Towards Automated Identification and Assessment of Security Weaknesses in Smart Buildings

Herson Tobias Esquivel Vargas

TOWARDS AUTOMATED IDENTIFICATION AND ASSESSMENT OF SECURITY WEAKNESSES IN SMART BUILDINGS

Herson Tobias Esquivel Vargas

TOWARDS AUTOMATED IDENTIFICATION AND ASSESSMENT OF SECURITY WEAKNESSES IN SMART BUILDINGS

DISSERTATION

to obtain
the degree of doctor at the University of Twente,
on the authority of the rector magnificus,
prof.dr.ir. A. Veldkamp,
on account of the decision of the Doctorate Board,
to be publicly defended
on Friday 25th of November 2022 at 16.45 hours

by

Herson Tobias Esquivel Vargas

born on the 6th of October, 1985
in Cartago, Costa Rica

This dissertation has been approved by:

Supervisors:
prof.dr. A. Peter
prof.dr. P.H. Hartel

Co-supervisor:
dr. M. Caselli

UNIVERSITY OF TWENTE.

Services and CyberSecurity Group
P.O. Box 217, 7500 AE
Enschede, the Netherlands

DIGITAL SOCIETY INSTITUTE

DSI Ph.D. Thesis Serie No. 22-007
Digital Society Institute
P.O. Box 217, 7500 AE
Enschede, the Netherlands

Cover design: Antonio Salazar Chinchilla and Herson Tobias Esquivel Vargas
Printed by: Gildeprint
ISSN: 2589-7721
ISBN: 978-90-365-5449-7
DOI: 10.3990/1.9789036554497
https://doi.org/10.3990/1.9789036554497

Graduation Committee:

Chair / secretary	prof.dr. J.N. Kok	Universiteit Twente
Supervisors	prof.dr. A. Peter	Universität Oldenburg
	prof.dr. P.H. Hartel	Universiteit Twente
Co-supervisor	dr. M. Caselli	Siemens AG
Committee Members	prof.dr.ir. A. Pras	Universiteit Twente
	prof.dr.ir. M.R. van Steen	Universiteit Twente
	dr. C. Alcaraz	Universidad de Málaga
	dr. N.O. Tippenhauer	CISPA - Helmholtz-Zentrum für Informationssicherheit
	prof. dr. S. Etalle	Technische Universiteit Eindhoven

A mamá.

Acknowledgements

I started my study-abroad adventure in the autumn of 2014. Back then I was not even used to split time in seasons other than dry and rainy. In any case, this was the least of all changes in my life. My plan was very simple: to start a specialization in cybersecurity with a masters program followed by a PhD track. But life also had plans for me.

This journey allowed me to meet many kind and smart people.

- To my supervision team: Andreas, Marco, and Pieter, you taught me to do research, to look at problems from different angles, and to pay special attention to the details at work. Thank you for your patience and support. I feel extremely lucky to have had you as my mentors. Andreas, besides your exceptional academic guidance you were always there to lend an empathetic ear when I needed it. I learned so much from you. Thank you very much. Marco, during your last months at the UT you introduced me to the topic of cyber-physical systems security. Back then it was something new to me that ended up fascinating me. Thanks a lot for staying along the way. Pieter, I got here in the first place because you and Raymond decided to give me the chance to enroll in the Kerckhoffs program. You triggered it all. Thank you very much for the opportunity.

- To my fellow PhD/Postdoc students: Ali, Amina, Bence, Philipp, Riccardo, Roeland, Thijs, Tim, Valeriu, and Yoep, it was very fun to work with all of you, not to mention the occasional *bitterballen*, chess/board games, and joyful coffee breaks. I'm gonna miss the *Twente Hacking Squad* meetings and the CTF weekends. I would also like to thank Asbat, Chakshu, Chris, Claudio, Dan, Donika, Farideh, Federico, Jerre, João, Kemilly, Lu, Meike, Meikel, Prince, Susanne, Una, and Zsolt.

