text. These logs are stored in an SQL database to ensure easy access from other modules. We use MySQL as our DBMS. Nonetheless, the database scripts are written using standard SQL statements that could be handled by other DBMSs as well.

The **PICS matching** process is done by Apache Solr [8]. This tool consists of a web application that ranks documents given some keywords. The behavior of Apache Solr is similar to Internet search engines but on a predefined set of documents. In our particular setting, the documents are PICSs and the keywords are the brand and model of all the fingerprinted devices. The API provided by Solr allows us to automate the queries. In this way, we can efficiently match all the devices with their corresponding PICS.

Prior to the **PICS interpretation** routine, we use Bro to identify implemented and non-implemented BACnet elements in the training traffic. Since all the devices of the same brand and model are described by the same PICS, we aggregate our observations in the database. Aggregated information per device type allows us to create the sets $\mathscr{P}$ and $\mathscr{A}$, required to use our $Jaccard'$ metric as defined in Equation (2).

The *reference sets* are extracted from PICSs in Portable Document Format. Layout-aware text extraction from PDF files is a well-known problem [24]. PDF is used for standardized information presentation rather than information extraction. We circumvent the problem by creating XML versions of the PICS using *Adobe Acrobat Pro*. Nonetheless, the process is error prone and the format and contents of the original file might be distorted in the output file. Moreover, there are typos and abbreviations in the PICS, that make it harder to pinpoint BACnet elements in the PICS.

We implemented a typo correction function based on the Levenshtein distance [17]. This approach measures how many deletions, additions or substitutions are required to convert one string into another. Since we know in advance the list of standard BACnet elements, we compute the Levenshtein distance between each word in the PICS with all the elements in the list. If the Levenshtein distance is less than a predefined threshold, we fix the string read from the PICS, else it is left unchanged. Our empirical studies show that one-fourth of the string length is a suitable fixing threshold for BACnet property names. For BACnet object names we use a fixed threshold set at 3.

Finally, a Python script reads from the database the network traffic observations ($\mathscr{P}$ and $\mathscr{A}$) and computes the $Jaccard'$ metric with all the reference sets extracted from the PICS. The reference set scoring the highest $Jaccard'$ value is considered the set of implemented elements. From this set, we create the specifications that are going to be loaded into the IDS. The specifications are generated in Bro Scripting Language. Nonetheless, our implementation can be adapted to support other IDSs as well.

7 EVALUATION

We present an evaluation of our prototype, starting with intermediate results gotten during the training traffic analysis (Sect. 7.1). The automatically extracted specifications were evaluated at two levels. First, we measured the effectiveness of our approach to create specifications concordant with the PICS (Sect. 7.2). Second, we applied the specifications on BACnet traffic to analyze their capability to detect attacks such as those described in our Threat Model. We analyzed traffic from a real BACnet network in an international University campus (Sect. 7.3). We conclude our evaluation analyzing traffic from a simple testbed where we execute our own BACnet attacks (Sect. 7.4).

7.1 Training Traffic Analysis

We trained our system with real BACnet traffic. The network is comprised of 646 BACnet devices in 8 buildings. Six devices whose PICSs are not available were not considered in our evaluation. Our local PICS repository consists of 10 PDF files that describe the remaining 640 devices.

The BACnet network has its own isolated VLAN which is not visible from the corporate LAN. However, our evaluation is based on traffic collected from one of the core switches. We used 4.5GB (4 days) of BACnet traffic for the training phase.

All the fingerprinted devices were identified using only $3 \cdot 10^6$ BACnet packets of the training data, which in our network represents roughly six hours (300MB) of traffic. BACnet devices were automatically matched to one of the PICS available in the repository. A manual verification confirmed that BACnet compliant devices were correctly matched to their corresponding PICS. A breakdown of our results is shown in Table 1.

