Leveraging Semantics for Actionable Intrusion Detection in Building Automation Systems

Davide Fauri[1], Michail Kapsalakis[2], Daniel Ricardo dos Santos[1], Elisa Costante[2], Jerry den Hartog[1], and Sandro Etalle[1,2]

[1]Eindhoven University of Technology[⋆], [2]SecurityMatters

Abstract. In smart buildings, physical components (e.g., controllers, sensors, and actuators) are interconnected and communicate with each other using network protocols such as BACnet. Many smart building networks are now connected to the Internet, enabling attackers to exploit vulnerabilities in critical buildings. Network monitoring is crucial to detect such attacks and allow building operators to react accordingly. In this paper, we propose an intrusion detection system for building automation networks that detects known and unknown attacks, as well as anomalous behavior. It does so by leveraging protocol knowledge and specific BACnet semantics: by using this information, the alerts raised by our system are *meaningful* and *actionable*. To validate our approach, we use a real-world dataset coming from the building network of a Dutch university, as well as a simulated dataset generated in our lab facilities.

1 Introduction

Building Automation Systems (BAS) are control systems that manage core physical components of buildings such as elevators, heating and ventilation, access control, and video surveillance [4,12]. Besides residential and commercial buildings, BAS also control critical facilities such as hospitals, airports, and data centers. Within a BAS, devices communicate with each other using network protocols such as BACnet, KNX, and Zigbee [8].

With the introduction of the Internet of Things (IoT), BAS may even be connected to the Internet. Hence, attackers can exploit vulnerabilities of protocols and devices to launch attacks on a building, which can lead to economic loss or harm building occupants [9,15]. Attacks on smart buildings can, e.g., cause blackouts by damaging power systems, grant access to restricted areas by tampering with physical access control, or damage data centers by stopping air conditioning. Reported attacks[1] include the 2016 attack that turned off the heating systems in two buildings in Finland and the 2017 attack that locked hotel guests in their rooms in Austria.

[⋆] Partially funded by EU-H2020-CITADEL (nr 700665), ITEA3-APPSTACLE (nr 15017), NWO-IDEA-ICS and NWO-SotJ (nr 628.013.001).

[1] See, e.g., `https://securityledger.com/2016/11/lets-get-cyberphysical-ddos-attack-halts-heating-in-finland/` and `https://www.nytimes.com/2017/01/30/world/europe/hotel-austria-bitcoin-ransom.html`

Intrusion Detection Systems (IDS) can monitor network activity to detect attacks. IDS are typically categorized into *knowledge-based* (when detection rules are specified from attack signatures; also known as misuse-based) and *behavior-based* (when the IDS relies on a model of legitimate behavior). Behavior-based approaches, in turn, are subdivided in *anomaly-based* (when a model of legitimate behavior is *learned*) and *specification-based* (when the model is *specified*) [6]. Applying knowledge-based approaches to BAS is challenging because attack signatures may be device-dependent, which limits their scope and makes them hard to obtain. Anomaly-based approaches [11,17,21] tend to adopt "black-box" machine learning techniques (e.g., artificial neural networks), which do not provide meaningful information to help understand the cause of an anomaly [19] (e.g., whether the anomaly is the result of an attack, or an irregular yet legitimate change). Specification-based approaches are based on vendor-provided documents [2,5], which is problematic when the documents are not available or not easily parsable.

Smart buildings are different from IT systems and even Industrial Control Systems. On one hand, they are dynamic environments where network traffic is a combination of multiple streams belonging to different categories—e.g., periodic time-driven patterns or unstructured human-driven activity [24]—which requires the use of fine-tuned anomaly-based detection that can raise meaningful alerts. On the other hand, the kind of devices hosted by BAS are relatively well-standardized and their protocols are expressive [2], allowing us to more easily derive knowledge-based detection rules. To achieve interpretable and actionable alerts, we leverage BACnet's rich protocol semantics and a semantics-aware detection model.

