Role Inference + Anomaly Detection = Situational Awareness in BACnet Networks

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

[1] Technical University of Eindhoven, 5600MB Eindhoven, NL,
{d.fauri, j.d.hartog, s.etalle}@tue.nl
[2] Forescout OT Center of Excellence, John F. Kennedylaan 2, 5612AB Eindhoven, NL
{michail.kapsalakis, daniel.dos.santos, elisa.costante}@forescout.com

Abstract. In smart buildings, cyber-physical components (e.g., controllers, sensors, and actuators) communicate with each other using network protocols such as BACnet. Many of these devices are now connected to the Internet, enabling attackers to exploit vulnerabilities on protocols and devices to attack buildings. Situational awareness and intrusion detection are thus critical to provide operators with a clear and dynamic picture of their network, and to allow them to react to threats and attacks. Due to Smart Buildings being relatively dynamic and heterogeneous environments, situational awareness further needs to rapidly adapt to the appearance of new devices, and to provide enough context and information to understand a device's behavior. In this paper, we propose a novel approach to situational awareness that leverages a combination of learning and knowledge of possible role devices. Specifically, we introduce a role-based situational awareness and intrusion detection system to monitor BACnet building automation networks. The system discovers devices, classifies them according to functional roles and detects deviations from the assigned roles. To validate our approach, we use a simulated dataset generated from a BACnet testbed, as well as a real-world dataset coming from the building network of a Dutch university.

1 Introduction

Building Automation Systems (BAS) are control systems that manage core physical components of building facilities such as elevators, access control, and video surveillance [12, 6]. 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. In this paper, we focus on BACnet [1], which is one of the most widely used protocols for BAS.

BACnet specifies optional security features for data confidentiality and integrity. Nevertheless, most smart buildings exchange data without authentication and devices are programmed to process every message received, opening them to exploitation by internal and external attackers [19]. These attacks can lead to

economic loss or even harm building occupants [10, 13]. Addressing the security of smart building networks is thus fundamental, but few solutions have been proposed in the literature (see, e.g., [7, 8, 11, 14, 16, 22]).

Security solutions are needed that make it easier to understand the heterogeneity of devices in a BAS [12]. Moreover, we note that buildings are live, dynamical systems: over time, their networks are expanded and modified with new devices. We therefore also need a solution that can easily adapt to changes in the system. Two complementary non-intrusive security techniques are: *situational awareness*, which helps in the identification and mitigation of security risks via detailed descriptions in a network map; and *intrusion detection*, which finds anomalous communication that may indicate the presence of attacks. However, to the best of our knowledge, there is currently no solution that provides situational awareness by automatically and continuously identifying, characterizing, and grouping BACnet devices in a monitored network. As for intrusion detection, current specification-based approaches depend on vendor-provided documentation [4, 7], which is problematic when the documents are not available or not easily parsable. Conversely, learning-based approaches [11, 14, 16] usually adopt black-box techniques, which provide little semantic information to help understand the cause of an anomaly and fix it [15]; moreover, such systems typically are limited in their adaptability, requiring to run a new learning phase whenever the network is modified (e.g., new devices appear on the network).

To overcome the gaps above, we propose a *role-based* situational awareness and adaptable intrusion detection system to monitor BACnet networks. A *role* represents the functional behavior assumed by a device in a network. For instance, *workstations* perform management functions; *field devices* and *controllers* perform automation functions; and *routers* perform network functions. Roles help improve situational awareness and intrusion detection in two fundamental ways:

1. they improve *understandability* of alerts and the network map: the role provides valuable context information, e.g. while a workstation sending out many requests may still be performing a legitimately function, this is unlikely when done by a controller. This knowledge improves the actionability of the alert; it becomes easier to select an appropriate response. By grouping together devices with similar function and properties it enriches the network map, improving awareness.
2. they improve *adaptability*: e.g. when a new device appears on the network, it is possible to apply rules and models based on the device's role. (Determining a device's role can be done much more quickly than the learning of a full intrusion detection model.)

Although some characteristics of a device (e.g., protocol and vendor) can be known by just parsing the observed network traffic, learning the role of a device is not trivial for three reasons. First, a role is not directly defined by the location of the device in the network nor by its capabilities as documented by the vendor (its so-called *BACnet profile* [1]), since smart devices can perform multiple roles and devices with different roles can be placed on the same network [12]. Second, devices can behave differently than documented [7]: a device can be over-specified

(when it is used for a role 'below' its documented capabilities) or under-specified (when it performs more functions than expected for its profile). Third, a BAS can change due to the addition of new subsystems or the integration in a larger system spanning multiple buildings. In such cases, the same roles may be performed by devices with different vendors, different capabilities or a different naming scheme. Thus, a device's role should be *inferred* from monitoring the device's behavior.

The main **contributions** of our approach are: (i) *role extraction*: the identification, classification, and grouping of devices in a network with specific behavioral roles inferred from fields extracted from BACnet network messages, using heuristics based on the protocol specification and on similarity classification; (ii) *role-based network intrusion detection* to raise alerts when a device behaves in disagreement with its assigned role; and the creation of (iii) a *network map with device roles* as well as device description (e.g., vendor, model, and firmware version)and device connections, which can be used for security assessments and to provide context for security alerts.

Through the contribution above our approach offers adaptability; network topology and device classifications (roles) are dynamically learned from network traffic, without depending on vendor-specific descriptions of each device, ensuring the dynamic map stays up-to-date with the adapting system. It also offers understandability: the intrusion detection model provides semantically rich alerts that are more easily interpreted by network operators, especially aided by the context provided by the dynamic network map.

The rest of this paper is organized as follows. Section 2 describes the details of BACnet used throughout the paper and discusses related work; Section 3 details our approach to situational awareness and intrusion detection; Section 4 shows our experimental setup and evaluation, using a real dataset coming from the network of a Dutch university, as well as a simulated dataset generated from a real testbed; and Section 5 concludes the paper.

