Specification Mining for Intrusion Detection in Networked Control Systems

Marco Caselli
University of Twente
m.caselli@utwente.nl

Emmanuele Zambon
University of Twente & SecurityMatters B.V.
emmanuele.zambon@secmatters.com

Johanna Amann
ICSI
johanna@icir.org

Robin Sommer
ICSI & LBNL
robin@icir.org

Frank Kargl
Ulm University
frank.kargl@uni-ulm.de

Abstract

This paper discusses a novel approach to specification-based intrusion detection in the field of networked control systems. Our approach reduces the substantial human effort required to deploy a specification-based intrusion detection system by automating the development of its specification rules. We observe that networked control systems often include comprehensive documentation used by operators to manage their infrastructures. Our approach leverages the same documentation to automatically derive the specification rules and continuously monitor network traffic. In this paper, we implement this approach for BACnet-based building automation systems and test its effectiveness against two real infrastructures deployed at the University of Twente and the Lawrence Berkeley National Laboratory (LBNL). Our implementation successfully identifies process control mistakes and potentially dangerous misconfigurations. This confirms the need for an improved monitoring of networked control system infrastructures.

1 Introduction

Starting from Denning's seminal work in 1986 [9], *intrusion detection* has evolved into a number of different approaches. Among them, *anomaly-based intrusion detection* and, most recently, *specification-based intrusion detection* have gained attention for their potential to detect previously unknown attacks (e.g., zero-day attacks).

A specification-based intrusion detection system (IDS) leverages functional specifications of a system to model its properties, or *features*, creating a reference of correct behavior. Differently from anomaly-based IDSs, behavior of features is not derived by a learning phase (prone to false positives) but directly extracted from documentation. This ensures the quality of the generated models and improves detection. Several research efforts in the literature confirm the accuracy of specification-based intrusion detection [62]. However, these approaches assume manual (human) analyses, often focused just on network protocol documentations.

Scaling a specification-based approach to an entire infrastructure faces three key challenges. First, targeted systems do not always have consistent features, and thus *constraints*, that allow to describe functional rules (e.g., common networks do not usually guarantee stable communication patterns or message timing). Second, even when such constraints are present, bridging the semantic gap between infrastructure properties and the low-level features actually observable by an IDS remains hard [22]. Third, deploying a specification-based intrusion detection requires an explicit and unambiguous description of the features' behaviors, as well as substantial human effort in crafting the related specification rules.

This work aims to fill this last challenge by automating specification-based intrusion detection to a fairly high degree. We propose an approach to automatically mine IDSs' specification rules from available documentation. Our approach works under the following assumptions:

- Documentation about monitored systems must be *available*. No specification-based intrusion detection would be possible without relying on correct information that describes an infrastructure's components, mechanisms and constraints. Documentation should be provided in an electronic form to allow automated knowledge extraction.
- Information retrieved from the documentation must be *linkable* to what an IDS can observe. On a network, information about components, mechanisms and constraints need to map to features monitored by the IDS (e.g., information on a whitelist of network services should link to the correct IP address).

We do not claim that these principles can be generically applied to any system. However, we observe that environments such as Networked Control Systems

(NCSs) [20] suit this approach. NCSs are "systems whose constituents such as sensors, actuators, and controllers are distributed over a network, and their corresponding control-loops are formed through a network layer" [30]. Examples of NCSs include: industrial control systems [65], building automation systems [42] and in-vehicle networks [31]. NCSs generally have the properties discussed above. Communications over these networks are quite stable [22] (e.g., neither the number of devices nor the way data is shared change regularly). Moreover, automation guarantees the presence of control algorithms and consequently the existence of consistent features that will eventually become the core of the specification rules. These features and their constraints also represent the most attractive target for an attacker who wants to manipulate the controlled processes [14].

The two assumptions defined above hold for NCSs. Information related to system features are often documented in configuration files, reference books and manuals (e.g., "Substation Configuration Language" files for industrial control systems, "Protocol Implementation Conformance Statement" for building automation systems, CAN matrixes and corresponding documentation for in-vehicle networks). Linkability is generally guaranteed by the absence of encryption and by the verbosity of the adopted protocols.

In this paper, we design and discuss a specification-based network intrusion detection system (NIDS) for BACnet-based building automation systems to demonstrate and investigate our concept. First, we present specification-based intrusion detection in §2. Then, we introduce building automation and BACnet in §3. We outline our approach in §4. Details of the approach in the context of building automation systems are discussed in §5 to §7. Finally, we examine the general applicability of our work in §8.

2 State of the Art

Ko et al. introduce specification-based intrusion detection in [34]. The authors describe their approach towards automated detection of Unix privileged program misuses and suggest to "specify programs' intended behavior" by modeling their normal execution beforehand. The proposed solution works through the definition of a Program Policy Specification Language aiming to formally define programs' operations by simple predicate logic and regular expressions. Later works such as [13] resume and improve the proposed ideas. Ko et al. improve their introductory research in [35] by defining a formal framework used to define and detail security-relevant behavior of Unix programs. In [33], the framework gets integrated into a comprehensive specification-based IDS, called SHIM. SHIM merges several different detection approaches that apply to both network communications and operating system activities. Its use, together with machine learning techniques, was shown to be an effective solution towards the development of automated intrusion response strategies [3]. From the previous works, Sekar et al. continue developing the research field by proposing complementary approaches in [52] (based on the use of the Auditing Specification Language) and [54] (based on a custom language called "Regular Expressions for Events" or REE). The same authors, present in [53] a hybrid approach aiming to increase the information of a specification-based IDS with the use of anomaly-based intrusion detection techniques. The authors use Extended Finite State Automata (EFSA) to model and detail network protocol behavior. Then they refine the set of monitored features via a learning phase exploiting statistical analyses on the traffic traces.

Over the years, researchers customized specification-based intrusion detection to fit different infrastructures such as mobile ad hoc networks [60, 44, 23, 56, 61, 19] and WLANs [16]. Furthermore, specification-based IDSs were developed for specific use cases both for network-based (e.g., VoIP technologies [59], carrier Ethernet [26]) and host-based security (e.g., kernel dynamic data structures [48], mobile operating systems [7]).

Specification-based intrusion detection has gained a main role in NCSs. Works such as [8], [27], and [37] present specification-based IDSs for Modbus, Zigbee and DNP3 respectively. Hadeli et al. notably apply a semi-automated specification approach to substation automation systems employing MMS and GOOSE [21]. The authors leverage operator input to parse infrastructure-related documentation and derive security checks. Ultimately, Berthier et al. take this approach a step forward by modeling not just employed protocols (in this case C12.22) but smart-meter security constraints and policies as well [4]. This research shows the feasibility and effectiveness of modeling high-level infrastructure properties.

State-of-the-art research on specification-based intrusion detection assumes that protocol and system documentation is readily available when designing and configuring the IDS. Few thoughts are spent on where and how to retrieve this information, especially not in an automated way. Moreover, all the aforementioned works do not explore *the possibility of autonomously extracting the information needed to build the related IDSs from documentation*, instead relying on human evaluation and translation to rules. This possibility would allow to efficiently apply specification-based approaches to entire infrastructures. As discussed in the introduction, our research focuses on this key aspect of specification-based intrusion detection with the aim of making its development more time-effective and accurate.

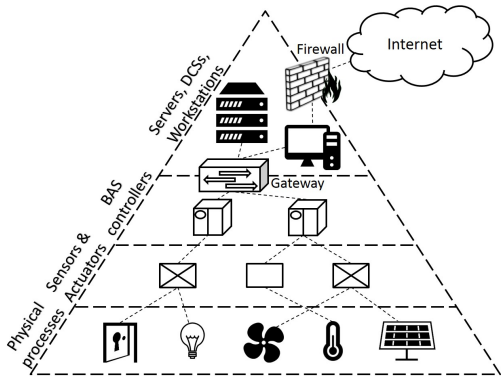


Figure 1: Building automation network layout

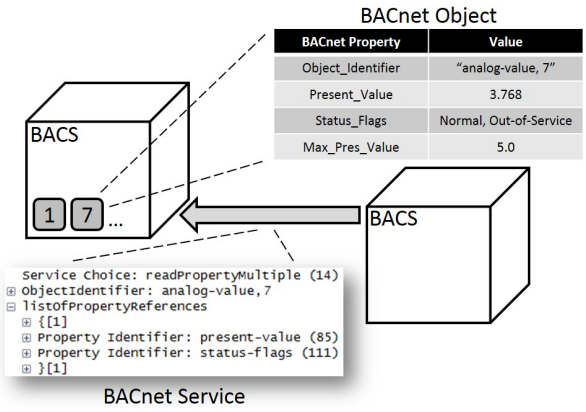


Figure 2: BACnet interaction example

3 Case Study: Building Automation

Building automation systems (BASs) or building management systems (BMSs) are networked infrastructures controlling operations and services within a building (or a group of buildings). Among other uses, building automation systems can monitor and control HVAC (heating, ventilation, and air conditioning), lighting, energy consumption, and physical security and safety [42].