In this paper, we propose an IDS to monitor building automation networks based on one of the most widely used protocols for BAS; BACnet. The proposed IDS uses knowledge about the semantics of BACnet and the BAS to improve both white-box anomaly detection [3] techniques (for unknown threats), and knowledge-based techniques (for known attacks). To the best of our knowledge, the use of protocol semantics for securing building automation networks has never been proposed. Our approach has two important benefits when compared to related work. First, the white-box intrusion detection approach learns models that are understandable by users, and provides semantically rich alerts that clearly indicate the reasons of an anomaly. The alerts are thus easier to interpret for network operators, which improves actionability [7]. Second, our approach does not depend on vendor-specific descriptions of each device. Instead, we exploit the structure imposed by the BACnet standard to elevate the knowledge-based part from signatures to more general knowledge about attack patterns. Note that, although we focus on BACnet, similar methods and techniques may be used for other building automation protocols, provided that they are as expressive as BACnet.

The rest of this paper is organized as follows. Section 2 provides background on the BACnet protocol and attacks in this scenario. Section 3 details our combined IDS approach. Sections 4 and 5, respectively, discuss implementation and experiments using a real dataset from the network of a Dutch university, and a simulated dataset generated in our lab facilities. Section 6 concludes the paper.

2 Background

BACnet [1] is one of the most widely used protocols for building automation. It is based on four layers: Physical, Data Link, Network, and Application. There are several BACnet variants. The Network and Application layers are the same for all variants, but there are seven possible combinations of Physical and Data Link layers, which are chosen according to requirements such as cost and speed.

A BACnet subnetwork is a connection of devices with the same Physical and Data Link layers that can directly exchange unicast, multicast or broadcast messages. A BACnet network consists of multiple subnetworks connected by BACnet Broadcast Management Devices used to broadcast messages from one subnetwork to another. If the interconnected subnetworks use different Physical and Data Link layers, they must also be connected by a BACnet Router.

BACnet defines a standard set of *Objects*, each with a standard set of *Properties* that together describe a device and its current status. *Services* are used by one BACnet device to obtain information from another device or command another device to perform an action. Each service request and service acknowledgment transmits properties of objects using a message packet sent over the network.

Every BACnet device must have a `Device` object, whose properties describe the device to the network. The choice of which other objects, properties, and services are present in a device is determined by its function and capabilities (e.g., an `AnalogInput` object is used to represent an analog sensor input). Some properties, such as `Description` and `DeviceType`, are set during installation; others, such as `PresentValue`, provide status information (e.g., the sensor input represented by the `AnalogInput` object). The `ReadProperty` service is implemented by every device to inform its properties to another device.

BACnet security. The BACnet standard specifies some security features to provide, e.g., data confidentiality and integrity, but their implementation is optional. This means that, in most smart buildings, BACnet data is exchanged without any kind of authentication, and BACnet devices are programmed to process every received message, opening them to exploitation by internal and external attackers [23]. There are several examples of attacks on BACnet devices and networks in the literature (see, e.g., [9,13,17]). We classify these attacks in the following four categories:

Network Reconnaissance (or Snooping) aims at gaining knowledge of network topology and information about objects, properties, and services. This knowledge can be used to plan the next actions of an attack or to organize a break-in by determining if people are present in the building (see, e.g., [14]).

Device Writing Access (or Tampering) can be used to isolate devices, compromise them to operate abnormally, or remotely control devices such as doors and elevators.

Traffic Redirection (or Spoofing) impersonates a device or a BACnet Router so that messages intended for a certain device never reach their destination.

Denial of Service (DoS) disables the communication between devices or makes a whole subnetwork unavailable. DoS attacks can isolate critical systems of a building, such as fire detectors.

Related work. Pan et al. [17] use a rule learner to detect abnormal BACnet traffic and to classify it according to attack types. They also propose an action handler to discard malicious packets. Johnstone et al. [11] used an Artificial Neural Network to detect specific timing attacks, e.g., values that are changed in quick succession, in BACnet. Tonejc et al. [21] introduced a framework that allows the characterization of BACnet network traffic using unsupervised machine learning algorithms, such as clustering, random forests, one-class support vector machines and support vector classifiers, after a pre-processing step that includes principal components analysis for dimensionality reduction. They consider packet headers, which reflect the network structure, but not the actual application data.