As part of the *PICS Interpretation* process, it is crucial to identify the set of implemented (i.e., present) and non-implemented (i.e., absent) object types, properties and services in the network. There are 60 standard object types, however, on average, our devices implement only 15.6. Considering our 640 BACnet devices we expected to observe around 9,984 object types ($15.6 \cdot 640$). We were able to discover 9,804

[1] https://github.com/SBIDS-BACnet/SBIDS-BACnet

Table 1: BACnet devices grouped by PICS

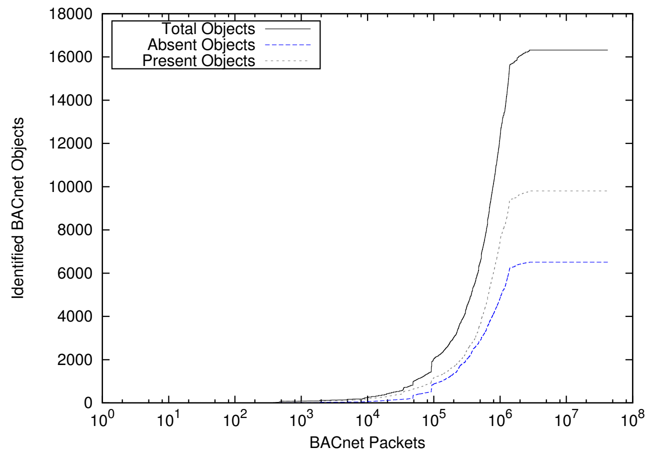
# of devices	Device Manufacturer	Device Model	PICS File
1	DEOS	COSMOS Open	PICS-1
3	Delta Controls	eBCON	PICS-2
5	Kieback&Peter	DDC4400	PICS-3
1	Mitsubishi	BM ADAPTER	PICS-4
25	Priva	Compri HX 3	
7	Priva	Compri HX	
87	Priva	Compri HX 8E	PICS-5
36	Priva	Compri HX 4	
13	Priva	Compri HX 6E	
403	Priva	Comforte CX	
16	Priva	HX 80E	PICS-6
31	Priva	Blue ID S10	PICS-7
5	Priva	Blue ID C4	PICS-8
1	Siemens	PXC00-U	
3	Siemens	PXC128-U	PICS-9
1	Siemens	PXC64-U	
2	Siemens	PXG80-N	PICS-10
3	LOYTEC	LVIS-3ME15-G2	N/A
3	Siemens	PXR11	N/A

implemented object types and 6,512 non-implemented object types. Similarly, we discovered 41,704 BACnet properties. The vast majority of properties (40,467) were implemented and only 1,237 were not implemented. Finally, we found 24,882 services of which 16,713 were implemented and the remaining 8,169 were not. Figure 6 shows the discovery rate for BACnet objects, properties and services.

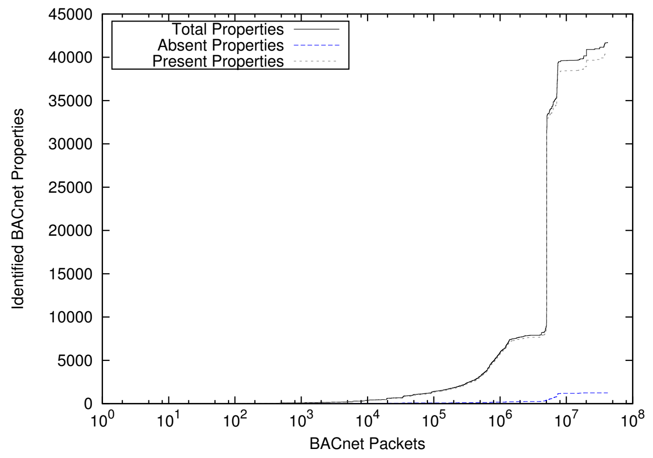
7.2 PICS Interpretation

For all the PICS shown in Table 1, we ran our algorithm to extract implemented objects, implemented properties, writable properties, and BIBBs. We present our results in terms of *precision* and *recall*. *Precision* refers to the fraction of extracted specifications that are concordant with the PICS. *Recall* is the fraction of specifications that were extracted from the PICS, considering the total amount of specifications that can be extracted.