2 Background

Network levels. A BAS is usually divided in three functional levels [12]. The *field level* contains sensors and actuators that interact with the physical world; the *automation level* implements the control logic to execute appropriate actions; and the *management level* is used by operators to monitor, configure and control the whole system. Devices in these levels communicate via network packets to inform their states and send commands to each other. Sensors send their readings to controllers, which in turn decide what actions to take and communicate their decisions to actuators. Recently, devices in the field and management levels became capable of performing tasks pertaining to the automation level. Thus, modern BAS network architectures may sometimes be simplified to two levels: multiple, local *control* networks interconnected by a common *backbone* network.

Protocol layers. BACnet [1] can be used on both control and backbone networks because its architecture is based on four layers. The Application and Network layers always have the same structure and are transparent to the underlying

network infrastructure. The choice of Data Link and Physical layers, instead, defines one of several BACnet variants. BACnet/IP, the protocol variant that uses UDP/IP in the Data Link Layer, is commonly used for the backbone communication between workstations and controllers, while BACnet MS/TP, another variant, is commonly used on control networks to connect to field devices.

In BACnet/IP, each node in the local network has a unique BACnet MAC address consisting of four bytes of IP address and two bytes of UDP port. The BACnet Virtual Link Layer (BVLL) is used for Data Link and provides the interface between the underlying capabilities of the communication infrastructure and the BACnet Network Layer. The Network Layer is used to unicast or broadcast messages on remote networks. These messages can be used for routing and network discovery, or to convey Application Protocol Data Units (APDU) between devices. The Application Layer is used to exchange data between BACnet devices using APDUs, which contain the actual application data, such as the present value of a thermostat. Application Layer messages have different PDU types: `Confirmed-Request`s are generated by requesting *client* devices, while `ACK`s or `Error`s are generated by responding *server* devices.

Network topology. A single BACnet network is a connection of devices with the same Physical and Data Link layers that can directly exchange unicast, multicast or broadcast messages. An interconnection of two or more BACnet networks using different Physical and Data Link layers (for example, the backbone and control networks) constitutes a BACnet internetwork. The devices responsible for transferring messages between different types of network are called BACnet Routers. Each network is assigned a unique BACnet network number and Routers advertise the numbers of networks that they route. Additionally, BACnet Broadcast Management Devices (BBMD) are used to propagate broadcast messages from one network to another. Devices that speak BACnet but are not exclusive members of a BACnet network (e.g., workstations) have to register to a BBMD as Foreign Devices. Finally, BACnet Gateways are used to route and translate messages towards networks with other communication protocols (e.g. KNX).

Objects, properties, and services. BACnet defines a standard set of *objects*, each with a standard set of *properties* that describe an object and its current status to other devices in the BACnet internetwork. *Services* are used by one BACnet device to obtain information from another device or command another device to perform an action. Each time a service is initiated by a client (or executed by a server), a request (or acknowledgement) message is sent over the network, transmitting properties of objects.

Every BACnet device must implement 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 analog sensor would possess an `AnalogInput` object). Some properties, such as `Description` and `DeviceType` are configured 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.

Related work. Three other applications discover and classify devices in BACnet Networks. Redpoint[3] includes an Nmap plugin that, contrary to our passive approach, sends BACnet commands to discover a network topology and enumerate all its BACnet devices. GRASSMARLIN[4] only classifies BACnet devices as client or server, as opposed to our role assignment. Finally, Caselli et al. [4] classify BACnet devices using device-specific Protocol Implementation Conformance Statements (PICSs). This allows for a better classification when vendors have extended the set of services that a device can initiate or execute.

Caselli et al. [4] also presented a specification-based BACnet IDS. Their system discovers model names and vendor IDs from the network traffic, and then searches the Internet for documentation related to each device. From these documents (e.g. PICS) and system configuration files, the IDS automatically generates a detection model about which services and properties are allowed for each device. Their approach suffers the disadvantage of depending on the availability of specification documents, and on them being machine-readable in the first place. More specifically, it requires documents to have a specific format and unambiguous notation. To overcome the latter limitations, the approach of [7] generalizes the interpretation of different PICS formats using network traffic. Our approach, on the other hand, does not need any external document.

Among anomaly-based BACnet IDS, different techniques were proposed. Pan et al. [14] used a rule learner to classify abnormal BACnet traffic according to attack types; the authors also proposed an action handler to discard malicious packets. Johnstone et al. [11] used an Artificial Neural Network to detect timing attacks in BACnet, e.g., values that are changed in quick succession. Tonejc et al. [16] 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, after a dimensionality reduction pre-processing step . The authors focus on the headers of packets, which reflect the structure of the network, but neglect 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 [15]. To address this issue, a different type of "white-box" IDS has been proposed in [5] for monitoring database transactions, and has been successfully applied to different scenarios [9, 20, 21]. In particular, both us [8] and Zheng and Reddy [22] used a white-box model as part of an IDS for BACnet networks, monitoring variable values and number of messages. Both the approaches are limited, though, in that they only monitor the network communication and don't provide context about the devices that are communicating.

[3] https://github.com/digitalbond/Redpoint
[4] https://github.com/iadgov/GRASSMARLIN

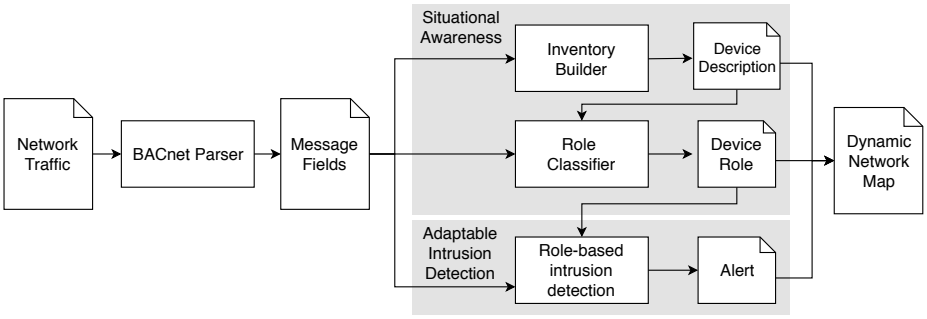

Fig. 1. Overview of our approach

3 Approach

Figure 1 shows an overview of our system, which has two main components: a *Situational Awareness* module that identifies and classifies devices in the network and an *Intrusion Detection* module that raises semantic *Alerts* when possible intrusions are detected (the semantics is provided by the information gathered and inferred about the devices involved in an alert).