A major disadvantage of the machine learning methods above is that they are "black-box" models, in the sense that they are hard to understand and modify and their alerts have a wide semantic gap, i.e. they do not provide enough semantic information to help understand the cause of an anomaly and to fix it [19].

Zheng and Reddy [24] observe that BACnet traffic is a combination of multiple flow-service streams that belong to "THE-driven" categories: Time-driven, Human-driven, and Event-driven. The authors then developed different intrusion detection systems based on traffic classification and different anomaly-detection models: interarrival-based for time-driven traffic, safe range-based for human-driven traffic, and volume-based for event-driven traffic. The authors do not consider knowledge or specification-based detection in their system.

Caselli et al. [2] presented a specification-based BACnet IDS. In their implementation, when model names and vendor IDs are discovered, the system looks for documentation related to each device in the Internet. From these documents (e.g., PICSs) and system configuration files, the IDS automatically generates detection rules, e.g., permissible services, objects, and properties of each device. The IDS then monitors the network with the extracted rules, raising an alert when a packet violates any of them. Their approach suffers the already mentioned disadvantages of depending on the availability and readability of specification documents. More specifically, it requires documents to have a specific format and unambiguous notation. To overcome these limitations, the approach of [5] generalizes the interpretation of different PICS formats using network traffic to solve the incompleteness and ambiguity problems.

Some works aim to not only detect but also prevent attacks in BACnet. Examples include firewalls [10] and intrusion prevent systems [13] that drop non-conforming packets, as well as traffic normalizers [20] that actively modify malicious BACnet traffic. All such tools can have serious consequences in building automation networks when dropping or modifying legal messages, thus delaying or ignoring critical actions. Another disadvantage is that they are unable to detect or prevent unknown attacks.

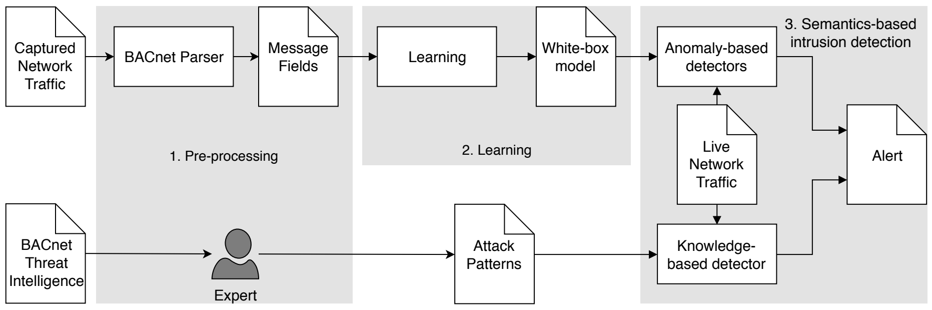

Fig. 1: Overview of our proposed solution

3 Intrusion detection

Figure 1 shows an overview of the IDS divided in three phases. The *Pre-processing* phase analyzes two sources of knowledge. The first is the BAS *Network Traffic* capture, which is processed by a *BACnet Parser* to extract the relevant *Message Fields* from each message. The second is a collection of *BACnet Threat Intelligence* resources, which are interpreted by a human domain *Expert* and manually refined into *Attack Patterns*.

In the *Learning* phase, a white-box model of legitimate (normal) behavior is learned from the parsed Message Fields.

The *Intrusion detection* phase is divided in two modules, one for *Knowledge-based* and one for *Anomaly-based* detection. Both modules continuously take Message Fields as input and can raise alerts for malicious behavior detected in the network, but they are complementary. The knowledge-based module compares a black-list of well-known Attack Patterns with the activity in the network traffic: the false positives rate is usually very low, but the obvious shortcoming is that unknown attacks are not detected. The anomaly-based module detects previously unseen attacks: it raises an alert whenever a device sends an anomalous number of messages, or when the content of the observed messages is abnormal.