- To the management and support staff: Bertine, Jeanette, and Suse, you helped me sort out all my administrative issues since the very beginning. I will always be thankful with you. Geert Jan, I enjoyed and learned so much building the testbed with you in the lab. Lorena, it was great to meet you. Thanks a lot for all your help. There aren't many Costa Ricans in Enschede, let alone in the university!

Many more people were an important part of my life during this adventure. In the Netherlands, Yazan and Sandip, it was great meeting you during the masters but it is even better that we still meet every time we can! In Costa Rica, Arturo, Aurelio, Daniel, Kattia, and Marcela, you made me feel at home while I was away. To my sister, Francela, thank you for being always there for me. Kevin and Nereo, you are like family to me regardless of where we are. I have no words to express my gratitude to all of you.

Finally, I would like to thank my wonderful wife, Liseth, without whom none of this would have been possible. Your unconditional support helped me keep on going during difficult moments (and there were quite some). During this time we became parents and learned so many things about us and about love. Inti and Sofía, you unknowingly became my reason to be. There is a world of wonder ahead of you. I wish you the happiest life. I love the three of you with all my heart.

Cartago, Costa Rica, 10/10/2022

Abstract

Smart buildings are equipped with computer systems that monitor and control diverse services such as air conditioning, indoor transportation, physical access control, and many others. Critical infrastructures like hospitals, airports, and data centers, leverage on such services to support their daily operations. However, the current popularity of smart buildings is founded on a decades-long history. Smart building systems have evolved from isolated networks using proprietary protocols to IT-integrated systems that use standardized communication protocols. They might even be connected to the Internet to allow remote building management. This transition has exposed smart buildings to a whole new set of security threats. For instance, there have been documented cases where attackers have managed to remotely disrupt the environmental conditions and physical access control of smart buildings. Due to the crucial role that smart buildings play in supporting organizations and the serious threat of cyber attacks against them, there is a pressing need to investigate how to improve their current security posture.

The transfer of mature IT security solutions to smart building systems seems a natural approach to enhance their security, however, the fundamental differences between both domains often require significant adaptation effort or to develop completely new solutions. For this reason, in recent years, a growing body knowledge about smart buildings security has been developed. However, most of these solutions have focused on intrusion *detection* and little efforts have been made to *prevent* cyber attacks. An effective way to prevent cyber attacks against smart buildings is by preemptively handling security weaknesses in customized applications and configurations that run the system. Unfortunately, this is often overlooked by smart building administrators due to, e.g., lack of specialized tools, staff, and training. We identify not only a research gap regarding this important task, but also an urgent need to provide (semi-) automated tools that help overcome the limitations faced by smart building administrators. The implementation of these tools requires sophisticated methods that incorporate technical and business-related insights to handle weaknesses according to the organization's best interest.

In this thesis, we investigate how to implement the first stages of a vulnerability management process for smart building applications and configurations. Beyond just vulnerabilities, we consider the *weaknesses* that give rise to vulnerabilities. In particular, our contributions address the *identification* and *assessment* of security

weaknesses for later remediation. These are two key activities to preemptively strengthen the security of smart buildings. The *identification* of weaknesses is the basis of any vulnerability management process as it provides the first insights about the current security state of a system. This is a challenging task because a deep understanding of the system's inner workings is often needed to obtain meaningful findings. We propose two approaches to identify security weaknesses; one focused on smart building applications and another on smart building configurations. In the first case, we model the application as a graph data structure comprised of sensors, setpoints, actuators, and control function nodes. The relationships among these components reveal the architecture of the system, which can then be analyzed in the search for security weaknesses. In the second case, we look for component misconfigurations that can be observed in their behavior, i.e., the way they interact with other components in the system. Leveraging official documentation from the components' manufacturers, we create a model of valid behavior for each of them, which is then compared with their actual behavior as observed in the network traffic. After identifying security weaknesses, we *assess* the sensitivity of the affected components, which is an important factor to prioritize weaknesses for later remediation. We propose a comprehensive approach to assess the sensitivity of smart building components based on technical and business-related features. The proposed methods are evaluated in real smart buildings and additional experiments are performed in testbeds and comparable simulated environments. These evaluations confirm the feasibility and effectivity of our (semi-) automated weakness identification and assessment approaches.