A building automation network usually follows a hierarchical layout [28] (Figure 1). At the bottom, sensors and actuators directly connect to the monitored physical processes and send information back and forth to building automation controllers. Controllers communicate with servers and distributed control systems (DCSs) to coordinate high-level control procedures and policies. Finally, operators can access and manage building automation components connecting through their workstations and human-machine interfaces (HMIs).

In the last decade, the employment of building automation solutions has constantly increased (both for commercial and residential buildings) and its market share is expected to grow in the following years [39]. Despite numerous benefits (e.g., energy efficiency, "smart homes", etc.) building automation makes several new threat scenarios not just feasible but realistic [24, 17, 18, 47]. Nevertheless, only few solutions have been proposed to improve building automation system security [18, 6, 45].

3.1 BACnet

The "Building Automation and Control Network" (BACnet) protocol [1] facilitates building automation system communication for a wide array of different devices and different settings. While exact statistics of the proliferation of BACnet are difficult to come by, already back in 2003 there were more than 28,000 BACnet installations in 82 countries [28].

BACnet has a layered protocol architecture, similar to the ISO/OSI model. The BACnet protocol has an application layer, containing the actual application data payload as well as a network layer that abstracts the differences of the network architectures supported by BACnet and implements its own routing protocol. Underneath the network layer, BACnet also specifies how it can be used with different types of data links.

The BACnet application layer rests on two important core concepts: *objects* and *services* (Figure 2). A Building Automation and Control System (BACS)[1] includes one or more BACnet objects that are used to represent its functions. Objects are of a specific type, like `Analog Input` or `Analog Output`. BACnet supports a wide range of high-level object types like `Calendars`, `Date Value` objects, or `Credential Data Input` objects. BACnet users and vendors can define "proprietary" objects as well to serve specific functionalities. Object types have different attributes that are called properties, which are extensible for specific purposes. The second core concept of the BACnet application layer are services. While objects describe the different functions that are implemented by a BACS, services define how to communicate with the BACS, offering functionality such as reading object information from a device. Their names reflect the semantics of the operation (e.g., `ReadProperty`).

Individual BACSs typically only support a small selection of possible objects and services. Manufacturers use a "Protocol Implementation Conformance Statement" (PICS) to describe which objects and services are implemented by a specific device. The BACnet standard implies that all BACSs shall have a PICS identifying "all of the portions of BACnet that are implemented" [1]. Information in PICS includes: a brief description of the de-

[1] In the remainder of this paper, we will refer to any controller implementing BACnet as BACS.

vice; a list of supported BACnet Interoperability Building Blocks (BIBBs) that define classes of BACnet services supported by a device; and a list of supported standard and proprietary BACnet objects and properties with their characteristics.

To complete device descriptions, operators may take advantage of configuration files such as "Engineering Data Exchange" (EDE) files [2]. Generally, operators compile these documents to represent internal characteristics of a deployed BACS. EDE files include detail information on devices' BACnet implementations (e.g., which BACnet objects a device is currently using) and value constraints (e.g., in Figure 2, Present_Value of Analog Value 7 must be less than 5.0).

3.2 Attacks on BACnet

BACnet defines a limited security architecture providing peer and operator authentication along with data confidentiality and integrity ("Clause 24 — Network Security" [1]). However, none of this is implemented in available products [43]. This leaves BACnet infrastructures vulnerable to numerous cyber-threats [24, 63, 58, 29].

We categorize attacks on BACnet into three main groups: snooping, denial of service (DoS), and process control subverting. This categorization derives from the list of BACnet protocol threats described in [24].

Snooping attacks concern stealing information about a specific building automation system. To achieve this goal, these attacks require access to the building automation system network. Once inside, attackers can take advantage of BACnet services such as `ReadProperty` and `ReadPropertyMultiple` to gain knowledge of the BACS. This includes device models, locations, status, and information on their BACnet support (e.g., which BACnet services and objects they implement). Attackers may need this information to understand the infrastructure and pave the way to further intrusions. However, snooping attacks do not disrupt any process of the building automation system.

Differently, DoS attacks try to interfere with control processes by making controllers unreachable for operators. This category only considers DoS attacks that are performed through the use of BACnet routing features (e.g., malicious modifications to the BACnet routing tables) and leaves other kinds of DoS out of its scope. As for the snooping attacks, DoS attacks need malicious users to have access to the network. Moreover, this kind of attacks requires information about the network layout. Attackers can achieve their goal by sending BACnet messages, such as `Initialize-Routing-Table`, to modify a BACS' routing tables. In this way, operators lose visibility on single devices or even entire sections of the building automation system.

Finally, process control subverting includes those attacks that directly modify control processes and, consequently, interfere with physical operation. This kind of attacks requires more skilled attackers with sufficient knowledge about the building automation system functioning. In this scenario, attackers exploit specific controllers by using several different BACnet services, such as `WriteProperty` or `DeleteObject`, to change the BACSs' structures and operations. This leads to a loss of control by the operators and, consequently, leads to risks for components and people.

3.3 Evaluation Environments

For this work, we analyzed two different building automation installations over more than two months of constant operation. The first building automation system belongs to the University of Twente in the Netherlands and is in charge of supervising utilities and services provided to the university campus. Its duties encompass energy consumption control, HVAC, and room monitoring and management (e.g., pressure and temperature control, shading, etc.). The second building automation system belongs to the Lawrence Berkeley National Laboratory (LBNL) and supervises several services on its premises. The LBNL process control focuses mostly on room monitoring and energy consumption for the Lab facilities. Both infrastructures deal with hundreds of BACSs from several different vendors.

The IDS we deployed at the University of Twente linked to a SPAN port on a switch directly connected with the SCADA servers monitoring the whole building automation system. The same switch is responsible for routing most of the traffic of the building automation network. This allowed us to capture and analyze most of the BACnet messages exchanged by BACSs. Differently, at LBNL we could monitor only a subset of the building automation system by linking to a switch in charge of connecting BACSs inside one building. However, this was sufficient to automatically gather the information needed for our approach to craft the specification rules.

The two infrastructures generally showed similar traffic patterns. Several BACSs shared the same sets of objects and used the same kind of messages to exchange information. Furthermore, both UT's and LBNL's traffic samples included numerous BACnet routing messages (e.g., `Who-Is`, `I-Am`, `Who-Has`, and `I-Have`) organizing communication paths within the two networks. However, the two infrastructures presented some differences related to communication and control strategies (e.g., all BACSs deployed at UT used confirmed services thus requiring acknowledgments from message recipients while some devices at LBNL used just unconfirmed ones). Particularly, the employment of

BACSs from different vendors led operators to employ individual procedures implemented through the use of `ConfirmedPrivateTransfer` BACnet services. Such services are used to invoke proprietary or non-standard routines in remote BACSs.

3.4 Setting and Threat Model

The reason to develop a network-based IDS is twofold. First, a network-based solution is easier to deploy than host-based ones. Secondly, this setup allows us to have minimum impact on NCS processes. Once deployed, we assume that our system is able to capture real-time traffic of the monitored building automation system in a completely passive fashion and to retrieve documentation publicly available on the Internet. This allows to gather the information we need to build specification rules and implement effective detection.

On the other side, we assume attackers can gain full access to the network as well. We consider this happening in a way that is similar to standard IT environments (e.g., phishing, software vulnerability exploitation). Tools such as Shodan [40] show how easy it is to find building automation networks exposing their devices to the Internet. Once inside, attackers can obtain a convenient viewpoint on the building automation control processes. Two key factors support this assumption. First, most building automation protocols take advantage of broadcast communications to exchange information among devices (e.g., routing notifications). This already allows attackers to easily observe a large part of the traffic. Secondly, the hierarchical structure of common building automation networks steers valuable information messages towards servers and DCSs. By gaining access to one of these servers, attackers can observe most of the traffic within the building automation system.

Within a building automation network, attackers may use attacks outlined in §3.2 to gain knowledge on, or subvert, the correct functioning of the building automation system. In this last scenario, any safety feedback in place can usually be overridden [49]. Therefore, attackers can put infrastructure components under stress, possibly threatening human safety when it comes to devices such as electrical equipment.

4 Specification Mining Approach

Our approach works towards automated development of specification rules for network security monitoring. Setting up and customizing a specification-based IDS for a particular infrastructure requires a large amount of information about the monitored system, implying a substantial manual effort in gathering and refining the specification rules. As details of the infrastructures are often described within specs and configuration files, especially in many NCS environments, the process of collecting this information—and, consequently, the development of the actual IDS—can be automated to a fairly high degree through the following steps (see also Figure 3):

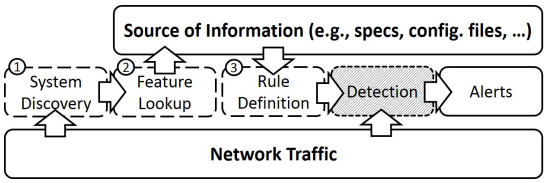


Figure 3: Specification-mining approach

(1) System Discovery gathers information about the monitored NCS. In this step, our system analyzes the network traffic in order to: 1) identify devices communicating on the network (e.g., models, brands); and 2) determine role and purpose of each identified device (e.g., a device is a controller, an HMI, etc.). Every time the system collects enough information about a specific device it proceeds with the next step.