All the information (e.g., attributes and roles of a device, raised alerts) is displayed on a *Dynamic Network Map*, i.e. a continuously updated graph that represents network devices and observed communications between them. The network map helps operators understand the network when identifying potential threats in advance (e.g., vulnerabilities that devices are exposed to or communication that should not exist between devices of certain roles or functional levels) and when taking corrective actions (e.g., updating vulnerable devices or isolating the network).

To build such a map, the system first analyzes captured *Network Traffic*, which is processed by a *BACnet Parser* to extract the relevant *Message Fields* from each message. The extracted fields are then sent to the two main modules.

The *Situational Awareness* module consists of an *Inventory Builder* that compiles information from the Message Fields into a list of devices with associated *Device Descriptions*; and a *Role Classifier* that assigns a role to discovered devices. Device Descriptions and Device Roles are used to enrich the network map. Every node in the map that corresponds to a BACnet device is labeled with its role and enriched with: (i) the services and objects that it supports (and that were seen in the network traffic); and (ii) information extracted from properties of its objects, such as device ID, device model, vendor ID, vendor name, firmware version, and location.

The *Role-based Intrusion Detection* module reads Message Fields and can raise alerts for malicious behavior detected when a device acts in disagreement with its assigned role. This anomalous behavior is detected when a device, e.g., uses services that are not allowed for its role or sends an unusual number of messages for its role.

In the remainder of this Section, we detail the three components introduced above: Inventory Builder (Section 3.1), Role Classifier (Section 3.2), and Role-based intrusion detection (Section 3.3).

3.1 Inventory builder

Automatic inventory building can be active, sending network messages to discover devices and their properties, or passive, analyzing the network traffic generated by the devices. The active method forces all network hosts to communicate their information, but it is invasive and may disrupt the normal process of a building or be blocked by a firewall. The passive method is noninvasive, but it is not able to detect devices and services that have no network activity [18]. We take a passive approach and analyze BACnet messages sent by the devices to extract services, objects, properties, and values that help us map the network.

Learning network connections between devices from captured traffic and representing these connections via a network map is a relatively straightforward process that has already been described in the literature [2]. Thus, we will focus on gaining information about the devices and their roles, since this is critical to enrich the network map and ultimately raise semantic alerts.

We extract the following features of BACnet devices: **(i)** Instance Number (a unique identifier for every BACnet device in the network); **(ii)** Object Name (a unique name for every BACnet device in the network); **(iii)** Vendor Name / ID (an integer uniquely identifying a vendor, which is assigned by the BACnet community); **(iv)** Model Name (assigned by the vendor); **(v)** Firmware version; **(vi)** Location (if it has been assigned by the operators); and **(vii)** the Data Link layer in use (which determines whether the device is nested or not). The identification label for a device is obtained from a combination of features (i) and (ii), while features (iii) through (vii) are descriptive information.

Features (i) to (vi) are obtained from the services sent by the devices in the BACnet Application Layer. The device instance number can be extracted from several services, such as `I-Am`, `I-Have`, `ConfirmedCOVNotification`, and `ConfirmedEventNotification`. The Vendor ID is acquired from `I-Am` and `ReadProperty` acknowledgments. The other features are obtained from replies to `ReadProperty` requests.

Feature (vii) is inferred from the BACnet Network Layer. At this layer, the source and destination bits specify the presence of a source or destination address. If those bits exist, then the BACnet/IP device sending or receiving the message is a BACnet Router, while the actual initiator or receiver is the device mentioned in the details of the BACnet Network Layer. The length of the source and destination addresses provide us with additional information about the network behind the BACnet Router. Table 1 shows the address length that devices should have according to the Data Link layer protocol they use. The Table shows two lengths shared by different protocols. The first is between BACnet/IP and Ethernet devices. This can be disambiguated since BACnet/IP addresses are identified when we parse the UDP/IP traffic. The second overlap is between MS/TP and ARCNET, which we cannot distinguish.

We also define two additional device features that have useful security implications: whether the device is a BBMD and whether it is a Foreign Device. These features are important because they indicate the possibility for a foreign BACnet device from an external network to register itself to a BBMD and be considered part of the internal BACnet network. For BBMD devices reachable from the public internet, this means that an attacker could access internal BACnet/IP devices even if they are not directly reachable via IP network, thereby circumventing network access control [3].

These two features are inferred from the BVLL layer of BACnet/IP, which includes a function code that determines the purpose of a message. When a device initiates a BACnet message with function code `BVLC-Result`, `Forwarded-NPDU`, or `Read-Foreign-Device-Table-Ack`, then that device is a BBMD. When a BACnet message is sent with the function code `Register-Foreign-Device`, or `Distribute-Broadcast-To-Network`, then the initiator is a Foreign Device, while the receiver is a BBMD.

3.2 Role classifier

BACnet devices belong to one or more standardized *Device Profiles*, according to the capabilities and services that they implement. If a device claims to belong to a certain profile, it must implement the minimal set of services that characterizes that profile. For example, all BACnet Smart Actuators need to provide the `WriteProperty` and `ReadProperty` services. Profiles are grouped into more general *Profile Families* [23]. An exhaustive list of standard profiles and profile families is shown in the first two columns of Table 2. The list of services implemented by each profile is available in [1].

BACnet has no property named, e.g., `DeviceProfile` or `ProfileFamily`. Therefore, it is not possible to directly request this information over the network. Instead, profiles are mentioned in the documentation of each device. It is common for device vendors to implement additional functionalities, sometimes pertaining to other profiles, which may or may not be included in the device's documentation [7]. Additionally, a device may provide functionalities that are not needed by the BAS. This may result in a discrepancy between documentation and operation where the observed behavior of a device is different from what is suggested by its documentation.