Samenvatting

Slimme gebouwen zijn uitgerust met computersystemen die diverse diensten, zoals airconditioning, transport binnen het gebouw, fysieke toegangssystemen en vele andere, monitoren en aansturen. Vitale infrastructuur zoals ziekenhuizen, luchthavens en datacenters gebruiken zulke diensten om hun dagelijkse activiteiten te ondersteunen. De huidige populariteit van slimme gebouwen heeft echter een decennia lange geschiedenis. Systemen van slimme gebouwen zijn geëvolueerd van geïsoleerde netwerken die gebruik maken van gesloten protocollen, naar ICT-geintegreerde systemen met gestandaardiseerde communicatieprotocollen. De systemen kunnen zelfs verbonden zijn met het internet om gebouwbeheer op afstand mogelijk te maken. Deze ontwikkelingen stellen slimme gebouwen bloot aan een heel nieuwe vorm van beveiligingsbedreigingen. Zo zijn er gevallen bekend van aanvallers die op afstand het binnenklimaat en het fysieke toegangssysteem hebben verstoord. Vanwege de belangrijke rol die slimme gebouw hebben in het ondersteunen van organisaties en de ernstige dreiging van digitale aanvallen tegen hen, is er een dringende behoefte om te onderzoeken hoe de huidige beveiliging kan worden verbeterd.

Het overbrengen van bewezen ICT-beveiligingsoplossingen naar het domein van slimme gebouwen lijkt een logische manier om de beveiliging te verbeteren, maar fundamentele verschillen tussen beide domeinen vragen vaak om een aanzienlijke inspanning om de aanpassing te maken of om het ontwikkelen van compleet nieuwe oplossingen. Om deze reden is in de recente jaren meer kennis opgebouwd over de beveiliging van slimme gebouwen. De meeste van deze oplossingen richten zich echter op het *detecteren* van aanvallen, terwijl slechts enkele zich richten op het *voorkomen* van digitale aanvallen. Een effectieve manier om digitale aanvallen tegen slimme gebouwen te voorkomen is door preventief zwakheden in de beveiliging van aangepaste applicaties en in systeemconfiguraties aan te pakken. Helaas wordt dit vaak over het hoofd gezien door de beheerders van slimme gebouwen vanwege, bijvoorbeeld, een gebrek aan gespecialiseerde tools, personeel of opleiding. Wij identificeren niet enkel een gebrek aan onderzoek ten aanzien van deze belangrijke taak, maar ook een dringende behoefte aan (semi-)geautomatiseerde tools die helpen de beperkingen te overkomen waarmee de beheerders van slimme gebouwen worden geconfronteerd. De implementatie van deze tools vereisen geavanceerde methoden die technische en bedrijfsgerelateerde inzichten bevatten om zwakheden

op de beste manier voor de organisatie aan te pakken.