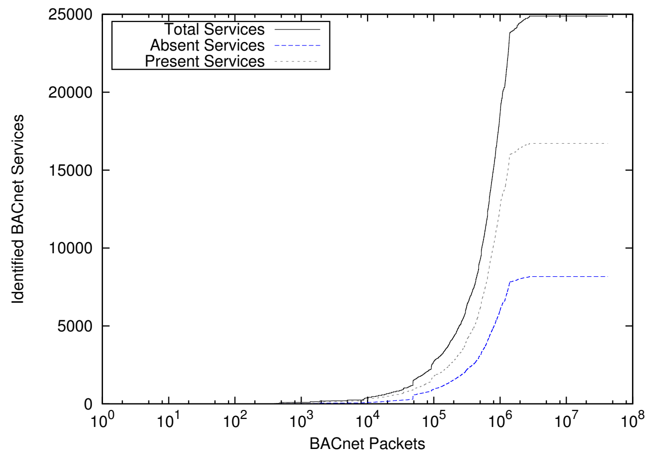
Table 2 shows the results gotten for three different types of specifications. We extracted a total of 136 specifications about implemented BACnet objects, 283 regarding implemented BIBBs and 57 about writable properties. Writable property specifications were extracted from only one of the PICS in our repository. This is because although we observed write requests on different devices, all of them were reduced to a single PICS (*PICS-5*). Therefore, our prototype used only this PICS to extract specifications about writable properties. We achieved 100% precision and recall for those three types of specifications.



(a)



(b)



(c)

Figure 6: Discovery rate for BACnet objects (a), properties (b), and services (c) during the training phase.

Table 2: Specification extraction performance of implemented objects, BIBBs, and writable properties.

PICS File	Object Specs.	BIBB Specs.	WProp. Specs.	Precision and Recall
PICS-1	17	38	-	100%
PICS-2	19	43	-	100%
PICS-3	22	49	-	100%
PICS-4	9	10	-	100%
PICS-5	16	27	57	100%
PICS-6	2	12	-	100%
PICS-7	16	31	-	100%
PICS-8	16	31	-	100%
PICS-9	18	34	-	100%
PICS-10	1	8	-	100%
TOTAL	**136**	**283**	**57**	**100%**

Results about the extraction of BACnet properties are shown in Table 3. Our algorithm extracted 2,064 specifications with a precision of 99.85%. The reason for precision loss was the misinterpretation of surrounding text strings as BACnet properties. Moreover, we got a recall of 99.57% due to abbreviations in property names that the typo correction routine was not able to fix. In Sect. 8 we discuss the implications of precision and recall below 100% and what can be done to mitigate the problem.

Table 3: Specification extraction performance of implemented properties.

PICS File	Property Specs.	Precision	Recall
PICS-1	244	100%	100%
PICS-2	224	99.11%	96.10%
PICS-3	520	100%	100%
PICS-4	46	100%	100%
PICS-5	237	100%	100%
PICS-6	41	100%	100%
PICS-7	242	99.59%	100%
PICS-8	244	100%	100%
PICS-9	259	100%	100%
PICS-10	7	100%	100%
TOTAL	**2,064**	**99.85%**	**99.57%**

It is worth noting that some PICS show only optional properties instead of all the implemented properties. Since our evaluation is based in the contents of the PICS, implicit properties were not taken into account.

7.3 Intrusion Detection on Real Traffic

The automatically extracted specifications were used to analyze BACnet traffic in our network for two months, which in our setting consists of roughly 80GB of data. The IDS raised alarms due to violations in our specifications about implemented objects and properties (see Table 4). We discovered 25 BACnet objects in 4 different device models. Two of the undocumented objects are standard objects, namely, *analog-output* and *command*. The remaining 23 discovered objects are proprietary. Similarly, we discovered 103 undocumented BACnet properties. Seven properties are standard (*time-of-device-restart*, *last-notify-record*, *feedback-value*, and four times the *profile-name* property) and 96 are proprietary. On average, 0.5% of the traffic makes reference to undocumented BACnet elements. As we discussed in Sect. 4, such behavior could be a sign of a backdoored device with grave consequences for the entire BAS. According to the BACnet operators such behavior is normal in our network and does not constitute an actual attack.