BACnet device profiles are too granular and redundant for situational awareness and intrusion detection (e.g., we do not usually need to distinguish between

Table 1. Address length (in bytes) for BACnet variants

Data Link layer	Destination length	Source length
BACnet/IP or Ethernet	6	6
MS/TP or ARCNET	1	1
LonTalk	2	2
LonTalk unique Neuron ID	7	2
ZigBee	3	3

a controller and a lighting controller), while profile families are built around the target application domain. Therefore, we classify devices using neither the standard profiles nor the profile families, but their functional roles. These roles are defined in a building automation context; in other operational settings (e.g. Industrial Control Systems, smart grids), the number and type of these roles might vary. We follow the BAS functional levels mentioned in Section 2 (namely, field, automation, and management), to which we add a fourth *routing level*, which contains devices that maintain the network infrastructure (e.g., routers and gateways). To stress the fact that this division is based on the actual behavior, instead of a device belonging to a level we say that it belongs to (assumes) a role. As these roles fundamentally differ in function, they also differ in the network behavior that we can observe. This leads to the following list of device roles (also shown in the third column of Table 2):

Table 2. Device Profiles, Profile Families, and Roles usually associated with them.

BACnet Device Profile	**BACnet Profile Family**	**Behavioral Role**
Adv. Operator Workstation	Operator Interfaces	Workstation
Operator Workstation		
Operator Display		
Building Controller	Controller	Controller
Adv. Application Controller		
Application Specific Controller		
Smart Actuator		Field Device
Smart Sensor		
Adv. Lighting Workstation	Lighting Operator Interfaces	Workstation
Lighting Operator Display		
Adv. Lighting Control Station	Lighting Control Stations	Controller
Lighting Control Station		
Lighting Supervisor	Lighting Controllers	
Lighting Device		Field Device
Adv. Life-Safety Workstation	Life-Safety Operator Interfaces	Workstation
Life-Safety Workstation		
Life-Safety Annunciator Panel		Field Device
Adv. Life-Safety Controller	Life-Safety Controllers	Controller
Life-Safety Controller		
Adv. Access Control Workstation	Access Control Operator Interfaces	Workstation
Access Control Workstation		
Access Control Security Display		
Adv. Access Control Controller	Access Control Controllers	Controller
Access Control Controller		
Router	Miscellaneous	Router
Gateway		
Broadcast Management Device		
Access Control Door Controller		Field Device
Access Control Credential Reader		

Routers are used to interconnect building automation devices from two or more networks, which may differ in the Data Link layer or Application layer.
Workstations are used at the management level to store historical data of building processes, to inform operators about the states of building components, and to adjust setpoints in Controllers.
Controllers are used for automation level tasks; they contain and execute the main logic processes that govern the BAS. They mainly interact with other roles by reading/writing property values.
Field Devices interact with the physical environment at the field level. They are connected to Controllers in two possible ways: either as simple inputs and outputs, in which case they might not be visible on BACnet networks, or as smart devices that implement BACnet and can communicate with other devices by using BACnet services.

The topology of the BAS internetwork (see Section 2) influences the behavior that we can observe. When monitoring only one network (e.g., the backbone IP network), it's not possible to observe the messages exchanged locally within another network (e.g. between Controllers and Field Devices on a control network). Since cost and performance constraints can sometimes prevent the monitoring of control networks, Field Devices are typically harder to discover and classify.

We propose a two-step approach to classify devices. First, we apply heuristics to identify roles based on the services observed over the network. Second, since heuristics might never be matched if a device does not send or receive the right services, we apply a similarity-based classification comparing known and unknown devices.

Heuristics-based classification (HBC). The HBC assigns a role to network devices by applying heuristics, derived from the BACnet specification, to features of the network layers of a BACnet message. We assume that devices do not actively spoof their behavior as that of another role, but note that the use of consistency rules constraining the values of observed properties (such as what is described in Section 3.3) could mitigate this issue.

The BACnet Network Layer contains information for the classification of BACnet Routers. A device can be classified as a Router in two ways: i) when it sends a BACnet `I-Am-Router-To-Network` message; and ii) from the messages exchanged between BACnet/IP and non-BACnet/IP devices. In these messages, a destination or source specifier determines the network number to which the device belongs. As a result, we know that the IP initiator or receiver of the message is a BACnet Router that serves the network of the nested device.

The services in the messages of the Application Layer allow us to classify Controllers and Workstations. We look for services and behaviors exclusive to a certain BACnet profile, and then label the devices performing those services with the associated role. As an example, when a device initiates a `WritePropertyMultiple` service, it is classified as a Workstation, because only Workstations should initiate that service. By the same principle, the device that executes a `WritePropertyMultiple` request and responds with a `Simple-ACK` is classified as a Controller. When observing some services like in the example

above, it is possible to classify both the source and destination devices; in other cases this is not possible, and only one device is classified. For example, the `ReadPropertyMultiple` service is always executed by Controllers but may have different clients.

The services that are always initiated by Workstations and executed by Controllers are: `WritePropertyMultiple`, `AcknowledgeAlarm`, `GetEventInformation`, `ReadRange`, `GetEnrollmentSummary`, `DeviceCommunicationControl`, `TimeSynchronization`, `ReinitializeDevice`, `AtomicReadFile`, and `AtomicWriteFile`. The services that are always initiated by Controllers and executed by Workstations are: `ConfirmedEventNotification` and `UnconfirmedEventNotification`. The services that are always initiated by Workstations but have different servers are: `GetAlarmSummary`, `CreateObject`, and `DeleteObject`. The `ReadPropertyMultiple` service is always executed by Controllers but may have different clients. Other services may be initiated or executed by different profiles, so they are not used in the classification.

This approach is adequate when devices strictly follow the BACnet specification. This is not always the case, since the specification also allows vendors to extend the set of services supported by their devices, and as such the behavior of a device can be matched by the HBC to more than one role. To mitigate this problem, we can extend the module with a learning phase. We assume that a device sends more messages with services belonging to the correct role than services of other roles. When the learning phase is over, the median number of messages sent per service by each device is calculated and compared against a threshold k; if a device initiates (or executes) a service exclusive to a role less than k times, then that service is ignored in the classification.

The two different versions of HBC (respectively, without a learning phase and with one) have benefits and limitations. The first approach immediately classifies a device when an exclusive service is observed; however, if the first observed message conveys a service belonging to the wrong role, the device is misclassified. On the other hand, the second approach classifies devices with a delay, but the results tend to be more accurate.