In dit proefschrift onderzoeken we hoe we de eerste stappen van een kwetsbaarheidsbeheerproces voor applicaties en configuraties van slimme gebouwen kunnen implementeren. Naast kwetsbaarheden kijken we naar de *zwakke punten* die aanleiding geven tot kwetsbaarheden. Onze bijdragen gaan met name in op de *identificatie* en *beoordeling* van zwakke punten in de beveiliging om deze later aan te kunnen pakken. Dit zijn twee kernactiviteiten om preventief de beveiliging van slimme gebouwen te versterken. De *identificatie* van zwakke punten is de basis van elk kwetsbaarheidsbeheerproces, omdat het de eerste inzichten verschaft over de huidige beveiligingsstatus van een systeem. Dit is een uitdagende taak omdat het voor zinvolle bevindingen vaak nodig is om de innerlijke werking van het systeem te doorgronden. We stellen twee benaderingen voor om zwakke punten in de beveiliging te identificeren: de een gericht op applicaties en de andere op configuraties van slimme gebouwen. In het eerste geval modelleren we de applicatie als een graaf bestaande uit knopen voor sensoren, richtwaarden, actuatoren en regelfuncties. De relaties tussen deze componenten onthullen de architectuur van het systeem, die vervolgens kunnen worden geanalyseerd bij het zoeken naar zwakke punten in de beveiliging. In het tweede geval zoeken we naar misconfiguraties van componenten die kunnen worden waargenomen in hun gedrag, d.w.z. de manier waarop ze omgaan met andere componenten in het systeem. Gebruikmakend van officiële documentatie van de fabrikanten van componenten, creëren we een model van valide gedrag voor elk van de componenten, om het vervolgens te vergelijken met hun werkelijke gedrag zoals waargenomen in het netwerkverkeer. Nadat we zwakke punten in de beveiliging hebben geïdentificeerd, *beoordelen* we de gevoeligheid van de getroffen componenten, wat een belangrijke factor is voor de prioritering van het aanpakken van het zwakke punt. We stellen een alomvattende aanpak voor om de gevoeligheid van componenten in slimme gebouwen te beoordelen op basis van technische en bedrijfsgerelateerde kenmerken. De voorgestelde methoden worden geëvalueerd in bestaande slimme gebouwen en aanvullende experimenten worden uitgevoerd in testopstellingen en vergelijkbaar gesimuleerde omgevingen. Deze evaluaties bevestigen de haalbaarheid en effectiviteit van onze (semi-)geautomatiseerde benaderingen voor het identificeren en beoordelen van zwakke punten.

Contents

Acronyms

AI artificial intelligence. 3, 114

ANSI American National Standards Institute. 16, 94, 95

ASHRAE American Society of Heating, Refrigerating and Air-Conditioning Engineers. 16, 94, 95

BACnet building automation and control networks. 15–20, 24, 27–29, 31, 51–61, 63, 64, 66–72, 82–84, 98–100, 104, 111, 112

BACS Building Automation and Control Systems. 15

BCMS business continuity management system. 74, 76, 90

BIA business impact analysis. 74–77, 82, 83, 89–92, 102, 103, 113

BIBBs BACnet interoperability building blocks. 17, 52, 55, 56, 58, 60, 63, 68

CCL closed control loop. 21–29, 31–34, 36, 37, 46, 48, 49, 110, 111, 114

CPS Cyber-Physical System. 13, 14, 18, 19, 21–27, 30, 31, 42, 45–49, 89, 111, 114

CVE Common Vulnerabilities and Exposures. 4, 21, 69, 88, 110, 115

CVSS Common Vulnerability Scoring System. 88, 90, 110

CWE Common Weakness Enumeration. 6, 8, 24, 47–49, 51, 52, 70–72, 111, 112, 114

DoS denial of service. 67, 72

FSM finite-state machine. 28, 29

HMI human-machine interface. 4, 6, 8, 19, 45, 51, 52, 69

HVAC heating, ventilation, and air conditioning. 1, 14

ICS Industrial Control System. 8–10, 13, 14, 18–21, 31, 34, 45, 47–49, 88, 106, 110, 111, 114

IEC International Electrotechnical Commission. 18

IEEE Institute of Electrical and Electronics Engineers. 68

IESNA Illuminating Engineering Society of North America. 95

IoT Internet of Things. 15, 48, 72, 107

IP Internet Protocol. 15, 20, 66, 99

ISACA Information Systems Audit and Control Association. 74

ISO International Organization for Standardization. 15, 16, 51, 68, 74, 75, 89

IT information technology. 2, 3, 8, 9, 13, 18, 20, 24, 42, 47, 49, 51, 88, 109–111