The IDS also raised alerts due to devices being queried about capabilities they are not supposed to have according to their PICS. In our setting such queries occur in less than 1.5% of the traffic mainly due to: (1) BACnet software used by the operators that tries to read as much information as possible from the devices; and (2) logging servers that regularly poll, via *read requests*, a predefined set of BACnet elements. In both cases, attempts to read non-implemented objects and properties cause all the alerts. We omit further details since the alerts we got are fully dependent on the specific configuration of our network. Even though we did not find ongoing attacks, we want to stress that this behavior should call the attention of the operators since it resembles popular reconnaissance tools that might be used during early stages of network attacks.

The IDS did not trigger any alerts regarding *writable properties* nor *BIBB* specifications.

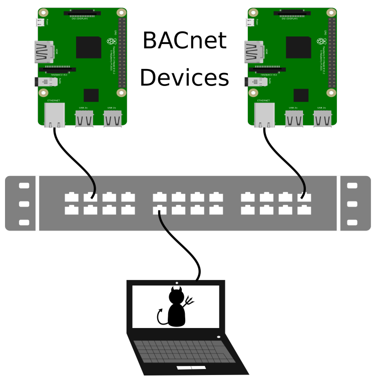


Figure 7: Our BACnet testbed. The two Raspberry Pi computers at the top simulate the *Priva Blue ID S10* and the *Mitsubishi BM ADAPTER*, respectively. The computer at the bottom represents the attacker.

Table 4: Undocumented BACnet objects and properties. Our findings are highlighted in bold font.

Device Model	Object	Property
COSMOS Open	8 (device)	**203 (time-of-device-restart)**
eBCON	0 (analog-input)	**1033, 1039, 1076**
	2 (analog-value)	**1067**
	5 (binary-value)	**1033**
	8 (device)	**1033, 1040, 1074, 1077, 1078, 1079, 1089, 1090, 1100, 1101, 1102, 1115, 1116, 1117, 1118, 1119, 1120, 1121, 1122, 1123, 1125, 1137, 1138, 1139, 1140, 1141, 1142, 1144, 1145, 1146, 1147, 1148, 1151, 1156, 1158, 1160, 1162, 1163, 1164, 1165, 1166, 1167, 1168, 1169, 1204, 1205, 1206, 1207, 1209, 1210, 1212, 1213, 1214, 1215, 1216, 1219**
	9 (event-enrollment)	**1033, 1037, 1038, 1045, 1192, 1198, 1199, 1213, 1216**
	15 (notification-class)	**1033, 1034, 1040, 1074**
	16 (program)	**1033**
	20 (trend-log)	**173 (last-notify-record), 1135**
	142, 149, 152, 176, 177, 178, 181, 183, 270, 272, 278, 284, 297, 298, 311	
Compri HX 8E	17 (schedule)	**168 (profile-name)**
	20 (trend-log)	**168 (profile-name)**
Comforte CX	**1 (analog-output)**	
Blue ID S10	17 (schedule)	**168 (profile-name)**
Blue ID C4	17 (schedule)	**168 (profile-name)**
PXC00-U	8 (device)	**3028, 3034**
	200, 201, 202, 204, 207, 208, 214, 215	
PXC128-U	1 (analog-output)	**40 (feedback-value)**
	4 (binary-output)	**3067**
	5 (binary-value)	**3067**
	7 (command)	
	8 (device)	**3019, 3028, 3034, 3061, 3062, 3063, 3064**
	20 (trend-log)	**3019**
PXC64-U	4 (binary-output)	**3067**
	8 (device)	**3028, 3034, 3061, 3062, 3063, 3064**
	20 (trend-log)	**3019**

7.4 Intrusion Detection on Synthetic Traffic

In addition to the real BACnet network, we set up a simple testbed to execute BACnet attacks in a controlled environment. In this section we elaborate and provide implementation details on three concrete BACnet attacks, namely, a BACnet backdoor, active device fingerprinting, and denial of service.