(2) Feature Lookup implements a set of information retrieval techniques to gather knowledge about devices identified during System Discovery. The purpose of this step is to: 1) find verified information (e.g., specs, configuration files) about the infrastructure's devices; and 2) select features and constraints from the retrieved documents and arrange results in a structured form.

A successful Feature Lookup relies on the assumptions of *availability* and *linkability* outlined in the introduction. The assumption of *availability* implies the existince of documents about infrastructures and components that are automatically retrievable. This requires the information to be provided in electronic form and being suitable for parsing. Also, this assumption includes an assurance on the authenticity of the retrieved information (e.g., by the use of reliable sources, by the employment of secure retrieval techniques). The assumption of *linkability* guarantees that the information derived by the retrieved documentation can be checked by the system against observations within the traffic (e.g., messages, variables, etc.). Particularly, after the identification of network devices and the successful retrieval of their related constraints from the documentation, the assumption of *linkability* enables assigning effective specification rules to the right targets.

(3) Rule Definition uses the knowledge obtained in the Feature Lookup to craft the specification rules. To achieve this goal, the system needs to: 1) select identified information from Feature Lookup; 2) translate this information to specification rules.

We focus our specification-based intrusion detection on *controllers* (e.g., BACSs for building automation systems, Programmable Logic Controllers or PLCs for ICSs, Electronic Control Units or ECUs for in-vehicular networks). This decision comes from the key role these components have within NCSs: Controllers are involved in any monitoring and control operation of the infrastructure either autonomously or accessed by operators. Furthermore, controllers are likely targets for attackers (as illustrated in §3.2).

We observe that NCS controllers share a number of properties. First, every controller *employs a limited set of variables to fulfill its function*. These variables can go from simple memory addresses to complex objects but *often have predetermined types*. Moreover, all controllers *use a limited set of methods (or services) to access and manipulate variables of other controllers*. Finally, *each variable can assume a limited range of values according to its type or the physical characteristic it represents*. We leverage these shared properties to define a set of general constraints, or *abstract rules*, checking NCS variables' *types*, *values* and access *methods*. These abstract rules are the seeds we use to automatically generate specialized specification rules. To achieve this, we define a mechanism that maps information retrieved in the Feature Lookup step to the abstract rules. This process automatically completes the abstract rules and, as a result, customizes detection for the monitored NCS.

Once a rule is defined, it becomes active and, thus, part of the detection mechanism. During detection, an active rule verifies if its related constraint is fulfilled or not. When this last condition becomes true, the system triggers an alert for the user.

Having presented the phases in a generic way, we now describe our experimental setup and, then, how we have instantiated them to build a specification-based IDS for BACnet-based building automation systems.

Implementation background We implement our approach using the Python programming language [51] and Bro [46]. Bro is a network traffic analyzer employed in different domains such as network security monitoring and performance measurement. The system comes with comprehensive built-in functionalitiy for traffic analysis and supports several network protocols ranging from standard (e.g., HTTP, FTP) to domain-specific (e.g., Modbus [41], DNP3 [10]). Bro provides a Turing-complete scripting language that allows users to select and analyze network events (e.g., connection establishments). We choose to describe specification rules through the "Bro scripting language" because of its efficiency and expressiveness. We developed a BACnet parser for Bro using Spicy [55], a parser generator whose specification language allows users to define a protocol's syntax and semantics at a high level. We publish the BACnet parsing code for Spicy, as well as the Python scripts , as open source software.[2] However, we cannot open-source the Bro code containing the rule checks due to privacy agreements with the two building automation system sites.

5 System Discovery

To identify BACSs we implement three different techniques that we term: *"BACnet Device Object analysis"*, *"BACnet Address linking"*, *"BACnet Property set fingerprinting"*. The first technique directly follows from the protocol standard and relies on the mandatory presence of a `Device` object in every BACS device. The `Device` object defines "a standardized object whose properties represent the externally visible characteristics of a BACnet device". Among these properties there are: `Object_Name`, `Vendor_Name`, `Vendor_Identifier`, `Model_Name`, `Firmware_Revision`, `Application_Software_Version`, `Location`, and `Description`. Most of these properties are set by vendors and provide information on a device's identity (e.g., `Model_Name`) and role (e.g., `Description`). BACnet services such as `ReadProperty` and `ReadPropertyMultiple` can access those properties. As these services are widely employed by user interfaces and logging servers to automatically update data related to infrastructure's components, information on Device objects regularly passes through the network and, thus, is available to System Discovery. As the `Object_Identifier` property of a `Device` object is a parameter that uniquely identifies a device in a BACnet network, a message such as the one in Figure 4 allows us to identify a BACS and understand its purpose. In the Wireshark screenshot example, BACS with identifier "17001" is a "Blue ID S10 Controller".

For BACnet objects of other types, since no information can be extracted from the IP address (multiple BACnet devices may share the same IP address), a further parameter allows to identify message sources and destinations: the BACnet address. As for the `Device` object's `Object_Identifier`, the BACnet address (together with the `Network Identifier`) is unique within a BACnet network. In the "BACnet Address linking" technique, the BACnet address bridges the gap between a known `Device` object and any BACnet object included in the same BACS. Figure 5 shows an example of this analysis. When "device 4001" is known (as a result of the previous technique), any message carrying both the related `Device` object's `Object_Identifier` and the

[2] `https://github.com/specification-mining-paper-usenix-2016/specification-mining`

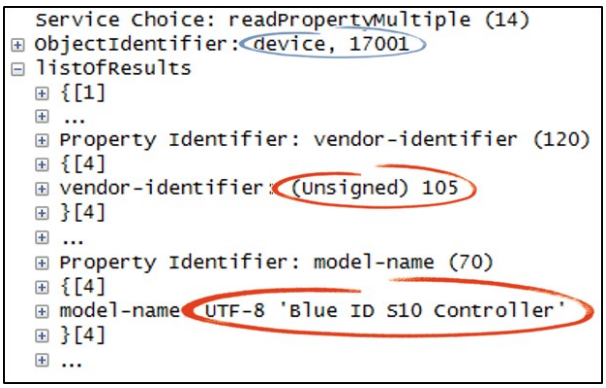
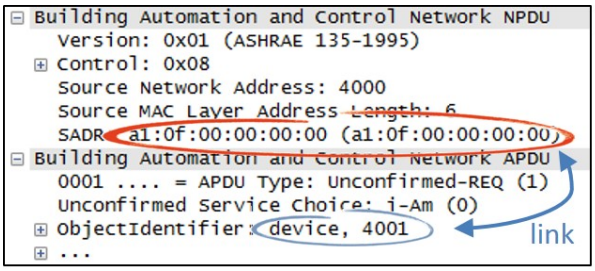
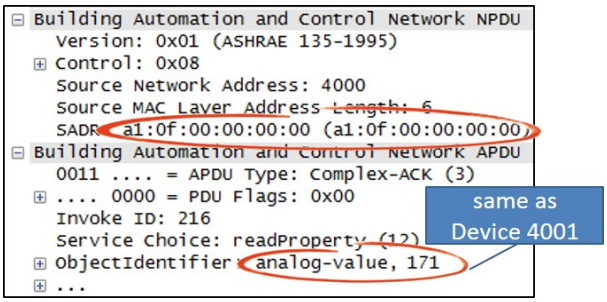


Figure 4: "BACnet Device Object analysis" example



(a) `I-Am` message



(b) `ReadProperty` message

Figure 5: "BACnet Address linking" example

BACnet Address allows us to link the two parameters (Figure 5a). Any later message then carrying the BACnet address along with a further object (e.g., "Analog Value 171") enables linking to the corresponding device (Figure 5b). This technique works well because `I-Am` messages pass the network frequently to ensure visibility of all BACnet objects.[3]

Finally, if no information can be extracted from `Device` objects or BACnet addresses, System Discovery can benefit from observations of the BACnet properties. As discussed in §3.1, the BACnet property set is extensible. Every object of a BACS has a set of standard and proprietary properties that form a "fingerprint" of that object and device. The third technique assumes that two objects sharing the same fingerprint are likely to be of the same kind. During System Discovery, it is possible to create a database of identified fingerprints each one pointing to the corresponding BACS (identified with the previous two techniques). Whenever an unknown object presents a property set already in the database, the system infers the most likely related device.

[3]The technique can also directly use messages carrying `Device` object information if the source BACnet address is present in the header. However, for this kind of messages, having the BACnet Address fields is not mandatory.