NIST National Institute of Standards and Technology. 4, 7, 89

OS operating system. 3, 4, 6, 20, 66, 67, 69

OT Operational Technology. 13, 18, 51, 88, 92, 106, 114

OWS operator work station. 19, 99, 112

P&ID piping and instrumentation diagram. 24, 26, 45, 46, 48

PDF Portable Document Format. 7, 9, 10, 51, 59, 112

PICS Protocol Implementation Conformance Statement. 52–61, 63, 64, 67–72

PLC Programmable Logic Controller. 13, 18, 19, 22, 45, 46, 74

RFC Request For Comments. 5, 68, 110

SCADA Supervisory Control and Data Acquisition. 18, 19

SDT shutdown time. 36–38, 41, 42, 46

SQL Structured Query Language. 58

TCP Transmission Control Protocol. 15, 20

TEP Tennessee Eastman Plant. 31, 34, 36, 37, 45–47, 49

Chapter 1

Introduction

Studies have found that people spend up to 90% of the time indoors [52]. Behind the scenes, building managers must ensure the right conditions for building's occupants at all times. Building management involves the control of long-established services such as heating, ventilation, and air conditioning (HVAC) and, more recently, additional services such as indoor transportation (e.g., elevators, escalators, travelators), video surveillance, physical access control, and many more. Moreover, building managers must ensure energy efficiency to minimize the operation costs of buildings. Building management is a challenging task not only for the high rate of occupancy in buildings, but also due to the ever growing list of services that must be provided in the most efficient way.

The automated control of building services came to be the solution for building management but it also brought new problems. On the one hand, automated control relieves building managers from repetitive tasks; for example, turning the lights on and off during specific time slots. On the other hand, each automated service is isolated from the rest, causing difficulties in their coordination. In this setting, it is not uncommon to find air conditioning systems fighting out heating systems while both of them try to reach mismatching temperature setpoints [14]. Conflicting parameters like these severely hinder the realization of energy efficient buildings.

The integration of diverse building services lies at the core of the *smart building* concept. According to [137], a smart building is "a building equipped with integrated technology systems such as building automation, life safety, telecommunications, user systems, and facility management systems". Unlike the isolated building services approach, smart buildings exchange information between different subsystems to provide the required services while achieving energy efficiency.

The advantages of smart buildings have boosted their popularity all over the world. A market research report by *Fortune Business Insights* states that the smart

building industry is expected to grow 21.6% in the next 6 years to become a $265.37 USD billion industry [63]. As part of its expected business success, this particular report highlights the demand of new building services caused by the ongoing COVID-19 pandemic.

1.1 Motivation

The development of smart building technologies has been led by a constant market pressure for new building services, neglecting important security aspects [136]. In the past, the lack of security was not considered a problem because smart buildings were implemented using isolated networks. Moreover, the use of proprietary protocols gave some "security by obscurity" sense of security. These two aspects, however, have drastically changed in the last 30 years with the development of standard protocols and the integration of IT and smart building networks [15, 119]. Additionally, the convenience of remote building monitoring and control has led to Internet connected buildings, opening up to a whole new set of threats.

Smart buildings are attractive targets for cyber attackers. Smart buildings not only lack security services commonly available in IT environments but, typically, also lack specialized and dedicated security staff [138]. Furthermore, the consequences of cyber attacks on smart buildings extend beyond the cyber domain to the physical world, potentially harming people, assets, and disrupting the environmental conditions required by organizations [136]. Lastly, there are buildings such as hospitals, airports, and data centers, often regarded as critical infrastructure. For these reasons, smart building attacks might not only have a high impact but also a high chance of success, thus, making them attractive targets for cyber attackers.