Our testbed consists of two BACnet/IP devices interconnected by a switch, as shown in Fig 7. We also added a laptop running Kali Linux 2017.1 to simulate the attacker's role. Both BACnet devices are implemented using Raspberry Pi computers running Raspbian OS. The BACnet software running on top of the OS is BACnet Stack 0.8.4 [14]. In order to reuse the specifications previously generated, we configured the BACnet Stack demo server with similar features (i.e., objects, properties, and services) as the *Priva Blue ID S10* and the *Mitsubishi BM ADAPTER* that we have in our real network. The switch is an HP ProCurve 2610-24, in which we configured a mirroring port to collect all the traffic.

Backdoor. Our first attack consists of a BACnet device whose firmware has been tampered with by a malicious actor. We added a backdoor in our BACnet device simulating the Priva controller. The backdoor is located in a standard

BACnet object (*CharacterString Value*) implementing all the mandatory properties (e.g., *object-identifier, present-value, status-flags*, etc.) and common optional properties (e.g., *description, out-of-service, event-state*, etc.). The backdoor functionality is implemented by means of a writable property that receives commands from the attacker, and a readable property which holds the output of the attacker's command. Concretely, the writable property is *present-value* and the readable property is *description*. In this way, the attacker sends Linux commands (e.g., *ls, cat, rm*, etc.) and then reads the output, all using syntactically valid BACnet objects, properties and services. Since the *CharacterString Value* object is not part of the device's PICS, our specifications spotted the violation even though we sent only one packet consisting of a write request to the *present-value* property. The same functionality could have been implemented in proprietary BACnet objects as we found many in our real BACnet network (Sect. 7.3).

Active Device Fingerprinting. The second experiment consists of an attacker running reconnaissance tools such as `nmap` on our two BACnet devices. Nmap [19] is a popular port scanner that can be extended via nmap scripts. There is a script specifically for BACnet that tries to fingerprint devices executing queries on multiple properties of the *device* object [10]. The requested properties are *vendor-identifier, vendor-name, object-identifier, firmware-revision, application-software-version, object-name, model-name, description* and *location*. From the attacker's laptop we ran the nmap script directed to the simulated Priva device and the simulated Mitsubishi device. In the first case we did not get any alerts since all 9 properties are implemented in the Priva device according to the PICS. On the other hand, the specifications of the Mitsubishi device triggered two alerts since the *description* and *location* properties are not implemented. This example shows the main drawback of our approach. Attacks that leverage on the expected behavior of devices cannot be detected by our specifications. At the same time, we show that we can detect stealthy attacks comprised of as few as a single network packet, as long as it represents a violation to the device's PICS.

Denial of Service. A third attack was launched against the simulated Mitsubishi device. We use the simulated Priva device as a pivot to reach the target. Taking advantage of the backdoor in the simulated Priva device, we downloaded a malicious binary file that sends reinitialize requests to the target. The binary file is based on the demo `bacrd` command in BACnet-Stack [14].

Besides the alerts generated because of the backdoor, we got two additional alerts. The first alert due to the fact that the pivot device is not supposed to send *Reinitialize Device* requests. The second alert because the target device is not supposed to receive such command. Those alerts demonstrate the usefulness of our BIBB specifications in a realistic attack scenario.

8 DISCUSSION

Training Phase. Our approach leverages on the availability of training BACnet traffic and PICSs. The BACnet traffic required for training purposes must reveal as many objects, properties, and services 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 write requests in our training traffic, all of them were directed to devices described by the same PICS. Therefore, we were able to extract writable property specifications in only one PICS. A way to overcome this problem is collecting traffic for a longer period, which increases the chance to observe relatively rare events. Moreover, in BASs where passiveness is not a strong requirement, actively probing devices can reveal the required features in short time.