The modular system described above supports different detection approaches, where modules are knowledge-based or anomaly-based detectors that are executed in parallel. Below, we describe the concrete detectors used in our implementation: the anomaly-based detection engine is composed by two detectors that are triggered by messages, or time passing; while the knowledge-based detection engine is composed by a detector triggered by messages.

3.1 Semantics-based anomaly detectors

Value range. BACnet objects assuming different values, such as an `AnalogInput` representing temperature, will have their values stay within a bounded range during normal behavior. Tampering attacks can be detected, and an alert raised, when an attacker tries to change the `PresentValue` property of an object to a value outside this range. To detect them, during a learning phase we build a white-list consisting of ranges of normal values for every such object. We do so by considering all the services that transmit values (e.g. `ReadProperty`, `WritePropertyMultiple`, etc.) and noting the minimum and maximum observed values. For instance, if during the learning phase we observe `ReadProperty` messages that contain the values 2 and 5 for the `PresentValue` property of an `Analog Input` object; and later we observe `WriteProperty` messages that contain the values 4 and 6 for the same property of the same object; then the observed normal range for that property is [2,6].

To reduce false positives in the detection, we widen the range by a certain tolerance t: in the example above, $t = 5\%$ would expand the normal range to [1.9, 6.3]. The BACnet protocol specification distinguishes between `Output` objects, usually sensor measurements, and `Input` objects, usually setpoints sent to actuators. This semantic distinction allows us to set two values for t: a stricter tolerance for setpoints, which have typically a low variance, and a more lenient one for noisy sensor readings. The tolerance value may also depend on the criticality of the object being monitored: for instance, a temperature setpoint of a server room should have a strict tolerance, while a hot water setpoint in a house could be much more lenient.

Number of messages. During normal behavior, we expect the frequency of messages having similar sources and types to fall within a normal range of values. If a device is compromised or an attack occurs (e.g. reconnaissance, denial of service), it may lead to an abnormal number of messages sent for a specific service. We thus raise an alert whenever we observe an anomalous frequency of messages sent for a service (either in general or by a specific device). We focus on computing the frequency per-service instead of per-device because due to how messages are propagated among BACnet subnetworks, knowledge of the initiated service is crucial in diagnosing the reason of excessive traffic [16].

To learn this normal frequency feature, we first divide the learning period $\mathcal{L} = \{I_0, ..., I_n\}$ in a sequence of consecutive time intervals $I = [t, t+T)$ with equal duration T. For each time interval, we gather a sample set $\mathcal{O}$ consisting of all the messages observed over that interval. We then define, for each given source device s and service k, a feature that counts the total number of messages sent by s that refer to k in an interval: $f(I) = \#\{m \in \mathcal{O} \mid m.source = \ s \wedge m.service = k\}$. We similarly compute the average count of messages referring to the service k, normalized on the number of devices that have ever been observed initiating that service: $g(I) = \#\{m \in \mathcal{O} \mid m.service = k\}/\#\{s \in D \mid s \text{ is active for } k\}$, where D is the set of all monitored network devices, and a device s is active for a service k if we observed s initiating k at least once during the learning period.

In learning an interval NV of normal values for the feature f (equivalently for g), we apply a metric over the set F of features computed over all time intervals, $F = \{f(I_0), \ldots, f(I_n)\}$. We considered three possible choices of metric: min-max, distance from mean, or deviation from median. We already used the min-max metric above; the interval runs from the minimum to the maximum value seen in F, extended with a tolerance t; i.e. $NV = [(1-t) \cdot \min_{f \in F} f, (1+t) \cdot \max_{f \in F} f]$. Similarly, the distance from mean starts from the mean frequency μ_f and is extended to $NV = [(1-t)\cdot\mu_f, (1+t)\cdot\mu_f]$. Deviation from median m_f uses a tolerance based on Median Absolute Deviation (MAD) rather than a fixed percentage: $NV = [(1 - c\cdot MAD)\cdot m_f, (1 + c\cdot MAD)\cdot m_f]$, where $MAD = median_{f\in F}(|f - m_f|)$ and c is a constant (called cutoff) given by the user.