Distance-based classification (DBC). When the HBC cannot classify a device, the DBC tries to do so by using similarities with known devices in the network. The DBC measures similarity using three features of a device d obtained by the inventory builder in Section 3.1: Vendor ID (an integer), the vendor-specific Model Name (a string of characters), and Data Link Layer type (a category), here denoted as $f_{\mathbf{iii}}(d)$, $f_{\mathbf{iv}}(d)$ and $f_{\mathbf{vii}}(d)$.

To measure the distance between two devices d_i and d_j using the Data Link Layer type, we use a discrete distance:

$$r_{DLT}(d_i, d_j) = \begin{cases} 0 & \text{, if } f_{\mathbf{vii}}(d_i) = f_{\mathbf{vii}}(d_j) \\ 1 & \text{, otherwise.} \end{cases} \tag{1}$$

The distance using the two vendor-specific features is defined as:

$$r_V(d_i, d_j) = \begin{cases} 1 & \text{, if } f_{\mathbf{iii}}(d_i) \neq f_{\mathbf{iii}}(d_j) \\ L\left(f_{\mathbf{iv}}(d_i), f_{\mathbf{iv}}(d_j)\right) & \text{, otherwise.} \end{cases} \quad (2)$$

$L(x, y)$ is the Levenshtein distance, which sums the number of insertions, deletions, and substitutions needed to convert one string to another. We ignore identical words in different positions and delete spaces in the Model Name string before calculating r_V. We then normalize the resulting distance into the range [0,1].

We compute the total distance between two devices as $r(d_i, d_j) = a \cdot r_{DLT}(d_i, d_j) + b \cdot r_V(d_i, d_j)$, with $a < b$, since devices of the same vendor and model are more likely to have the same role.

For each unknown device d_i, the DBC loops through all classified devices d_j and computes $r(d_i, d_j)$. Then, it performs a variant of the k-Nearest Neighbors classification algorithm: given a threshold distance $\bar{r}$, it keeps three separate counters of all classified Controllers, Routers and Workstations that are "close" to d_i, i.e. those devices d_j where $r(d_i, d_j) < \bar{r}$. Finally, the module selects the greatest of the three counters and checks whether it is greater than a minimum support level selected according to the network size (e.g., 2 for a small network, 10 for a larger one). The role whose counter fulfills the conditions above is assigned to the device; if no role fulfills the conditions (i.e. if there is no greatest evidence counter because of a tie or if the counter is lower than the minimum support), the device remains unclassified and an alert may be generated to inform the network operator that an unidentified device exists in the network. This classification carries a little uncertainty and to avoid error propagation, devices that are classified by DBC are not taken into account to classify another device.

3.3 Role-based intrusion detection

Effective intrusion detection requires, in addition to good detection rate at a low false positive rate, alerts that are meaningful and provide enough information and context to make them actionable. The content of the communication, and the available context from situational awareness, enable building an effective semantics-aware intrusion detection system. Here, we illustrate how device features and values observed and inferred from the network can be used to build a semantic intrusion detection model, in the style of the white-box framework [5].

White-box intrusion detection. A white-box detection model consists of a set of rules that classify network traffic (single messages or groups of messages) as normal or anomalous. These rules are expressed in *features* that capture relevant properties of the traffic and devices, such as the properties observed by the inventory builder (see Section 3.1). We consider *numerical* features (e.g., integers), *nominal* features (e.g., names), and *compound features*, which are tuples of other features. Rules may be specified, for example, as a whitelist of allowed services, or *consistency* rules between features, e.g., a device cannot have a Router role and a vendor different than 'Contemporary Controls' in a specific network. Rules can also be learned from 'normal behavior' captured in a training set.

Learning normal behavior, so that deviations represent *attacks* (or other conditions of interest, e.g., malfunctioning devices), can be difficult because normal traffic and attacks may have the same values on some features. However, if we choose the right features, attacks will have values that are rare among normal traffic on at least one of the features. For features that can take many different values, such as numeric features, individual values may be rare even for normal traffic. Yet rare values should indicate that a message is an attack. Therefore, we group values into *bins* and consider the occurrence of bins rather than individual values. For nominal features, we usually consider the binning where each bin consists of a single element. For numeric features, we usually consider bins that are ranges $[v_l, v_u)$. For compound features, we consider bins that are the product of such bins. The fundamental assumption in learning is that attack traffic will lead to features yielding *anomalous bins*: bins that have a low probability of occurring among normal traffic.

Our main goal is to find anomalies along with their causes (the features yielding anomalous values) so they can be presented to an operator who can then evaluate and address them. As discussed above, a role is an important indicator of how a device should behave in a BACnet network. Below, we illustrate how a role-based white-box framework can be defined. We focus on role-based features that exploit the situational awareness information to detect the (otherwise hard to find) types of attacks mentioned in Section 4.3. This could easily be combined with additional features that have already proven to be effective in detecting more general types of attacks in different settings [5, 9, 21].

The role-based detection model considers both specified and learned rules (i.e. features with a labeling of their bins as normal or anomalous). The former capture the type of services that devices in a certain role offer. For this feature, we specify the normal values (bins) based on the BACnet specification. Notice that this specification can be ported across networks. We also learn rules for features that capture the number of messages (over a fixed period of time) that devices in an role send and/or receive, in total or for a specific service. These features are specific to a network and the normal bins are learned from training data from this particular network.

Specification: types of messages. The BACnet specification [1] restricts the set of services that a device should be able to use or offer depending on its BACnet profile. Since behavior that does not match a device's role can indicate that it is compromised, we define, for each role, a whitelist (set of normal bins) of allowed services. For those devices that were assigned a role by the classifier in Section 3.2, the detector checks this whitelist against the observed messages. For example, a Controller sending a `WritePropertyMultiple` request will raise an alert, as only a Workstation should initiate that service. The IDS operator can update the whitelist as needed, for example to address services added by vendors or in reaction to false positives during detection.