Attacks on buildings can have serious consequences. For instance, an attack on an airport ventilation subsystem can be extremely costly since natural ventilation (e.g., windows) is typically unavailable due to physical security reasons. After such an attack, the resulting high level of CO_2 in crowded areas and its effect on human health, would likely lead to the shutdown of the affected space [71]. Attacks in hospitals can also have serious consequences with the aggravating circumstance that the effect might be suffered by ill patients; for example, tampering with services such as ventilation and illumination in operating rooms or manipulating the lights to trigger seizures in people suffering from photosensitive epilepsy [116].

Attacks against smart buildings are happening. A 2 019 report by Kaspersky details that 37,8% of computers in building-based automation systems were targeted by cyber attacks [82]. Although the number of smart building attacks is likely under-reported, there have been documented cases. In one case from 2 016, a cyber

attack affected the heating subsystem in a residential building in Finland during winter [93]. One year later, attackers remotely locked all the doors in a hotel in Austria until a ransom was paid [22]. Still another case happened in 2 018, where the temperature of a storage room in a dutch supermarket was maliciously modified spoiling medicines and food [128].

Preventing attacks is, generally, cheaper than dealing with them. In the IT domain, it is estimated that attacks could be prevented with only one fifth of the overall cost of dealing with the attack [88]. *While most security research in the smart buildings domain focuses on attack detection [29, 43, 44, 105, 132, 144], there is a gap with respect to attack prevention controls.*

Security controls that help to prevent attacks are selected on the basis of a risk management process. Traditional risk management involves the identification, assessment, and treatment of risks [98]. Such risks originate from *vulnerabilities* in assets that might be exploited by threats [73]. In this context, **vulnerability management** is one of the most important attack prevention controls implemented in IT systems. Due to its relevance, different industries such as, e.g., the card payment industry, require the implementation of a vulnerability management process [37]. Vulnerability management is comprised of three steps continuously repeated:

1. *Vulnerability identification.* It is crucial to locate the vulnerabilities in order to manage them. The identification of vulnerabilities starts with an inventory of all assets in the system along with their technical features such as the operating system (OS), network services, applications, and configurations. These features constitute a coarse classification of the sources of vulnerabilities. The identification of vulnerabilities in IT environments is typically done using tools like vulnerability scanners and input fuzzers [94, 109, 110]. More recent approaches use artificial intelligence (AI) to identify weaknesses and vulnerabilities [143]. The output of this subprocess is an aggregated set of the vulnerabilities found by different means.

2. *Vulnerability prioritization.* Among all the vulnerabilities previously identified, it is important to know which of them must be fixed first. Although, ideally, all vulnerabilities should be fixed, this is hardly the case in practice; for instance, due to limited resources to fix them (e.g., time, money, staff), technical difficulties (e.g., firmware updates), organizational risk appetite, among others. Existing methods to prioritize vulnerabilities take into consideration different criteria such as severity [47], expected exploitation time [97], estimated risk [115], and even social media discussions [32]. Whereas some proprietary tools home brew secret methods [109, 110, 129], more recent

approaches look at a combination of different criteria [9]. The outcome of this subprocess is a ranked list of vulnerabilities.

3. *Vulnerability remediation.* The remediation of vulnerabilities improves the security of the analyzed system. It involves the patching or reconfiguration needed to fix the vulnerabilities found in the first step. This stage closes the vulnerability management cycle.

To apply the vulnerability management process on smart buildings, there are challenges to solve in each of the three steps. First, regarding the *identification* of vulnerabilities, the applicability of tools such as vulnerability scanners and input fuzzers is limited. On the one hand, vulnerability scanners usually search for Common Vulnerabilities and Exposures (CVEs) affecting the system under scrutiny. The main disadvantage of this approach is that vulnerability scanners will, at best, find vulnerabilities in generic smart building software such as the OSs and network services pre-installed in building controllers, operators' software, human-machine interface (HMI) firmware, and others, but not in smart building *applications* and smart building *configurations* that are specific for each building and, therefore, lack CVEs. On the other hand, fuzzers submit malicious, malformed, or simply random data as inputs to the target system to discover vulnerabilities [94]. However, it has been documented that some resource constrained devices in smart building networks are not robust enough to deal with abnormal traffic [23]. Thus, fuzzing might actually exploit vulnerabilities rather than just identify them as the vulnerability management process intends to, which makes it unfit for smart building networks. For these reasons, vulnerability scanners and input fuzzers are insufficient and unfit to find vulnerabilities in smart buildings.