Table 1: University of Twente - BACS device list

# of devices	Vendor	Model	Role
5	Kieback&Peter	DDC4000	DCS
15	Priva	HX 80E	Router
7	Priva	Compri HX	Controller
25	Priva	Compri HX 3	Controller
36	Priva	Compri HX 4	Controller
12	Priva	Compri HX 6E	Controller
85	Priva	Compri HX 8E	Controller
2	Priva	Blue ID S10	Controller
16	Priva	Comforte CX	HMI
2	Delta Controls	eBCON	Controller
3	Siemens	PXG80-N	Controller
3	Siemens	PXC64-U	Controller
3	Siemens	PXC128-U	Controller
3	Siemens	PXR11	Controller
3	Siemens	PXC00-U + PXA30-RS	Controller
1	Unknown	Unknown	-

Table 2: LBNL - BACS device list

# of devices	Vendor	Model	Role
23	Automated Logic	LGR	Router/Gateway
14	Automated Logic	ME	Controller
11	Automated Logic	SE	Controller
159	Automated Logic	ZN	Controller
1	Automated Logic	WebCTRL	HMI
9	Johnson	NAE	Controller
1	Johnson	NIE	Controller
4	Paragon Controls Inc.	EQ	Controller
4	Sierra	BTU Meter	Energy meter
4	Sierra	FFP	Controller
1	Tracer	UC400	Controller
2	Niagara	AX Station	SCADA server
7	Unknown	Unknown	-

Experiments Previous work by us shows that traditional fingerprinting techniques are usually ineffective on most NCSs [5]. In our tests, tools such as Nmap [38] and P0f [64] were able to identify just a limited number of Windows and Linux workstations. The techniques presented above proved more effective. Thanks to frequent `ReadPropertyMultiple`, our system was able to gather information on most BACSs. Moreover, BACnet address linking and BACnet property set fingerprinting allowed the system to link most of the observed BACnet objects to identified devices. At the end of System Discovery, we gathered information on ∼15k BACnet objects belonging to the 445 devices shown in Tables 1 and 2.

Thanks to the information from the operators, we

know that we correctly identified 98.2% of the BACSs actually deployed (445 out of 453 devices). Eight devices did not link to any useful BACnet message or identifiable property set. However, these devices convey almost no information over the network (a few hundreds BACnet messages over two months of capturing compared to an average of tens of thousands) and did not involve any notable equipment. Identifying the aforementioned 445 devices took just a few hours of monitoring.

6 Feature Lookup

Searching for documentation on identified BACSs is possible because the two assumptions of *availability* and *linkability* hold for BACnet-based building automation systems. Verified information about BACSs is *available* within PICSs and EDE files. This information includes BACSs' vendors, models and even refers to specific BACnet objects, thus is *linkable* to what we observed over System Discovery.

Feature Lookup targets both online and offline documentation. On the one hand, we use Google APIs to search and retrieve publicly available documents such as PICSs on the Internet. On the other hand, we retrieve EDE files from private repositories in the installations. Both cases allow for document authenticity. In the former case, we narrow the search to a subset of reliable sources such as vendors' websites and reputable third parties (e.g., BACnet International Laboratories[4]). In the latter case, we assume a secure connection to a trusted dataset managed by the operators.

Once a BACS links to one or more of these documents, our system parses the documents looking for useful information. According to BACnet specifications, a PICS has a standard template and we observe that most PICSs are closely modeled to it. Figure 6 shows three extracts from the PICS of the "Blue ID S10 Controller" mentioned in the previous section.

As outlined in §3.1, each PICS provides a description of the related BACS and the BIBBs it implements (Figure 6a). Moreover, PICSs include information about supported BACnet objects and properties, as well as their characteristics (Figures 6b and 6c).

EDE files also follow a standard template but they use a simpler "comma-separated values" (CSV) format. Each EDE file presents details of a specific BACS (Figure 7 shows an extract of Device 4001 EDE file). Data includes all implemented BACnet objects (e.g., "device 4001" owns "Analog Value 171", "Multi-state Value 15", etc.) and their descriptions. Furthermore, EDE files include information about `Present_Value` properties with

[4] `http://www.bacnetinternational.org/`

Product information	
Date	2013-07-23
Product name and model number	Blue ID S10 Controller
Application software version	1.0
Firmware revision	1.00
BACnet protocol revision	9
Product description	The Blue ID S10 Controller contains a powerful microprocessor. It can be easily programmed for building automation purposes. The processing speed and computing power mesh seamlessly with the requirements of modern and integrated systems. The controller uses a reliable operating system that ensures quality and operational security. It is fully controllable via BACnet including commissioning, writing, reading, alarm and event handling.
BACnet standardized device profile	BACnet Building Controller (B-BC)

Vendor information	
Vendor name	Priva
Vendor ID	105
Contact information	PO Box 18 2678 ZG The Netherlands www.priva.co.uk

Supported BIBBs	
DS-RP-A	Data Sharing - Read Property - A
DS-RP-B	Data Sharing - Read Property - B
DS-RPM-A	Data Sharing - Read Property Multiple - A

(a) PICS excerpt 1

Standard object types supported[1]	Dynamically creatable and deletable
Accumulator	no
Analog Input	no
Analog Value	no
Binary Input	no
Binary Output	no
Binary Value	no
Calendar	yes
Device	no

(b) PICS excerpt 2

Supported properties per object type	Required or optional	Readable (R) or readable/writeable (R/W)	Additional comments
Accumulator			
Object_Identifier	required	R	
Object_Name	required	R	
Object_Type	required	R	
Present_Value	required	R/W	
Description	optional	R	

(c) PICS excerpt 3

Figure 6: PICS example

PROJECT_NAME	UTWENTE4\Hoofdschema							
VERSION_OF_REFERENCEFILE	126							
TIMESTAMP_OF_LAST_CHANGE	14-04-2015, 11:11:59.890							
AUTHOR_OF_LAST_CHANGE	Regel Partners							
VERSION_OF_LAYOUT	2							
device obj.-instance	object-name	object-type	object-instance	present-value-default	min-present-value	max-present-value	unit-code	description
4001	AV - 171	2	171	0	0	100	98	setp. KM1\Y1 - stuurm
4001	MV - 15	19	15	1				Selectie nieuw\Selec
4001	AV - 11	2	11	100	0	32767	71	Selectie nieuw\Bedri
4001	AV - 42	2	42	0	0	2	95	Selectie nieuw\Te g
4001	AV - 43	2	43	0	0	2	95	Selectie nieuw\Ze g

Figure 7: EDE file example

value ranges (e.g., "Analog Value 171" can vary from `min-present-value` 0 to `max-present-value` 100).

Experiments The program we implemented to search for online documentation uses the outputs of System Discovery (vendors and models) and further keywords such as "PICS" to retrieve information about identified BACSs. The system ranks Google results coming from public repositories (e.g., www.bacnetinternational.net) and the web by quantifying the presence of the keywords in document titles. For example, the "Blue ID S10 Controller" links to a PDF document titled "BACnet_PICS_Blue_ID_S10_Controller.pdf" (Figure 6). With this technique we identified a PICS for 99.3% of the de-

vices deployed in the two building automation systems (442 out of 445 among the devices identified in the System Discovery step). Two 'Siemens PXR11' and one 'Paragon Controls Inc. EQ' were the only devices that did not link to any PICS. However, we could not find the related PICSs even by a manual search either.

Offline research targeted specific devices directly. While online documentation always provides general information about BACSs of a certain kind (e.g., all "Blue ID S10 Controller"), offline repositories provide detailed information related only to devices deployed in the monitored building automation system. For this reason, instead of vendors and models we searched through the available documents using device `Object_Instances`. For example, starting from "Device object 4001" from System Discovery, we found an EDE file titled "Controller 4001_EDE.csv". While LBNL did not provide any configuration file, operators from the University of Twente shared with us 10 files of this kind. While they confirmed that there was indeed an EDE file for every deployed device, they could not grant us unlimited access to all of them due to information sensitivity. For this reason, the operators chose the 10 files based on roles and purposes of the related devices. Each file we obtained described a BACS identified over System Discovery.

The aforementioned privacy concerns refer to the initial manual analysis we had to perform over the EDE files and would not hamper the applicability of our approach. In an ideal deployment, one would have a secure connection between the IDS and the machine storing the EDE files, without any human activity involved for retrieval operations and processing. However, both University of Twente and LBNL operators store infrastructure documentation on computers also used for other purposes than building automation, and direct connections to those resources were infeasible.

Finally, we implemented two programs to parse PICSs and EDE files respectively. In the first case, the program goes from document's top to bottom guided by the diagram shown in Figure 8. For every available PICS, the program first selects all implemented BIBBs and BACnet objects (Figures 6a and 6b). Each object can be creatable/deletable and this information follows the object as a "yes/no" or equivalent symbols (Figure 6b). Finally, for every object, the script selects a list of properties that can be writable or not (Figure 6c). Figure 9 shows parsing results of the "Blue ID S10 Controller" coming from the PICS showed in Figure 6.

Most of the retrieved PICS did not have any information about property values. Instead, this information was included in the EDE files. A further program went through all EDE files selecting `Present_Value` minimum and maximum values for every listed object. This new information was structured as shown in Figure 10.