Learning: number of messages. We expect the frequency of messages during normal operation to be consistent over time; i.e. it should be close to frequencies observed during training. Frequency is a feature of sequences of messages rather

than individual messages. To measure it, we group messages into fixed-length time intervals and count the number of messages of interest within each interval. We use a feature of the message to define whether it is of interest. For example, messages of interest that have value (s, d) on compound feature $(m.service, m.source)$ are used to count how often a device d sends a message pertaining to a service s. We will use $f_{(s,d)}$ to denote this feature. Thus, for $[m_1, \ldots, m_n]$ messages in a time interval, this gives:

$$f_{(s,d)}([m_1, \ldots, m_n]) = \#\{i \in \{1, \ldots, n\} \mid m_i.service = s \wedge m_i.source = d\}.$$

We can also look at the frequency with which a service is called, irrespective of the calling device, which we expect to be proportional to the number of potential clients. Thus, we introduce the feature f_s given by:

$$f_s([m_1, \ldots, m_n]) = \#\{i \in \{1, \ldots, n\} \mid m_i.service = s\}/\#D$$

where s is a service and D is the current set of all devices discovered. Note how we take the average frequency of calls per (potential) client device by dividing the total number of calls by the number of devices.

Our detection model is composed of a 'normal bin' for each frequency: to set this interval, we use the range of frequencies observed among all time intervals in the training data. To reduce false positives, we further extend this range by a tolerance of 5%. We also expand our detection model by using the roles learned during classification: specifically, we create a range for the frequency of service use per each role. For each service s and each role r, we define the bounds of that range as the minimum (maximum) of the bounds of all the ranges for $f_{s,d}$, where d are all the devices with role r. In the detection phase, we collect messages over time intervals of the same length and compute the features above to check whether they fall in the normal range (bin). Suppose that d is a new device detected on the network, and that a normal range of values has not yet been learned for $f_{(s,d)}$. We then perform detection using the range of the service for devices in role r if d has been classified to role r, or that of f_s (i.e. the normal range of the service) if d has not been classified to a role. In this way, we can always check $f_{(s,d)}$ against a reasonable range for any $d \in D$.

4 Validation

We implemented our system on top of SilentDefense[5], a network monitoring and IDS tool developed by Forescout. The network monitoring component has a built-in BACnet dissector and parser: the parser provides the extracted fields to the Deep Protocol Behavior Inspection engine of SilentDefense, which allows a network operator to see all BACnet message details. The role-based classification and intrusion detection modules were written in Lua on top of this engine. Figure 2 shows, on the left, an example network map with a Router, a Controller,

[5] https://www.forescout.com/platform/silentdefense/

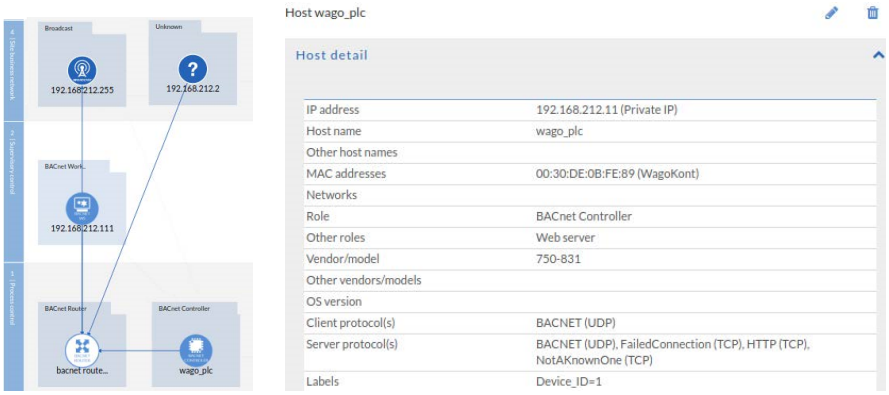

Fig. 2. Network Map (left) and Device Description (right)

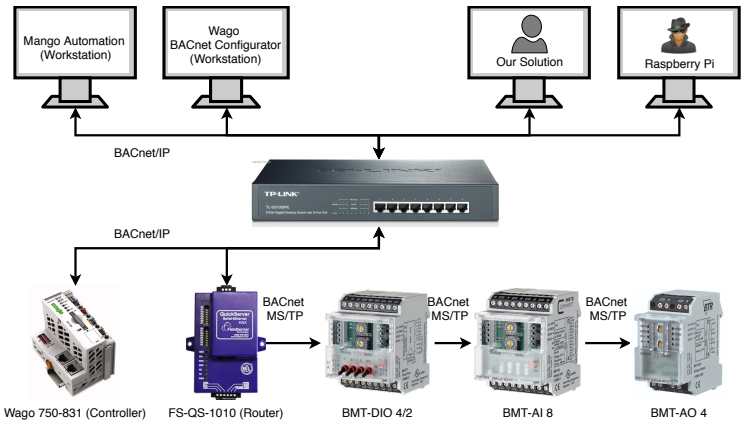

Fig. 3. Network diagram of the BACnet Lab.

a Workstation, and an Unknown device, which could not be classified. The same Figure shows, on the right, the available properties of a selected controller node.

To validate our approach, we used a realistic lab facility and real-world data (Section 4.1) to test device identification and role classification (Section 4.2) as well as intrusion detection (Section 4.3).

4.1 Datasets

Dataset 1 comes from 10 minutes of traffic in our BACnet Lab, which is a simulation environment containing real devices. The scenario implemented in this lab is a small building with motion and temperature sensors that send signals to controllers in order to switch lights and fans on or off.

Figure 3 depicts the following devices involved in the lab. The main building controller (Wago 750-831) implements the system logic by reading and writing

inputs and outputs of the I/O modules. A BACnet Router (FS-QS-1010) connects one IP network with one MS/TP network. Three devices connected with RS485 communicate via BACnet MS/TP (Metz Connect BMT I/O modules). A digital I/O reads and writes digital inputs and outputs, such as motion sensors, light bulbs or fans (with two states: on/off), whereas analog I/O modules read and write analog inputs and outputs, such as temperature sensors, dimmable LEDs, and fans with different speeds. A motion sensor, a LED bulb, and a fan are connected to a BMT-DIO4/2 module, while a temperature sensor is connected to a BMT-AI8 module. A BACnet Workstation can configure devices in the network using the Wago configuration software. A second BACnet Workstation monitors the lab and lets users modify setpoints using the Mango Automation software. Finally, a Raspberry Pi is connected to the BACnet/IP network via the bacpypes[6] Python library. This device is only used for validating the intrusion detection module, and is not included in the role classification results.