We passively collected training traffic from a real network during 4 days. Nonetheless, we learned the vast majority of features during the first 6 hours. This is, of course, highly dependent on the network activity and the monitoring point where the traffic is collected.

Non-certified devices. PICS constitute the source of our specifications. All BACnet certified devices are required to have a PICS. However, not all BACnet devices are certified, and therefore, the availability of PICS might not be guaranteed. In our network we discovered 6 non-certified devices for which no PICS was found. The recommendation to use only certified devices is twofold. First, it guarantees compliance to the standard. Second, the technical documentation is publicly available in the BACnet International website [3]. The same recommendation has been backed by local regulatory organizations such as AMEV in Germany [21].

Precision & Recall. Considering BACnet networks with high device heterogeneity, manual specification extraction from PICS is unscalable. Automatic rule extraction allows, otherwise unfeasible, specification-based intrusion detection. Ideally, 100% of precision and recall would provide a complete set of specifications fully concordant with the PICSs. In our experiments, our approach achieves this goal for implemented object, writable property, and BIBB specifications. However, implemented property specifications suffered from a minor loss of precision and recall. Even though both parameters are above 99.5%, this opens a small window for false positives and false negatives. False positives are produced by rules that the algorithm failed to extract from the PICS. If present, false alerts can be addressed by adding the missing specification to the automatically extracted set of rules. This ensures that the alert will not occur again. False negatives, on the other hand, are produced by *incorrectly* extracted rules. For example, a property written close to two BACnet objects in the PICS, is misinterpreted and assigned to the wrong object in the specifications. This problem is harder to identify and to fix. Still, operators can use false alerts to investigate whether the missing rule in one place was due to an additional (and incorrect) rule in a different place.

Intrusion Detection. In our experiments, the main cause of alerts were requests to non-implemented properties. We found out that BACnet management software often requests a predefined set of properties (particularly on the *device* object), some of which might not be implemented in the targeted devices. This is the exact same behavior of reconnaissance tools like nmap, as we showed in Sect. 7.4. Since reconnaissance is one of the first stages in active attacks, it is worth carefully looking into this kind of alerts as soon as they are triggered.

The second cause of alerts were undocumented elements in BACnet devices. Again, in Sect. 7.4 we showed how an undocumented BACnet object can conceal a backdoor. It is possible to argue that the same behavior could have been implemented in a *documented* object, and therefore remain undetected by our approach. However, it is worth noting that we found more than 100 undocumented elements in the real BACnet network; any of them potentially hiding dangerous behavior. The evidence collected using real traffic supports the relevance of our detection capabilities.

Finally, it is important to emphasize that we do not claim detection capabilities on all attack instances, but only on those exposing PICS violations. Still, our automatically extracted specifications are simple and yet sufficient to detect a broad set of attacks. Even though evasion is possible, our approach forces the adversary to tailor attacks specifically for the targeted BAS. Thus, we argue that our approach raises the bar for attackers and provides new insights towards a comprehensive defense mechanism for BASs.

9 RELATED WORK

For most protocols, including BACnet, there are a few documents (RFCs, IEEE or ISO standards) describing them. Several researchers have used textual descriptions of network protocols to *manually* extract specifications [13, 22, 26]. We aim to automate the specification extraction process in order to ease the applicability of SB-ID in BAS environments.

Stakhanova *et al.* [28] explore the possibility to automate the specification extraction process for host-based intrusion detection. This is done by 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 automatically populates with rules the specification-based subsystem. Our approach differs not only on the audit data (network traffic) but fundamentally, on the source where the specifications come from (PICS).

Tonejc *et al.* [30] evaluate four different machine learning algorithms to perform anomaly-based intrusion detection in BACnet. The selected features are header fields at different layers (even the IP layer). A different approach presented in [7] uses network flows to detect attacks in BACnet. Our approach exclusively looks at the BACnet application layer, which makes it independent of the underlying protocols.