Second, existing methods to *prioritize* vulnerabilities are unfit for the smart buildings domain as they do not capture the impact of cyber-physical attacks (e.g., physical influence, environmental damage, safety threats, etc.). The National Institute of Standards and Technology (NIST) has published a "Framework for Improving Critical Infrastructure Cybersecurity" that recommends the use of business drivers to guide cybersecurity activities such as the prioritization of vulnerabilities [21]. However, it does not state how to do it nor is there any established method to do it in the smart buildings domain.

Third, in smart buildings we identify two types of *remediations* depending on where the underlying vulnerability has been found. For one thing, vulnerabilities in generic smart building software are typically fixed with patches provided by the device or software manufacturer. For another, vulnerabilities in customized applications or configurations must be fixed in-house by modifying the affected components. However, there are problems to implement both kinds of remediations.

Lacking in-house resources to find and fix vulnerabilities in custom applications and configurations might cause long delays to deploy necessary remediations. Moreover, and regardless of the source of vulnerabilities, the application of remediations might cause building services outages that negatively affect the supported business processes.

Just like in the general IT domain, the availability of (semi-) automated tools is crucial to successfully develop a vulnerability management process for smart buildings. Being a cyclic process, typically underfunded, and with scarce security specialists, *automation* becomes an essential feature to keep the vulnerability management process continuously running. It is also an enabling feature for those organizations that have not yet implemented a vulnerability management process.

According to RFC 4949, vulnerabilities are exploitable weaknesses in a system [124]. Therefore, vulnerabilities are a subset of a system's security weaknesses. In this thesis, we explore how to implement a vulnerability management process for smart buildings. However, we widen the reach of the typical vulnerability management process to the superset of *security weaknesses*. The motivation behind this decision is to address the source of vulnerabilities before they are recognized as such; a process that often involves risky exploitation attempts in real infrastructures [49]. Moreover, the limited built-in security services in smart buildings render weaknesses as likely exploitable.

1.2 Scope and Aim of the Thesis

The aim of our work is to improve the current security posture of smart buildings. To do so, we focus on the implementation of a *weakness-extended* vulnerability management process for smart buildings, an attack prevention control that has been understudied in this domain despite its importance. In particular, our research aims to solve problems in the first two stages of the vulnerability management process, namely, the *identification* and *prioritization* of security weaknesses, while discussing future research directions regarding their *remediation*. Given that the prioritization of security weaknesses is a multi-dimensional task, we center our attention on one key aspect, i.e., the *assessment* of smart building components potentially affected by weaknesses. A transversal objective of this thesis is to investigate how to (semi-) automate the identification and assessment of security weaknesses. We now elaborate on the scope defined for the weakness identification and weakness assessment activities.

1.2.1 Weakness Identification

We focus on the analysis of building-specific sources of vulnerabilities, namely, smart building *applications* and *configurations*. The identification of weaknesses in generic smart building software (e.g., OSs, network services, etc.) is out of the scope of this work as there are already tools that cover those [30, 99]. We focus on building-specific weaknesses to contribute filling the research gap regarding the security of software and configurations that are unique to each building. Moreover, and in contrast to generic software weaknesses, building-specific weaknesses might lead to stealthier and more targeted attacks as these are closer to the physical processes.