Dataset 2 comes from a real BACnet network from the campus of a Dutch university. We analyzed 9 days of traffic, totaling 106GB of data and around 20 million BACnet messages. We did not have access to the topology of the real network and the profiles of the devices. To validate our results, we extracted information from the network traffic, such as vendor and model names, and we were able to identify most devices by searching their profiles in vendor websites. Furthermore, this dataset is only limited to the traffic observed on the IP backbone network: thus, as mentioned in Section 3.2, no Field Devices could be discovered.

4.2 Classification results

We evaluated the effectiveness of discovery of devices and of their classification into roles using Datasets 1 and 2. We also compared the use of only HBC against the use of both HBC and DBC. These classification steps are described in Section 3.2. Notice that for HBC we described two methods: immediate classification, and classification at the end of a learning phase to address devices that extend the BACnet specification. In all our experiments, the results were the same for both methods, so we report only the results of HBC versus HBC+DBC. When choosing the DBC parameters, we found good results with a=0.35, b=0.65 and $\bar{r}$=0.3.

Table 3 shows the devices classified by HBC and HBC+DBC. In Dataset 1, both methods gave the same results, classifying 7 devices but being unable to assign a role to the Mango Automation virtual machine. This is because the virtual machine acted as an HMI rather than a workstation (it read and wrote values to the controller, and displayed them to the user, without any other complex activities). All the classified devices were assigned the correct role. To test whether our DBC method was working as intended, we removed the role of one of the I/O modules from the results of HBC. We then applied DBC to this unclassified device; the method correctly computed the other two I/O modules as being the "closest" to the unclassified device, and assigned it their role.

[6] https://github.com/JoelBender/bacpypes

Table 3. Classification results for HBC and DBC

Dataset 1 (both methods had the same results)

Role	Ground truth	Classification	TP	FP
Controller	4	4	4	0
Router	1	1	1	0
Workstation	2	1	1	0
Total	8	7	7	0

Dataset 2		**HBC**			**HBC + DBC**		
Role	Ground truth	Classification	TP	FP	Classification	TP	FP
Controller	219	213	212	1	220	219	1
Router	21	21	21	0	21	21	0
Workstation	1	0	0	0	0	0	0
Total	241	234	233	1	241	240	1

In Dataset 2, the system discovered 241 devices for which we were able to find a ground truth via manual classification. It found 4 additional devices for which we could not establish a ground truth, because we did not have any information about their vendor or model name. Thus, we excluded them from validation. Using either HBC or HBC+DBC, 3 of these devices are classified as Workstations and 1 is not classified. Of the 241 devices in our data, 39 are BACnet/IP devices and 202 use other BACnet variants. HBC was able to assign a role to only 234 of these devices; all the unclassified devices were Controllers whose traffic was not exclusive to any role. Instead, HBC+DBC managed to assign a role to all discovered devices. Since a learning period for HBC was not needed, the overall classification step was considerably fast (<1h of network traffic was used).

Both methods had over 99% classification accuracy: all the Routers and all the classified Controllers were assigned the correct role. One Workstation was misclassified as a Controller, because it executed an exclusive service for Controllers (`ReadPropertyMultiple`), without showing any behavior exclusive to Workstations.

4.3 Intrusion detection results

We evaluated the attack detection capabilities of our role-based intrusion detection module, and measured how many false positive alerts (FP) it raised on legitimate traffic. We have already demonstrated the feasibility of our white-box anomaly detection approach in a smart building setting [8], giving good trade-offs between false positives and detection rate when applied on simulated cases as well as on data from real-world networks. To expand upon this state of the art, we validated our approach on the same testbed and datasets used in [8] and described in Section 4.1. To showcase the importance of behavioral roles in helping intrusion detection, we implemented the following synthetic attacks, in addition to those already presented in [8].

All the attacks were launched using a Raspberry Pi. The Pi represents a device that has been compromised by an attacker, e.g. as shown by [3], has been recently added to the BACnet network, and is being used as an entry point to send malicious messages.

Snooping. We simulate a reconnaissance attack launched from a compromised Controller. To model this, we forcibly classify the Pi as a Controller during the role classification step. We broadcast a `Who-Is` request to retrieve the address and instance number of all devices in the network, and then send them a series of `ReadProperty` requests to read their model name, vendor ID, and supported capabilities. Note that a Controller is expected to be able to send `Who-Is` and `ReadProperty` requests: what makes the behavior anomalous, is the large frequency of such requests, i.e. the feature $f_{(s,d)}$ described in Section 3.3. Moreover, as the device is new and therefore there is no 'normal bin' learned for that feature, our detection falls back on the comparison between $f_{(s,d)}$ and the frequency for other Controllers in the network.

Tampering. We simulate a tampering attack launched from a compromised Field Device. To model this, we forcibly classify the Pi as a Field Device as before. We send a `WriteProperty` request to the Metz Connect module governing the light bulb, turning the bulb off. In [8], this attack could not be detected, as the system only analyzed the request itself, and considered it legitimate. For a role-based IDS, instead, this behavior violates the specified whitelist for Field Devices, since they are restricted from initiating `WriteProperty` services.

Both attacks caused the Role-based Intrusion Detection module to raise alerts.

To evaluate the usability of our intrusion detection, we split Dataset 2, containing approx. 9 days of network traffic, into 172 hours for training and approx. 47 hours for testing. This choice of splitting is motivated by the need to learn a full week's worth of data during the training, to include time-driven behaviors that might occur only on certain days. We followed the work of [17] and computed both the total number of FP and the average rate of FP per hour which were raised by the two detectors described in Section 3.3. We obtained:

- 0 FP (0 FP/h) for the specification-based types of messages detector;
- 304 FP (around 6.4 FP/h) for the learning-based number of messages detector.

We did not obtain any FP for the specification-based detector because the behavior of all devices was consistent throughout the dataset; that is, once a device was classified as belonging to a particular role, it kept behaving as appropriate for that role. The performance of the learning-based detector is good when using the default range tolerance of 5%; depending on the criticality of the monitored traffic, raising the tolerance can further reduce the amount of FP.