In the detection phase, messages are filtered by source s and service k, and are sampled with time intervals of the same length T as above. The feature f is calculated over each of these intervals as they come. An alert is raised if the observed number of messages is outside the learned normal range for f and g. When s is a new device on the network, a normal range of values is not available as it has not been learned yet. Instead, we compare the value of the feature f with only the normal range for the average service frequency, that is g. Because our method tries to detect anomalies that can harm the system, we concentrate on message frequencies that are more than the upper bound of NV; as a consequence, devices that send less messages than the lower bound will not trigger any alert.

3.2 Knowledge-based detector

This detector uses a black-list of known attack patterns expressed in terms of the BACnet and BAS semantics. An expert can specify stateless detection rules, checking for known malicious values in a combination of one or more message fields. For example, a rule may raise an alert if the source address of observed messages is set to a broadcast address, either in the IP layer or in the BACnet Network Layer, as this is indicative of a DoS attack. We also consider stateful rules; for example, observed messages having the same `BACnetAddress` but different `DeviceID` are indicative of device spoofing; similarly, network number spoofing may be detected by looking for different `NetworkNumber`s for the same `Router`. In this case, the state comprises the pairs `(BACnetAddress,DeviceID)` and `(Router,NetworkNumber)` observed so far.

4 Implementation

We implemented the intrusion detection modules on top of SilentDefense[2], an IDS for industrial control systems developed by SecurityMatters. We used Wireshark's BACnet dissector[3] to represent BACnet packets in a readable format and developed a custom parser using binpac [18]. The parser provides the extracted fields to the Deep Protocol Behavior Inspection engine of SilentDefense, which allows a security operator to see all BACnet message details. The intrusion detection scripts were implemented in Lua.

Figure 2 shows an example alert raised by the IDS when a device does not conform to its normal behavior for service `ReadPropertyMultiple`. Notice how this alert is informative and enables a security operator to quickly assess the situation. In case the operator realizes this is a false positive, the upper limit in the valid range can be easily changed to an appropriate value (i.e. there is no need to learn the model again).

[2] `https://www.secmatters.com/product`
[3] `https://wiki.wireshark.org/Protocols/bacnet`

```
Device does not conform to device and service normal number of messages for service
ReadPropertyMultiple. Details:
BACnet device 100 sent 3643 messages during a 30 minutes interval.
Normal number of messages for this device for this service: [0, 703].
Normal number of messages for this service: [3593, 3641].
```

Fig. 2: Example alert

BACnet testbed. To run attacks and test our intrusion detection approach, we developed a testbed modeling a lighting and temperature control system in a small building, and containing the following real devices:

– two sensors (motion & temperature) and two actuators (fan & LED bulb);
– one digital I/O and two analog I/O devices connected via serial cable to the sensors and actuators, and communicating via BACnet MS/TP;
– a BACnet Router that connects one MS/TP network with one IP network;
– a BACnet/IP Controller that implements the logic of the system by reading and writing inputs and outputs of the I/O modules;
– a BACnet/IP Workstation used to configure devices in the network;
– a BACnet/IP Workstation that monitors the testbed and lets users modify setpoints;
– a Raspberry Pi used to run attacking scripts from the IP network.

The testbed implements two automated functions. First, when the motion sensor state goes from 0 to 1, the controller sends a command to the I/O module to switch on the LED by changing the state of one of its outputs. Second, the I/O module continuously reads the temperature values sent by the sensor and informs the controller. The controller activates the fan when the sensed value is greater than a setpoint set by the operator.