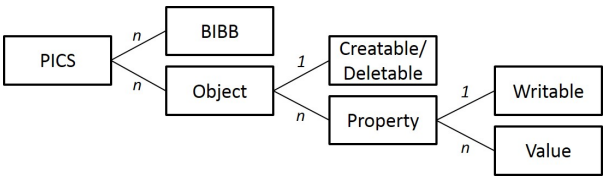


Figure 8: PICS parsing diagram

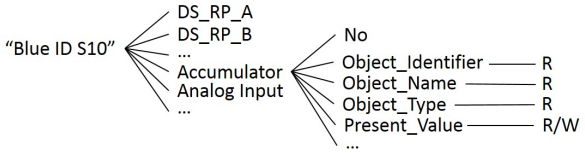


Figure 9: Parsing PICS example results

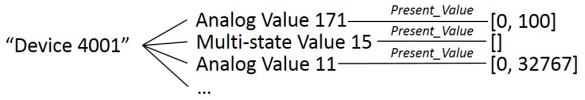


Figure 10: Parsing EDE example results

7 Rule Definition and Detection

Next we describe how the information gathered in previous steps is used to define specification rules. In §4, we motivated our focus on variables' *types*, *values* and related access *methods* as basis for our specification rules. From this, we derive three *abstract* rules: 1) a "Type" rule checks if a variable of a specific type is allowed; 2) a "Value" rule checks which values a variable may assume; and 3) a "Method" rule checks which methods can be used to access a specific variable. All rules have the same structure: each element (type, value, method) is evaluated against a set of allowed possibilities. For example, in the "Type" rule, a variable's type is evaluated against all the allowed types of variable a controller may implement (Algorithm 1).

We use a Python program to automate the process of mapping information retrieved over Feature Lookup to the abstract rules. In the following, we discuss how we map these abstract rules into specification rules for monitoring for each type.

Type Rule: The "Type" rule checks which BACnet objects and properties each BACS can use. This information comes from the PICSs (Figures 6b and 6c) and, thus, is included in the results of Feature Lookup (Figure 9). Therefore, a script selects allowed objects and properties of each identified BACS and transforms the "Type" rule into the two specification rules shown in Algorithm 2.

In the case of the "Blue ID S10 Controller", the

Algorithm 1 Abstract "Type" rule

1: **if** $\text{Variable}_{\text{type}} \notin \text{Controller}_{\text{AllowedVariableTypes}}$ **then**
2: *Alert*("Variable type not permitted")
3: **end if**

Algorithm 2 BACnet "Type" rules

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

1: **if** BACnet Property $\notin \text{Controller}_{\text{Object}_{\text{AllowedPropertyTypes}}}$ **then**
2: *Alert*("Forbidden Property")
3: **end if**

$\text{Controller}_{\text{AllowedObjectTypes}}$ set contains objects `Accumulator`, `Analog Input`, etc. In the same way, the $\text{Controller}_{\textbf{Accumulator}_{\text{AllowedPropertyTypes}}}$ set of a "Blue ID S10 Controller" contains properties `Object_Identifier`, `Object_Name`, etc. Whenever the system captures a BACnet message including an object and some property, the two rules check object and property types respectively and alert if these types are not included in the defined sets. This allows the system to detect snooping attacks and any other attack dealing with unexpected objects and properties.

Value Rule: The "Value" rule checks which values BACnet properties may assume. This information comes from the EDE files (Figures 7 and 10) and, thus, is automatically mapped to the concrete rule as shown in Algorithm 3.

Algorithm 3 BACnet "Value" rule

1: **if** BACnet Property value $\notin \text{Controller}_{\text{(Object,Property)}_{\text{AllowedPropertyValues}}}$ **then**
2: *Alert*("Forbidden Value")
3: **end if**

For example, when it comes to "Device 4001", the system alerts if "Analog Value 171" is below 0 or above 100. This rule protects the infrastructure against process control subverting scenarios, and thus attacks attempting to modify parameters of the physical and control processes.

Method Rule: The "Method" rule validates the BACnet services each BACS can use. This information comes from the PICSs in the form of a list of BIBBs (Figures 6a) and is included in the results of Feature Lookup (Figure 9). BIBBs can be replaced with corresponding services by a simple lookup operation. Therefore, Algorithm 4 checks if a BACnet service belongs to the set of allowed services.

In the case of the "Blue ID S10 Controller", the $\text{Controller}_{\text{AllowedServices}}$ includes services from BIBBs DS_RP_A (`ReadProperty_Request`), DS_RP_B (`ReadProperty_Response`), etc. This rule allows the system to detect attackers misusing BACnet services to fulfill their goals.

Algorithm 4 BACnet "Method" rule

1: **if** BACnet Service $\notin \text{Controller}_{\text{AllowedServices}}$ **then**
2: *Alert*("Forbidden Service")
3: **end if**

Furthermore, we use the "Method" rule to check which BACnet object is creatable/deletable and which BACnet property is writable. Following the standard, we compile three sets of BACnet services with services that create objects, that delete objects, and that write properties respectively. Then, the system uses the information from Feature Lookup to define checks on non-creatable/deletable objects and non-writable properties by using Algorithm 5.

Algorithm 5 BACnet additional "Method" rules

1: **if** BACnet Service $\in$ CreateObjectServices **then**
2: *Alert*("Forbidden object creation")
3: **end if**

1: **if** BACnet Service $\in$ DeleteObjectServices **then**
2: *Alert*("Forbidden object deletion")
3: **end if**

1: **if** BACnet Service $\in$ WritePropertyServices **then**
2: *Alert*("Forbidden property writing")
3: **end if**

For example, the first two rules alert the presence of services attempting to create or delete `Accumulator` objects belonging to a "Blue ID S10 Controller". The third rule reports any service attempting to write to a non-writable property, such as `Accumulator`'s `Object_Identifier`.

Experiments Our system filled the abstract rules with the information coming from Feature Lookup crafting hundreds of specification rules. To improve efficiency we arrange the specification rules in an order that avoids meaningless checks (e.g., we do not want to check a BACnet property if we already know that the BACnet object it belongs to is not allowed). For every captured BACnet message, the system checks if the BACnet service is allowed; then, if involved BACnet objects can be used, created or deleted; then, if involved BACnet properties are allowed and writable. Finally, the system examines properties' actual values. Only a small set of specification rules are of this last type due to the limited number of EDE files that operators provided us with.

As outlined in §3.3, we tested our approach against more than two months of real traffic. Over the two months of capturing, our system triggered 237 unique alerts; 226 at the University of Twente and 11 at LBNL.

Table 3: Detection results

Abstract Rule	Specification Rule	# Alerts
Type Rule	Forbidden object	2
	Forbidden property	234
Value Rule	Forbidden value	0
Method Rule	Forbidden service	0
	Forbidden object creation	0
	Forbidden object deletion	0
	Forbidden property writing	1

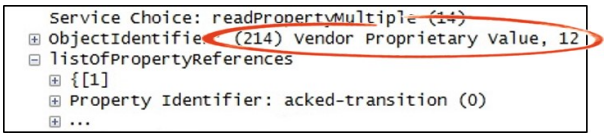
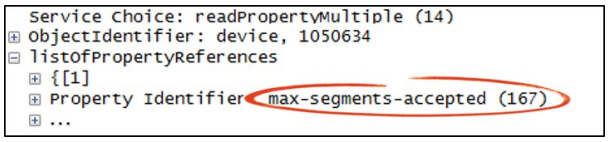
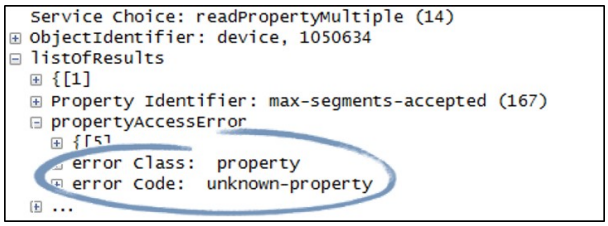


Figure 11: Unexpected object ReadProperty_Request



(a) Unexpected property ReadProperty_Request



(b) ReadProperty_Response confirmation of the alert

Figure 12: Unexpected property read operation

The two results differ because of the different views we achieved over the two infrastructures (as already described, at LBNL we could monitor only a subset of the building automation system and thus a subset of the traffic). Table 3 shows the three abstract rules, the corresponding specification rules and whether or not a rule raised an alert.

We did not find any evidence of malicious activities over the time span of the captures. However, our approach still provided interesting insights. At the University of Twente, the system raised alerts on two BACSs using forbidden objects. Both cases involved a "proprietary" object never described within the PICS. According to the available documentation, the two devices (two Siemens controllers PXC128-U) should not include anything that was not defined within the BACnet standard. Nevertheless, a device probed the two controllers (Figure 11) and received back correct BACnet responses about an unknown object. A meeting with the operators revealed that this object is vendor-defined and gathers information on parameters of the BACSs recognizable and understandable by vendors only. Operators confirmed that vendors have access to the building automation system to monitor their devices, and use of an unknown BACnet object happens even though the documentation does not mention this possibility because of its internal nature. However, operators did not know that involved BACSs provide specific functionalities and attackers can potentially exploit such circumstances.

Detection on BACnet properties provided the highest number of alerts (all alerts at LBNL were of this kind). Our system generated several alerts on `ReadProperty` and `ReadPropertyMultiple` messages attempting to retrieve non-existing properties. As a matter of fact, these properties were not defined by the PICSs and, for most cases, we could eventually confirm the non-existence of these properties by observing some BACnet errors carried in the responses to those read requests (Figure 12).