Additionally, we evaluated whether our intrusion detection approach could adapt well to new devices appearing on the network, without learning a model of their behavior first. To do so, we modified the above experiment to simulate the appearance of a new device, effectively performing leave-one-out cross validation. For each one of the 241 classified devices, we ran a separate training phase excluding all information from that device. Then, we included the same device in

the testing phase, comparing its frequency values with the generic ranges learned for its role. Finally, we measured the increase in FP: on average, the number of FP rose to 310 (around 6.5 FP/h), not impacting significantly the FP rate of the overall system. This demonstrates the adaptability of our role-based approach. The increase in alerts during cross-validation is explained by a small amount ($< 5\%$) of devices whose behavior is significantly different from all other devices with the same role. As expected, when such devices are not taken into account when building the range of values in the training phase, their 'unique' behavior is detected as anomalous. We deem this small amount of devices as acceptable; if a large number of devices would show different behavior for the same role, it might be sensible to develop a more fine-grained classification into 'sub-roles'.

5 Conclusions and future work

We proposed an approach that parses the network traffic of a building automation system to achieve three goals: (i) the discovery, characterization, and role assignment of devices in the network; (ii) role-based intrusion detection; and (iii) the creation of a dynamic network map to increase situational awareness.

By observing, parsing, and interpreting network messages, we extract useful information about the devices, build a network map to provide operators with details about their system, and detect attacks. Once an attack is detected, we generate alerts that include semantic information helpful to the operators. We implemented and validated our approach on real and simulated datasets.

As future work, we intend to deploy our solution on real operational environments as part of SilentDefense; to extract more semantic information about devices, possibly with the use of an ontology (e.g., an object with degree units is a temperature sensor); and to develop heuristics to further refine the roles.

References

1. ASHRAE: BACnet - a data communication protocol for building automation and control networks. Standard (2016)
2. Becker, R., Eick, S., Wilks, A.: Visualizing network data. IEEE Transactions on Visualization and Computer Graphics **1**(1), 16–28 (1995)
3. Brandstetter, T., Reisinger, K.: (in)security in building automation how to create dark buildings with light speed. In: Blackhat (2017)
4. Caselli, M., Zambon, E., Amann, J., Sommer, R., Kargl, F.: Specification mining for intrusion detection in networked control systems. In: 25th USENIX Security Symposium. pp. 791–806 (2016)
5. Costante, E., den Hartog, J., Petković, M., Etalle, S., Pechenizkiy, M.: A white-box anomaly-based framework for database leakage detection. Journal of Information Security and Applications **32**, 27–46 (2017)
6. Domingues, P., Carreira, P., Vieira, R., Kastner, W.: Building automation systems: Concepts and technology review. Computer Standards & Interfaces **45**, 1–12 (2016)
7. Esquivel-Vargas, H., Caselli, M., Peter, A.: Automatic deployment of specification-based intrusion detection in the BACnet protocol. In: Proceedings of the 2017 Workshop on Cyber-Physical Systems Security and PrivaCy. pp. 25–36 (2017)

8. Fauri, D., Kapsalakis, M., dos Santos, D., Costante, E., den Hartog, J., Etalle, S.: Leveraging semantics for actionable intrusion detection in building automation systems. In: Critical Information Infrastructures Security. pp. 113–125 (2019)
9. Fauri, D., dos Santos, D., Costante, E., den Hartog, J., Etalle, S., Tonetta, S.: From system specification to anomaly detection (and back). In: Proceedings of the 2017 Workshop on Cyber-Physical Systems Security and PrivaCy. pp. 13–24 (2017)
10. Holmberg, D.: BACnet wide area network security threat assessment. Tech. rep., NIST (2003)
11. Johnstone, M., Peacock, M., den Hartog, J.: Timing attack detection on BACnet via a machine learning approach. In: Proceedings of the 13th Australian Information Security Management Conference. pp. 57–64 (2015)
12. Kastner, W., Neugschwandtner, G., Soucek, S., Newman, H.M.: Communication systems for building automation and control. Proceedings of the IEEE **93**(6), 1178–1203 (2005)
13. Mundt, T., Wickboldt, P.: Security in building automation systems - a first analysis. In: International Conference On Cyber Security And Protection Of Digital Services. pp. 1–8 (2016)
14. Pan, Z., Hariri, S., Al-Nashif, Y.: Anomaly based intrusion detection for building automation and control networks. In: IEEE/ACS 11th International Conference on Computer Systems and Applications. pp. 72–77 (2014)
15. Sommer, R., Paxson, V.: Outside the closed world: On using machine learning for network intrusion detection. In: IEEE Symposium on Security and Privacy. pp. 305–316 (2010)
16. Tonejc, J., Guttes, S., Kobekova, A., Kaur, J.: Machine learning methods for anomaly detection in BACnet networks. Journal of Universal Computer Science **22**(9), 1203–1224 (2016)
17. 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: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 1092–1105 (2016)
18. Webster, S., Lippmann, R., Zissman, M.: Experience using active and passive mapping for network situational awareness. In: 5th IEEE International Symposium on Network Computing and Applications. pp. 19–26 (2006)
19. Wendzel, S., Tonejc, J., Kaur, J., Kobekova, A.: Cyber Security of Smart Buildings, chap. 16, pp. 327–351. John Wiley & Sons (2017)
20. Yüksel, O., den Hartog, J., Etalle, S.: Reading between the fields: Practical, effective intrusion detection for industrial control systems. In: Proceedings of the 31st Annual ACM Symposium on Applied Computing. pp. 2063–2070 (2016)
21. Yüksel, Ö., den Hartog, J., Etalle, S.: Towards useful anomaly detection for back office networks. In: Information Systems Security. pp. 509–520. Springer (2016)
22. Zheng, Z., Reddy, A.: Safeguarding building automation networks: THE-driven anomaly detector based on traffic analysis. In: 26th International Conference on Computer Communication and Networks. pp. 1–11 (2017)
23. Ziegenfus, S.: BACnet® is in a "family way". ASHRAE Journal **58**(9), 100–102 (2016)