Attacks. We implemented the following synthetic attacks in Python, using the bacpypes[4] library to exchange BACnet messages between the Raspberry Pi and the rest of the testbed network. All the attacks are successful, because the BACnet devices in the testbed do not implement any authorization check and accept all the messages that come from any device. This is typical for building automation systems [23].
Snooping. We broadcast `Who-Is` messages to retrieve the address and instance number of all devices in the network, and then send `ReadProperty` services to these devices to read their model name, vendor ID, and the objects and services supported by them.
Tampering1. We send a `WriteProperty` request to the digital I/O controller and toggle the state of the LED bulb or of the cooler fan. In this attack, we send a single message to change the state of an output only once.
Tampering2. We send a `ReadProperty` request to the main controller to extract the current (analog) temperature setpoint value; we then send a `WriteProperty`

[4] `https://github.com/JoelBender/bacpypes`

request to the same controller to increase this value by five degrees. As a result, the fan stops working and the temperature increases in the room.
Spoofing1. We listen to BACnet messages until receiving an `I-Am` unconfirmed request with device instance number equal to that of the digital I/O controller. We then immediately send a new `I-Am` message with the same details, except for a malicious IP address. As a result, the connection between the other BACnet devices in the network and the legitimate device is broken and the attacker is able to read and change their contents.
Spoofing2. Similar to the previous attack, but in this case we impersonate a BACnet Router by sending fake `I-Am-Router-To-Network` messages including the network number of another legitimate BACnet Router. The goal of this attack is twofold: i) traffic redirection, since all BACnet/IP devices that want to communicate with non-BACnet/IP devices nested behind this router send messages to the attacker machine; ii) denial of service, since the nested devices cannot receive messages from BACnet/IP devices. However, BACnet/IP devices reconfigure their routing tables when a nested device sends any kind of message, because the network number is included in BACnet Network Layer and the message is sent by the legitimate BACnet Router.
Reflected DDoS. We broadcast 1000 `Who-Is` requests in a few seconds, without a device range. As a result, a total of 6000 `I-Am` messages are broadcasted by the 6 BACnet devices in the testbed. The BACnet Router is overloaded and starts rejecting all the messages that it receives: as a result, the BACnet/IP devices cannot communicate with BACnet MS/TP devices.

5 Experiments

The goal of the experiments was to validate the attack detection capabilities of our IDS, and to measure how many false positive alerts (FP) it raised on legitimate traffic. To achieve those goals, we used a real and a synthetic dataset. We split each of them into 70% for learning the white-box model, and 30% for validating the false alerts. We considered these datasets to be attack-free: therefore, we regarded any alert raised from the validation data as a false alert.
Dataset 1 comes from a real BACnet network of a Dutch University. We analyzed nine days of traffic, totalling 106 GB of data and 20 million BACnet messages. We could not use the infrastructure of the university to perform attacks, but Dataset 1 helped us to evaluate the number of false alerts that our IDS might raise when deployed in a real scenario. We extracted two partial datasets from Dataset 1 to examine whether the duration of the learning period affected the accuracy of our IDS. The first partial dataset (D1.1) includes approximately 4 days of network traffic, split in 50 hours of training and 47 hours of testing. The second partial dataset (D1.2) includes the whole 9 days of traffic, split in 172 hours of training and the same 47 hours of testing as in D1.1.
Dataset 2 comes from our BACnet testbed presented in Section 4. We captured 10 minutes of traffic with no attacks: due to the small size of our testbed, this short time span is still sufficiently representative of the normal behavior on the

network. After measuring the number of false alerts from the validation data, we then re-used the same white-box model learned from the training data to test the detection capabilities of our IDS. To do so, we launched the attacks described in Section 4 and evaluated if the IDS raised a corresponding alert.