A BACS asking for unimplemented properties is not necessarily a violation of the specs. In fact, all PICSs define what a BACS implements without defining what other BACSs may ask for. A situation in which a BACS sends back a BACnet-error response to warn about a non-existing property (Figure 12b) is in line with the specs and should be of no harm for the system. However, the reason to alert on situations of this kind is twofold. First, this situation may be of interest from a security perspective. Despite being handled by the BACnet protocol, these circumstances may hide a "network discovery" scenario where an attacker tries to gain knowledge of the infrastructure by randomly probing BACSs. As described in §3.2, snooping is one plausible attack in building automation systems. Secondly, the same situation shows a common side-effect of the joint use of different BACnet software solutions. As servers and workstations do not usually know in advance which BACSs they will connect to, predefined BACnet discovery messages exist in order to gather general information of building automation components. These messages do not consider which BACnet properties are defined for each device and simply use large sets of them. This consequently generates several error responses on the network.

To dig deeper into property-related issues, we extended the "unknown property" specification rule to also check if properties enforced by PICSs were always implemented. Therefore, we created a further instance of "Type" rule checking all BACnet error messages to detect missing properties that were supposed to be used by the BACSs. The system revealed several messages reporting "unknown-property" errors about properties declared to be part of devices' BACnet implementations.

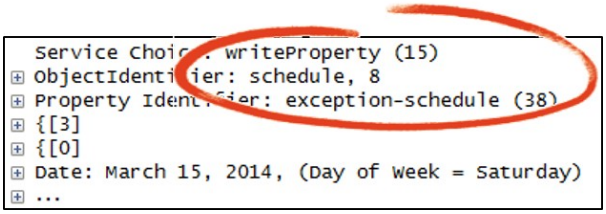


Figure 13: Unexpected property write request

All these mismatches between implementation and specification are particularly relevant for what concerns interoperability. In fact, software solutions that define their interactions with a BACS based on its public documentation can incur into inconsistencies caused by incorrect or lacking implementations.

Finally, the system triggered an alert corresponding to an unexpected write operation on a BACnet property supposed to be readable only. A Priva controller received a BACnet `WriteProperty` request on the `Exception_schedule` of an object `Schedule` (Figure 13). Despite what we knew from the related PICS, the BACS sent back a SimpleACK message, acknowledging the success of the operation (the actual writing was confirmed by later read operations). These kinds of situations are especially dangerous due to the unpredictability of their results. As no indication is provided by the vendor, the write operation can either succeed or fail, and may generate a response or not (even independently from the actual modification of the value within the property). Meeting with the operators revealed that this write operation was due to a human mistake during the configuration of the Priva controller. However, the same situation could fit the "process control subverting" scenario described in §3.2.

8 Discussion

Performed experiments confirm the feasibility of the approach within building automation systems and pave the way for its application to different domains.

8.1 Analysis of the Results

By construction, our IDS is able to detect events that do not match the specifications coming from retrieved documentation. This aspect leads to two considerations:

- On the one hand, an alert raised by the system does not necessarily refer to a security-relevant event as the related mismatches may not directly harm the monitored devices. However, all findings revealed network activities otherwise invisible to operators. Over the two months of analysis, every alert identified either an actual mismatch between device documentation and implementation (e.g., unimplemented BACnet Properties) or an operator mistake (e.g., the unexpected writing operation). As already discussed, these issues can cause significant gaps in the knowledge operators have about their infrastructures and may potentially lead to dangerous misconfigurations of the involved systems. The meeting with the operators at the University of Twente confirmed that employed HMIs were not able to signal any of the misconfigurations found or even notify the users on generated BACnet errors. As a result, the University of Twente asked to deploy our system into the building automation system continuously and let operators receive notifications of the generated alerts. So while our datasets did not include actual attacks, we were able to reliably detect notable deviations from the specifications at zero false-positives. This result is in line with the work of Uppuluri et al. [62] showing that specification-based intrusion detection works towards optimal detection rate while substantially decreasing the number of false positives compared to anomaly-based detection.

- On the other hand, our approach does not necessarily detect all possible attacks threatening the monitored infrastructure. In fact, any attack operating within the boundaries defined by employed specifications would not be caught by our IDS. However, our solution substantially narrows down what a malicious user can do and covers most of the attack scenarios defined within the categories listed in §3.2. Furthermore, our solution does not exclude the use of other approaches such as pure anomaly-based intrusion detection either improving the obtained rule set or working in parallel.

Each one of the implemented phases effectively achieved the defined goals. Thanks to the numerous read operations, System Discovery took just a few hours of network sniffing to gather all the information needed to describe the whole set of BACSs. With this information, our approach was able to rapidly and automatically identify available sources of information and craft effective specification rules.

Feature Lookup focused just on structured documents such as PICSs and EDE files. In some of the tests, we further extended online research to documents such as BACS user manuals. Our system was able to download 10 manuals related to components deployed in the monitored infrastructures. However, we decided to not further employ manuals because an analysis showed they were fullly overlappig with the information found in the PICSs. Nevertheless, one way to improve our

specification-mining approach is to enable handling heterogeneous documentation and, especially, unstructured information. To this regard, we observe that Feature Lookup should abstract from domain-specific parsing scripts and generalize the process of mining and structuring infrastructure features. Correctly selecting information can take advantage of standard data mining and natural language processing. Our work did not present a general approach to this activity. However, works such as [57, 15, 50] may fulfill this goal. With more general techniques capable of extracting knowledge from heterogeneous documentation, the effort of deploying the system completely converges on mapping retrieved information to the abstract rules. According to the monitored infrastructures, operators should identify the related concepts of variable type, value and access method and, eventually, let the system interpret data coming from Feature Lookup and instantiate the specification rules.

Even without such a general approach, our solution drastically reduced the time needed to deploy intrusion detection into a BACnet-based building automation system. Obtaining the same set of specification rules by hand would have required substantial effort, making it infeasible for larger infrastructures. Furthermore, the obtained system comes with the intrinsic capability to update according to the changes of the monitored infrastructure. In fact, whenever new BACSs are deployed, our system transparently reads the new information over the network and goes through the three steps all over again. In the end, this solution makes the implemented system directly applicable to any other BACnet infrastructure with no further effort on configuration or deployment.

The proposed approach works likewise for different building automation technologies. As discussed, this would mostly require a modification of the mapping process linking retrieved information and abstract rules but would leave the core concept unchanged. Other building automation infrastructures such as KNX [32] and LonWorks [11] also meet the requirements of *availability* and *linkability*. These widely used protocols present characteristics similar to the ones observed for BACnet. Moreover, both KNX and LonWorks promote and support the use of documents describing protocol implementation details (although not as formal as BACnet PICS).

To show the generality of our approach beyond building automation systems we outline how the same specification-mining technique applies to two different domains of NCSs, namely ICS and in-vehicle networks.

8.2 Industrial Control Systems

ICS is a term generally used to indicate several types of control systems (e.g., Supervisory Control And Data Acquisition or "SCADA") used in industrial production for monitoring and controlling physical processes. ICSs work over several domains such as energy, water treatment, manufacturing, etc. and embrace a wide family of technologies. Among them, Modbus [41], MMS [25], IEC104 [12], and DNP3 [10] are some of the most used protocols and standards deployed for industrial control.

Specification-based intrusion detection for ICSs is not new. Works such as [4] show the effectiveness of this approach applied to electrical grids. However, applying a set of specification rules to a real deployment still requires manually crafting all parameters on specific needs. Again, our research can improve the use of specification-based intrusion detection by leveraging available information of the deployments. For example, in the smart grid scenario, we would focus on Programmable Logic Controllers (PLCs) and Remote Terminal Units (RTUs), as these play a main role in the infrastructure. We would analyze variables handled by these controllers, their types, values and access methods and then use the abstract rules defined in §7.

Regarding the assumption of §6, verified information about the smart gird is *available* within configuration files that use the "Substation Configuration Language" (SCL). SCL files usually provide formal representations of modeled data and communication services. The information included in these files is *linkable* thanks to the included detail descriptions of the involved infrastructures (e.g., "Substation Configuration Description" files). Besides SCL files, operators usually store additional documentation describing physical and control processes as in the building automation use case. An IDS can leverage this documentation to gather further information and derive specification rules.

The three steps of the approach remain unchanged. System discovery will passively gather data about devices communicating over the ICS network. According to the verbosity of the involved protocols, an IDS will eventually collect enough information to identify infrastructure components and start the Feature Lookup step. Once information about PLCs and RTUs functioning is retrieved, Rule Definition will use it to define the actual specification rules.

8.3 In-Vehicle Networks

Similar argumentation can be applied to communication of Electronic Control Units (ECUs) over automotive bus systems like the "Controller Area Network" (CAN) or FlexRay found in all of today's cars.

CAN is a network where connected ECUs communicate by means of small messages with a payload of only 8 bytes. CAN uses content-based addressing where messages only carry a 11 (or 21) bit message identifier, and receiving ECUs will select messages relevant to them

based on this message identifier. Message identifiers also serve as prioritization, as the employed CSMA/CR medium access scheme will always grant priority to the message with the lowest message identifier avoiding collisions on the bus. Transport layer protocols such as ISO-TP allow for transfer of longer messages fragmented into smaller network packets and more complex forms of addressing crossing gateways connecting multiple CAN segments.