In [15], Kaur *et al.* developed a traffic normalizer that either drops or fixes (whenever possible) malformed BACnet packets. 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 status flags.

Preliminary work on BACnet timing analysis is presented in [12]. The authors consider legitimate command patterns that could represent an attack. Specifically, the time difference between frames containing write requests is analyzed using an Artificial Neural Network.

The closest work to ours was done by Caselli *et al.* [6]. The authors developed an approach to extract specifications from BACnet PICS written with a predetermined format. Their approach suffers from two major limitations: (1) the parsing process does not yield results from documents using a different format; and (2) even for documents following the dictated format, the combination of device capabilities (text strings) and auxiliary information (arbitrary symbols or text) cannot be disambiguated. Rather than developing a rigid extraction mechanism for each PICS format found in the wild, our approach generalizes the way to interpret different PICS formats using network traffic to solve the incompleteness and ambiguity problems in [6]. Our experimental results, based on our new approach, show a significant improvement over the state of the art. The differences at design level have a direct impact at the implementation level. The gain in flexibility offered by our approach requires additional processing (e.g., network traffic analysis) and infrastructure (e.g., databases). Finally, the experiments presented in [6] demonstrate how rule violations help to discover anomalous behavior that resembles BACnet attacks. Our research not only validates and measures their findings but also presents realistic attack scenarios implemented and executed on an isolated testbed. Unlike the work in [6], the core of our research lies on automatic specification extraction, for this reason, we also evaluate the performance of our algorithm in terms of precision and recall.

10 CONCLUSION

We have presented the first fully automated approach to deploy specification-based intrusion detection at network level. We implemented our prototype on BACnet, in a passive way and with network-wide coverage. Our specifications are individually tailored for each device in the network, in such a way that (1) devices should not *expose* capabilities beyond what is claimed in their PICS; and (2) devices should not *be queried* about capabilities they are not supposed to have according to their PICS. We showed concrete attacks that can be detected if either of the aforementioned conditions are breached. Our evaluations with both, real and synthetic traffic, prove that PICS derived specifications are useful to detect BACnet attacks and behavior that closely resembles attacks.

Unlike previous works, our approach is able to detect attacks comprised of as few as a single BACnet packet. The high precision and recall achieved during the specification extraction process translates into reliable BAS monitoring.

The impact and feasibility of attacks in BASs 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.

ACKNOWLEDGMENTS

This work is partially funded by the Costa Rica Institute of Technology. The authors would like to thank Henk Hobbelink and Jeroen van Ingen for their technical support throughout this project.

REFERENCES

[1] ANSI/ASHRAE STANDARD 135-2016. 2016. A Data Communication Protocol for Building Automation and Control Networks. (2016).

[2] Stefan Axelsson. 2000. *Intrusion detection systems: A survey and taxonomy*. Technical Report. Technical report.

[3] BACnet International 2017. BACnet Testing Laboratories. (2017). http://bacnetinternational.net/btl/

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

[5] Dan Bilefsky. 2017. Hackers Use New Tactic at Austrian Hotel: Locking the Doors. (2017). Retrieved Jun 27, 2017 from https://www.nytimes.com/2017/01/30/world/europe/hotel-austria-bitcoin-ransom.html

[6] Marco Caselli, Emmanuele Zambon, Johanna Amann, Robin Sommer, and Frank Kargl. 2016. Specification Mining for Intrusion Detection in Networked Control Systems. In *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, Austin, TX, 791–806. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/caselli

[7] Pavel Čeleda, Radek Krejčí, and Vojtech Krmíček. 2012. Flow-Based Security Issue Detection in Building Automation and Control Networks.. In *EUNICE*. Springer, 64–75.

[8] The Apache Software Foundation. 2017. Apache Solr. (2017). https://lucene.apache.org/solr/

[9] Glenn Greenwald. 2014. *No place to hide: Edward Snowden, the NSA, and the US surveillance state*. Macmillan.