Results. Our IDS managed to raise alerts for 5 out of 6 attacks: all of the *Snooping*, *Spoofing* and *Reflected DDoS* attacks were detected by either the number of messages anomaly detector, or the knowledge-based detector. Among the *Tampering* set of attacks, the IDS could only detect the *Tampering2* attack, through the value range anomaly detector; the other attack took place unnoticed. This is not surprising: the *Tampering1* attack was expected to be undetectable by our approach, as it is just an isolated command that simply toggles the binary value of a switch from on to off. During the learning phase we observed both of these values, which were then included in the range of normal data. From the point of view of the IDS, the attack was thus indistinguishable from the action of a legitimate user. It is important to note that no single detector could catch all types of attacks; we conclude that we need a combination of anomaly-based and behavior-based detectors to detect different kinds of attacks.

To evaluate the usability of our IDS, we followed the work of [22] and computed both the total number of FP and the average rate of FP per hour which were raised on our datasets. We limit this analysis on the two semantics-based anomaly detectors, as we reasonably expect that the specified detection rules used in the knowledge-based detector are precise enough to not generate many FP.

Table 1a presents the evaluation results for the value range anomaly detector, with two possible settings for the interval tolerance parameter t. The training interval in D1.1 is clearly too short to learn the full range of behaviour leading to many false positives. The longer training period in D1.2 which spans a whole week leads to fewer FP. Buildings are live, dynamical systems with many time-driven and human-driven regularities [24]: seven days is a manageable period of time in which we expect to observe the full range of normal values. We still need

Table 1: False positive alerts raised by the anomaly detectors

	D1.1	D1.2
$t = 5\%$	6144	1531
$t = 20\%$	2510	3

(a) Value range detector

	D1.1		D1.2		D2	
Interval	30m	60m	30m	60m	30s	60s
min-max	204	141	54	8	1	1
mean	1058	506	1151	528	6	5
$\text{median}_{c=1}$	676	521	763	563	4	4
$\text{median}_{c=3}$	505	398	511	444	1	1
$\text{median}_{c=5}$	456	379	418	400	0	0

(b) Number of messages detector

to adjust t to balance the tradeoff between detection and FP rate, taking into account the criticality of the monitored value. Assuming a tolerance of 20% is acceptable throughout all the monitored buildings in the network, and using the longer training period (D1.2) results in around 0.06 FP/h.

Table 1b shows the results for the number of messages anomaly detector. We tested different settings during the learning phase. When computing the frequency values, we used two different interval durations: T=30 and 60 minutes for Dataset 1, and T=30 and 60 seconds for the considerably shorter Dataset 2. When computing the range of normal values, we used the following metrics and parameters: min-max with tolerance $t = 5\%$; deviation from the median with cutoff values $c = 1$, 3 and 5; and distance from mean with tolerance $t = 5\%$. The min-max metric provides the least false alerts, since by construction it does not regard any value from the training data as anomalous. However, outliers during training could lead to overly large intervals, hindering detection. Mean and especially median are more robust to outliers during training. The tighter resulting intervals do cause more FP. We also see that training on both work and weekend days (D1.2) skews the intervals for measures of central tendency such as mean and median, leading to slightly more FP. As behaviour differs between work and weekend days adding profiling [3] would likely improve results.

Furthermore, Table 1b indicates that both the duration of the intervals and the size of the dataset can have an effect on the number of false alerts. We observe that using longer time intervals reduces the number of FP. This reduction happens because, in both training and detection, we compute the frequency of messages over a longer time T. This increases the chance of ‘averaging out’ the effect of short, possibly anomalous bursts of intense traffic: frequencies computed during detection will tend to be closer to the learned normal values, unless the bursts last for a significant portion of the time interval defined by T.The parameter T should then be tuned with care to balance FP rate, duration of the attacks that can be detected, and the timeliness of the raised alerts.

6 Conclusion

We proposed an IDS approach for BACnet networks that leverages the semantic information provided by the communication protocol. It exploits known attack patterns and normal network behavior of BACnet devices to detect a significant number of attacks. Once an attack is detected, the system generates enriched alerts that include semantic information helpful to the operators.

The IDS provided good results to detect the implemented BACnet attacks, while raising a satisfactory number of false alerts. The tolerance levels for the anomaly-based modules depend on the operation and criticality of each building. In general, we suggest thresholds and cutoffs that are able to balance false alerts and detection rates. As future work, we intend to test whether other BAS protocols, such as KNX and ZigBee, offer enough semantics information to allow for a similarly made IDS.