In order to maintain and manage the assignment and semantics of message identifiers, the design phase of an automotive network involves setup of a so-called CAN-Matrix that lists exactly which ECU is supposed to sent which message identifier, which ECUs will receive messages of certain type and also the payload syntax and semantics. This design is done using sophisticated tools like Vector Informatics CANOe.[5] The data provided by such tools is a perfect data source for specification-based IDS and for our approach, so the criteria of *availability* is met. *linkability* is more of a concern, as messages per se do not contain information on their source or type and a recipient needs to know (part of) the CAN matrix to identify how to decode a certain message ID. However, with the CAN matrix, we do have information on the types of ECUs available and can therefore conduct System discovery. This information can then be used to conduct Feature Lookup. A lot of relevant information (which messages are supposed to be seen on which bus segment) is again contained in the CAN matrix. Unfortunately, documentation in vehicular networks is not as standardized as the PICSs are in BACnet. So feature lookup would probably require more detailed investigations and more complex document parsing. Rule definition is then straightforward. However, having no source or destination addresses in packets, one would have to focus on message IDs, bus segments, and payload for detection.

While specification-based intrusion detection has been proposed many times especially for CAN-based networks [36, 31], a structured approach to rule-mining is missing in this domain so far and we see this as a promising field of application for our approach.

9 Conclusion

As networked control technologies are rapidly emerging, the need for securing these systems faces the key challenge of quickly scaling up to a multitude of heterogeneous devices. Our research aims to automate the deployment of effective security solutions, as well to adapt them in parallel with the monitored systems' lifecycle. More concretely, we present a novel approach to specification-based intrusion detection for NCSs. While state-of-the-art solutions exploit manually-crafted specification rules, we discuss the feasibility of automatically mining these rules from available documentation. The tests performed on real building automation systems show the effectiveness of the obtained systems and confirm the time improvement in their development and deployment.

[5] `http://vector.com/vi_canoe_en.html`

10 Acknowledgments

The authors would like to explicitly thank Dina Hadžiosmanović and Andreas Peter for the insightful discussions that gave rise to this research. Furthermore, the authors would like to acknowledge the work of Geert Jan Laanstra, Henk Hobbelink, Vincent Stoffer and Chris Weyandt at the University of Twente and the Lawrence Berkeley National Laboratory.

This research has been partially supported by the European Commission through project FP7-SEC-607093-PREEMPTIVE funded by the 7th Framework Program. This work has also been supported by the U.S. National Science Foundation under Award CNS-1314973. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors or originators, and do not necessarily reflect the views of the sponsors.

References

[1] ANSI/ASHRAE STANDARD 135-2012. A data communication protocol for building automation and control networks, 2012.

[2] BACNET INTEREST GROUP EUROPE. Engineering data exchange template for BACnet systems - "description of the EDE data fields", 2007.

[3] BALEPIN, I., MALTSEV, S., ROWE, J., AND LEVITT, K. N. Using specification-based intrusion detection for automated response. In *Recent Advances in Intrusion Detection, 6th International Symposium, RAID 2003, Pittsburgh, PA, USA, September 8-10, Proceedings* (2003), pp. 136–154.

[4] BERTHIER, R., AND SANDERS, W. H. Specification-based intrusion detection for advanced metering infrastructures. In *17th IEEE Pacific Rim International Symposium on Dependable Computing, PRDC 2011, Pasadena, CA, USA, December 12-14* (2011), pp. 184–193.

[5] CASELLI, M., HADŽIOSMANOVIĆ, D., ZAMBON, E., AND KARGL, F. On the feasibility of device fingerprinting in industrial control systems. In *Critical Information Infrastructures Security - 8th International Workshop, CRITIS 2013, Amsterdam, The Netherlands, September 16-18, Revised Selected Papers* (2013), pp. 155–166.

[6] ČELEDA, P., KREJČÍ, R., AND KRMÍČEK, V. Flow-based security issue detection in building automation and control networks. In *Information and Communication Technologies - 18th EUNICE/ IFIP WG 6.2, 6.6 International Conference, EUNICE 2012, Budapest, Hungary, August 29-31, Proceedings* (2012), pp. 64–75.

[7] CHAUGULE, A., XU, Z., AND ZHU, S. A specification based intrusion detection framework for mobile phones. In *Applied Cryptography and Network Security - 9th International Conference, ACNS 2011, Nerja, Spain, June 7-10, Proceedings* (2011), pp. 19–37.

[8] CHEUNG, S., DUTERTRE, B., FONG, M., LINDQVIST, U., SKINNER, K., AND VALDES, A. Using model-based intrusion detection for SCADA networks. In *Proceedings of the SCADA Security Scientific Symposium, Miami Beach, Florida, USA, 7 December* (2007), pp. 1–12.

[9] DENNING, D. E. An intrusion-detection model. In *Proceedings of the 1986 IEEE Symposium on Security and Privacy, Oakland, California, USA, April 7-9* (1986), pp. 118–133.

[10] DIST-1815-WG. IEEE standard for electric power systems communications-distributed network protocol (DNP3), 2012. `https://standards.ieee.org/findstds/standard/1815-2012.html`.

[11] ECHELON CORPORATION. LonTalk protocol specification v3.0, 1994. `http://www.enerlon.com/JobAids/Lontalk%20Protocol%20Spec.pdf`.

[12] EQUIPMENT, IEC TELECONTROL. Systems—part 5-104: Transmission protocols - network access for IEC 60870-5-101 using standard transport profiles.

[13] FORREST, S., HOFMEYR, S. A., SOMAYAJI, A., AND LONGSTAFF, T. A. A sense of self for Unix processes. In *IEEE Symposium on Security and Privacy, Oakland, CA, USA, May 6-8* (1996), pp. 120–128.

[14] FOVINO, I. N., CARCANO, A., MUREL, T. D. L., TROMBETTA, A., AND MASERA, M. Modbus/DNP3 state-based intrusion detection system. In *24th IEEE International Conference on Advanced Information Networking and Applications, AINA 2010, Perth, Australia, April 20-13* (2010), pp. 729–736.

[15] GILDEA, D., AND JURAFSKY, D. Automatic labeling of semantic roles. *Computational Linguistics 28*, 3 (2002), 245–288.

[16] GILL, R., SMITH, J., AND CLARK, A. J. Specification-based intrusion detection in WLANs. In *22nd Annual Computer Security Applications Conference (ACSAC 2006), Miami Beach, Florida, USA, 11-15 December* (2006), pp. 141–152.

[17] GRANZER, W., KASTNER, W., NEUGSCHWANDTNER, G., AND PRAUS, F. Security in networked building automation systems. Tech. rep., 2005.

[18] GRANZER, W., PRAUS, F., AND KASTNER, W. Security in building automation systems. *IEEE Trans. Industrial Electronics 57*, 11 (2010), 3622–3630.

[19] GRÖNKVIST, J., HANSSON, A., AND SKÖLD, M. Evaluation of a specification-based intrusion detection system for AODV. In *The Sixth Annual Mediterranean Ad Hoc Networking Workshop* (2007), pp. 121–128.

[20] GUPTA, R. A., AND CHOW, M. Networked control system: Overview and research trends. *IEEE Trans. Industrial Electronics 57*, 7 (2010), 2527–2535.

[21] HADELI, H., SCHIERHOLZ, R., BRAENDLE, M., AND TUDUCE, C. Leveraging determinism in industrial control systems for advanced anomaly detection and reliable security configuration. In *Proceedings of 12th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2009, Palma de Mallorca, Spain, September 22-25* (2009), pp. 1–8.

[22] HADŽIOSMANOVIĆ, D., BOLZONI, D., ETALLE, S., AND HARTEL, P. H. Challenges and opportunities in securing industrial control systems. In *Complexity in Engineering, COMPENG 2012, Aachen, Germany, June 11-13* (2012), pp. 1–6.

[23] HASSAN, H. M., MAHMOUD, M., AND EL-KASSAS, S. Securing the AODV protocol using specification-based intrusion detection. In *Q2SWinet'06 - Proceedings of the Second ACM Workshop on Q2S and Security for Wireless and Mobile Networks, Terromolinos, Spain, October 2* (2006), pp. 33–36.

[24] HOLMBERG, D. G., AND EVANS, D. *BACnet Wide Area Network Security Threat Assessment*. US Department of Commerce, National Institute of Standards and Technology NIST, 2003.

[25] ISO. Industrial automation systems – manufacturing message specification – part 2: Protocol specification, 2003.

[26] JIEKE, P., REDOL, J., AND CORREIA, M. Specification-based intrusion detection system for carrier ethernet. In *WEBIST 2007 - Proceedings of the Third International Conference on Web Information Systems and Technologies, Volume IT, Barcelona, Spain, March 3-6* (2007), pp. 426–429.

[27] JOKAR, P., NICANFAR, H., AND LEUNG, V. C. M. Specification-based intrusion detection for home area networks in smart grids. In *IEEE Second International Conference on Smart Grid Communications, SmartGridComm 2011, Brussels, Belgium, October 17-20* (2011), pp. 208–213.

[28] KASTNER, W., NEUGSCHWANDTNER, G., SOUCEK, S., AND NEWMAN, M. H. Communication systems for building automation and control. *Proceedings of the IEEE 93*, 6 (2005), 1178–1203.