[10] Stephen Hilt and Michael Toecker. 2017. bacnet-info NSE Script. (2017). https://nmap.org/nsedoc/scripts/bacnet-info.html

[11] Paul Jaccard. 1912. The Distribution of the Flora in the Alpine Zone. *New Phytologist* 11, 2 (1912), 37–50. https://doi.org/10.1111/j.1469-8137.1912.tb05611.x

[12] Michael N Johnstone, Matthew Peacock, and JI den Hartog. 2015. Timing attack detection on BACnet via a machine learning approach. (2015).

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

[14] Steve Karg. 2017. BACnet Stack. (2017). http://bacnet.sourceforge.net/

[15] Jaspreet Kaur, Jernej Tonejc, Steffen Wendzel, and Michael Meier. 2015. Securing BACnets Pitfalls. In *ICT Systems Security and Privacy Protection*. Springer, 616–629.

[16] Stephen Lacey. 2013. Hackers Penetrate Googles Building Management System. (2013). Retrieved Aug 04, 2017 from https://www.greentechmedia.com/articles/read/hackers-penetrate-googles-building-management-system

[17] Vladimir I Levenshtein. 1966. Binary codes capable of correcting deletions, insertions and reversals. In *Soviet physics doklady*, Vol. 10. 707.

[18] Hui Lin, Adam Slagell, Zbigniew Kalbarczyk, and Ravishankar K Iyer. 2012. *Using a specification-based intrusion detection system to extend the DNP3 protocol with security functionalities*. Technical Report. Coordinated Science Laboratory. University of Illinois at Urbana-Champaign.

[19] Gordon Lyon. 2017. Nmap: the Network Mapper. (2017). https://nmap.org/

[20] Michael Newman. 2013. *BACnet The Global Standard for Building Automation and Control Networks*. Momentum Press, New York, N.Y.

[21] Federal Ministry of Transport Building and Urban Affairs. 2011. BACnet in public buildings. (2011). http://www.amev-online.de/AMEVInhalt/Planen/Gebaeudeautomation/BACnet%202011%20V%201.2/bacnet2011v1-2en.pdf

[22] Christoforos Panos, Christos Xenakis, Platon Kotzias, and Ioannis Stavrakakis. 2014. A specification-based intrusion detection engine for infrastructure-less networks. *Computer Communications* 54 (2014), 67–83.

[23] Nicole Perlroth. 2014. Heat System Called Door to Target for Hackers. (2014). Retrieved Jun 23, 2017 from https://www.nytimes.com/2014/02/06/technology/heat-system-called-door-to-target-for-hackers.html

[24] Cartic Ramakrishnan, Abhishek Patnia, Eduard Hovy, and Gully APC Burns. 2012. Layout-aware text extraction from full-text PDF of scientific articles. *Source Code for Biology and Medicine* 7, 1 (2012), 1–10. https://doi.org/10.1186/1751-0473-7-7

[25] Eyal Ronen and Adi Shamir. 2016. Extended functionality attacks on iot devices: The case of smart lights. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*. IEEE, 3–12.

[26] R Sekar, Ajay Gupta, James Frullo, Tushar Shanbhag, Abhishek Tiwari, Henglin Yang, and Sheng Zhou. 2002. Specification-based anomaly detection: a new approach for detecting network intrusions. In *Proceedings of the 9th ACM conference on Computer and communications security*. ACM, 265–274.

[27] Shodan.io. 2017. Shodan. (2017). https://www.shodan.io/

[28] Natalia Stakhanova, Samik Basu, and Johnny Wong. 2010. On the symbiosis of specification-based and anomaly-based detection. *computers & security* 29, 2 (2010), 253–268.

[29] The Bro Project 2017. Bro Network Monitoring System. (2017). https://www.bro.org/

[30] Jernej Tonejc, Sabrina Güttes, Alexandra Kobekova, and Jaspreet Kaur. 2016. Machine Learning Methods for Anomaly Detection in BACnet Networks. *J. UCS* 22, 9 (2016), 1203–1224.