References

1. ASHRAE: BACnet - a data communication protocol for building automation and control networks. Standard (2016)
2. Caselli, M., Zambon, E., Amann, J., Sommer, R., Kargl, F.: Specification mining for intrusion detection in networked control systems. In: Proc. of USENIX Security (2016)
3. Costante, E., den Hartog, J., Petković, M., Etalle, S., Pechenizkiy, M.: A white-box anomaly-based framework for database leakage detection. JISA **32** (2017)
4. Domingues, P., Carreira, P., Vieira, R., Kastner, W.: Building automation systems: Concepts and technology review. Computer Standards & Interfaces **45**(Supplement C) (2016)
5. Esquivel-Vargas, H., Caselli, M., Peter, A.: Automatic deployment of specification-based intrusion detection in the BACnet protocol. In: Proc. of CPS-SPC (2017)
6. Etalle, S.: From intrusion detection to software design. In: ESORICS (2017)
7. Fauri, D., dos Santos, D., Costante, E., den Hartog, J., Etalle, S., Tonetta, S.: From system specification to anomaly detection (and back). In: CPS-SPC (2017)
8. Hersent, O., Boswarthick, D., Elloumi, O.: The Internet of Things: Key Applications and Protocols. John Wiley & Sons (2011)
9. Holmberg, D.: BACnet wide area network security threat assessment. Tech. rep., NIST (20013)
10. Holmberg, D.: Using the BACnet firewall router. ASHRAE Journal **48**(11) (2006)
11. Johnstone, M., Peacock, M., den Hartog, J.: Timing attack detection on BACnet via a machine learning approach. In: Proc. of AISM (2015)
12. Kastner, W., Neugschwandtner, G., Soucek, S., Newman, H.M.: Communication systems for building automation and control. Proceedings of the IEEE **93**(6) (2005)
13. Kaur, J., Tonejc, J., Wendzel, S., Meier, M.: Securing BACnet's pitfalls. In: Proc. of IFIP SEC (2015)
14. Möllers, F., Sorge, C.: Deducing user presence from inter-message intervals in home automation systems. In: Proc. of IFIP SEC (2016)
15. Mundt, T., Wickboldt, P.: Security in building automation systems - a first analysis. In: Proc. of Cyber Security (2016)
16. Newman, H.: Broadcasting BACnet®. ASHRAE Journal **52** (2010)
17. Pan, Z., Hariri, S., Al-Nashif, Y.: Anomaly based intrusion detection for building automation and control networks. In: Proc. of AICCSA (2014)
18. Pang, R., Paxson, V., Sommer, R., Peterson, L.: Binpac: A yacc for writing application protocol parsers. In: Proc. of IMC (2006)
19. Sommer, R., Paxson, V.: Outside the closed world: On using machine learning for network intrusion detection. In: Proc. of IEEE S&P (2010)
20. Szlósarczyk, S., Wendzel, S., Kaur, J., Schubert, F.: Towards suppressing attacks on and improving resilience of building automation systems - an approach exemplified using BACnet. In: GI Sicherheit (2014)
21. Tonejc, J., Guttes, S., Kobekova, A., Kaur, J.: Machine learning methods for anomaly detection in BACnet networks. JUCS **22**(9) (2016)
22. Urbina, D., Giraldo, J., Cardenas, A., Tippenhauer, N., Valente, J., Faisal, M., Ruths, J., Candell, R., Sandberg, H.: Limiting the impact of stealthy attacks on industrial control systems. In: Proc. ACM SIGSAC CCS (2016)
23. Wendzel, S., Tonejc, J., Kaur, J., Kobekova, A.: Cyber Security of Smart Buildings (2017)
24. Zheng, Z., Reddy, A.: Safeguarding building automation networks: THE-driven anomaly detector based on traffic analysis. In: Proc. of ICCCN (2017)