[29] KAUR, J., TONEJC, J., WENDZEL, S., AND MEIER, M. Securing BACnet's pitfalls. In *ICT Systems Security and Privacy Protection - 30th IFIP TC 11 International Conference, SEC 2015, Hamburg, Germany, May 26-28, Proceedings* (2015), pp. 616–629.

[30] KIM, K., AND KUMAR, P. R. The importance, design and implementation of a middleware for networked control systems. *Springer Lecture Notes in Control and Information Sciences 406*, 1 (2010), 1–29.

[31] KLEBERGER, P., OLOVSSON, T., AND JONSSON, E. Security aspects of the in-vehicle network in the connected car. In *IEEE Intelligent Vehicles Symposium (IV), 2011, Baden-Baden, Germany, June 5-9* (2011), pp. 528–533.

[32] KNX ASSOCIATION. KNX Standard, 2011. `https://www.knx.org`.

[33] KO, C., BRUTCH, P., ROWE, J., TSAFNAT, G., AND LEVITT, K. N. System health and intrusion monitoring using a hierarchy of constraints. In *Recent Advances in Intrusion Detection, 4th International Symposium, RAID 2001 Davis, CA, USA, October 10-12, Proceedings* (2001), pp. 190–204.

[34] KO, C., FINK, G., AND LEVITT, K. N. Automated detection of vulnerabilities in privileged programs by execution monitoring. In *10th Annual Computer Security Applications Conference, ACSAC 1994, Orlando, FL, USA, 5-9 December* (1994), pp. 134–144.

[35] KO, C., RUSCHITZKA, M., AND LEVITT, K. N. Execution monitoring of security-critical programs in distributed systems: A specification-based approach. In *IEEE Symposium on Security and Privacy, Oakland, CA, USA, May 4-7* (1997), pp. 175–187.

[36] LARSON, U. E., NILSSON, D. K., AND JONSSON, E. An approach to specification-based attack detection for in-vehicle networks. In *IEEE Intelligent Vehicles Symposium (IV), 2008, Eindhoven, the Netherlands, June 4-6* (2008), pp. 220–225.

[37] LIN, H., SLAGELL, A. J., MARTINO, C. D., KALBARCZYK, Z., AND IYER, R. K. Adapting Bro into SCADA: Building a specification-based intrusion detection system for the DNP3 protocol. In *Cyber Security and Information Intelligence, CSIIRW '13, Oak Ridge, TN, USA, January 8-10* (2013), p. 5.

[38] LYON, G. F. *Nmap network scanning: The official Nmap project guide to network discovery and security scanning*. Insecure, 2009. `https://nmap.org/`.

[39] MANYIKA, J., CHUI, M., BUGHIN, J., DOBBS, R., BISSON, P., AND MARRS, A. Disruptive technologies: Advances that will transform life, business, and the global economy. Tech. rep., 2013.

[40] MATHERLY, J. C. SHODAN: the computer search engine, Jun 2016. `http://www.shodanhq.com/`.

[41] MODBUS-IDA. Modbus application protocol specification v1.1b3, 2012. `http://www.modbus.org`.

[42] NATIONAL JOINT APPRENTICESHIP & TECHNICAL COMMITTEE. *Building Automation: Control Devices and Applications*. American Technical Publishers, Inc., 2008.

[43] NEWMAN, M. *BACnet: The Global Standard for Building Automation and Control Networks*. Momentum Press, 2013.

[44] ORSET, J., ALCALDE, B., AND CAVALLI, A. R. An EFSM-based intrusion detection system for ad hoc networks. In *Automated Technology for Verification and Analysis, Third International Symposium, ATVA 2005, Taipei, Taiwan, October 4-7, Proceedings* (2005), pp. 400–413.

[45] PAN, Z., HARIRI, S., AND AL-NASHIF, Y. B. Anomaly based intrusion detection for building automation and control networks. In *11th IEEE/ACS International Conference on Computer Systems and Applications, AICCSA 2014, Doha, Qatar, November 10-13* (2014), pp. 72–77.

[46] PAXSON, V. Bro: A system for detecting network intruders in real-time. In *Proceedings of the 7th USENIX Security Symposium, San Antonio, TX, USA, January 26-29* (1998).

[47] PEACOCK, M. D., AND JOHNSTONE, M. N. An analysis of security issues in building automation systems.

[48] PETRONI, N. L., FRASER, T., WALTERS, A., AND ARBAUGH, W. A. An architecture for specification-based detection of semantic integrity violations in kernel dynamic data. In *Proceedings of the 15th USENIX Security Symposium, Vancouver, BC, Canada, July 31 - August 4* (2006).

[49] SALSBURY, T. I. The smart building. In *Springer Handbook of Automation*. Springer, 2009, pp. 1079–1093.

[50] SANEIFAR, H., BONNIOL, S., LAURENT, A., PONCELET, P., AND ROCHE, M. Terminology extraction from log files. In *Database and Expert Systems Applications, 20th International Conference, DEXA 2009, Linz, Austria, August 31 - September 4, Proceedings* (2009), pp. 769–776.

[51] SANNER, M. F. Python: a programming language for software integration and development. *J Mol Graph Model 17*, 1 (1999), 57–61. `https://www.python.org/`.

[52] SEKAR, R., CAI, Y., AND SEGAL, M. A specification-based approach for building survivable systems. In *Proceedings of the National Information Systems Security Conference (NISSC'98)* (1998), pp. 338–347.

[53] SEKAR, R., GUPTA, A. K., FRULLO, J., SHANBHAG, T., TIWARI, A., YANG, H., AND ZHOU, S. Specification-based anomaly detection: A new approach for detecting network intrusions. In *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, November 18-22* (2002), pp. 265–274.

[54] SEKAR, R., AND UPPULURI, P. Synthesizing fast intrusion prevention/detection systems from high-level specifications. In *Proceedings of the 8th USENIX Security Symposium, Washington, D.C., August 23-26* (1999).

[55] SOMMER, R., AMANN, J., AND HALL, S. Spicy: A unified deep packet inspection framework dissecting all your data. Tech. rep., ICSI, 2015. TR-15-004.

[56] SONG, T., KO, C., TSENG, C. H., BALASUBRAMANYAM, P., CHAUDHARY, A., AND LEVITT, K. N. Formal reasoning about a specification-based intrusion detection for dynamic autoconfiguration protocols in ad hoc networks. In *Formal Aspects in Security and Trust, Third International Workshop, FAST 2005, Newcastle upon Tyne, UK, July 18-19, Revised Selected Papers* (2005), pp. 16–33.

[57] STRZALKOWSKI, T. Natural language information retrieval. *Inf. Process. Manage. 31*, 3 (1995), 397–417.

[58] SZLÓSARCZYK, S., WENDZEL, S., KAUR, J., MEIER, M., AND SCHUBERT, F. Towards suppressing attacks on and improving resilience of building automation systems - an approach exemplified using BACnet. In *Sicherheit 2014: Sicherheit, Schutz und Zuverlässigkeit, Beiträge der 7. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik e.V. (GI), 19.-21. März 2014, Wien, Österreich* (2014), pp. 407–418.

[59] TRUONG, P., NIEH, D., AND MOH, M. Specification-based intrusion detection for H.323-based voice over IP. In *Proceedings of the Fifth IEEE International Symposium on Signal Processing and Information Technology (ISSPIT 2005), Athens, Greece, December 18-21* (2005), pp. 387–392.

[60] TSENG, C., BALASUBRAMANYAM, P., KO, C., LIMPRASITTIPORN, R., ROWE, J., AND LEVITT, K. N. A specification-based intrusion detection system for AODV. In *Proceedings of the 1st ACM Workshop on Security of ad hoc and Sensor Networks, SASN 2003, Fairfax, Virginia, USA* (2003), pp. 125–134.

[61] TSENG, C. H., SONG, T., BALASUBRAMANYAM, P., KO, C., AND LEVITT, K. N. A specification-based intrusion detection model for OLSR. In *Recent Advances in Intrusion Detection, 8th International Symposium, RAID 2005, Seattle, WA, USA, September 7-9, Revised Papers* (2005), pp. 330–350.

[62] UPPULURI, P., AND SEKAR, R. Experiences with specification-based intrusion detection. In *Recent Advances in Intrusion Detection, 4th International Symposium, RAID 2001 Davis, CA, USA, October 10-12, Proceedings* (2001), pp. 172–189.

[63] WENDZEL, S., KAHLER, B., AND RIST, T. Covert channels and their prevention in building automation protocols: A prototype exemplified using BACnet. In *2012 IEEE International Conference on Green Computing and Communications, Conference on Internet of Things, and Conference on Cyber, Physical and Social Computing, GreenCom/iThings/CPSCom 2012, Besancon, France, November 20-23* (2012), pp. 731–736.

[64] ZALEWSKI, M. P0f: Passive OS fingerprinting tool, 2006. `lcamtuf.coredump.cx/p0f3/`.

[65] ZHANG, P. *Industrial Control Technology: A Handbook for Engineers and Researchers*. William Andrew Inc., 2008.