By weaknesses in smart building *applications* we refer to flaws in the software that controls building services that adversaries could leverage to execute attacks. This software is fully customized for each building and responds to the specific requirements of the organization hosted in the building. A concrete example of a smart building application is one that starts the ventilation and air conditioning of a specific room if it has been previously booked in the facilities management system by an authorized user. A smart building application is distributed horizontally among building controllers (e.g., the ventilation and air conditioning controllers) and vertically between building controllers and management devices such as, user directories, historians, operator workstations, and HMIs. Weaknesses in smart building applications can be inadvertently introduced by local programmers or external staff in charge of the smart building operation. We aim to analyze system-wide smart building applications to identify flaws in their design. Since the system design might not be properly documented (e.g., unavailable or outdated documentation) and there are no standardized programming languages from which this design can be extracted, the design may need to be reverse engineered from the component interactions in the smart building. Thus, the first challenge is how to obtain a— possibly simplified yet still useful —model of the system design. The second challenge is how to identify software weaknesses (e.g., from Mitre's Common Weakness Enumeration (CWE) database) in the aforementioned model of the system.

By weaknesses in the smart building *configuration* we refer to misconfigurations in hardware or software components that adversaries could leverage to execute or conceal attacks. Misconfigurations can be unwittingly caused by local staff, outsourced companies, or device manufacturers. The configuration of smart building components determines their *behavior*, for example, the way in which they interact with other components. Our goal is to analyze the behavior of individual components such as smart building controllers, operator workstation software, HMI's, and others, in order to identify anomalies. To do so, we aim to automate

the creation of a *valid behavior model* for all the components passively observed in the smart building's network traffic. The valid behavior model must be extracted from trustworthy sources such as official documentation written by the components' manufacturers. This task is challenging because the documentation is typically written using non-standard layouts and published as PDF files, far from ideal for machine readability. The scalability provided by automation is crucial to cope with the diversity of devices observed in dynamic smart building networks.

1.2.2 Weakness Assessment

As it was discussed before, there are different methods to prioritize vulnerabilities ranging from severity to social media discussions, expected exploitation time, and many others. We argue that a comprehensive estimation of risk should be the path towards the prioritization of weaknesses in smart buildings. A comprehensive risk estimation involves up to 29 factors that influence the *probability* and *impact* of loss events [73]. One of the crucial factors to quantify the impact is the *sensitivity* of assets from the organization's perspective. This view is in line with NIST's advice on using business drivers to guide cybersecurity activities [21].

Our goal is to assess the importance of smart building components according to their contribution to the business processes. Thus, given a weakness on a specific component, we can estimate its potential consequences for the organization hosted in the smart building. In essence, we aim at creating a sensitivity assessment for smart building components. Such an assessment should contemplate not only technical details (e.g., communication dependencies), but also environmental aspects (e.g., business processes, their scheduling, and others).

1.3 Research Questions

Based on the existing problems that hinder the implementation of a *weakness-extended* vulnerability management process for smart buildings, this work focuses on the following main research question:

How to (semi-) automatically identify and assess security weaknesses in smart building applications and their configuration?

To address this general research question, we perform a set of studies in real smart building infrastructures complemented with experiments in testbeds and simulated environments. These studies focus on two domains, namely, the identification and assessment of weaknesses.

To address the *identification* of weaknesses we propose two specific research questions. The first question aims at identifying security weaknesses in smart building *applications*, whereas the second question focuses on the identification of security weaknesses in the *configuration* of smart building components:

Research Question 1. *How to (semi-) automate the identification of weaknesses in smart building applications?*

Research Question 2. *How to (semi-) automate the identification of weaknesses in smart building configurations?*

To address the *assessment* of weaknesses, we propose an additional research question:

Research Question 3. *How to (semi-) automate a sensitivity assessment for smart building components?*

The goal of *Research Question 1* is to explore the creation of a (semi-) automated method to analyze smart building applications in the search for weaknesses. We do so in two steps. First, we show how to automatically create an abstract representation of the information flow in the system as a graph data structure; and then, we illustrate how to search for weaknesses in this graph. These weaknesses are taken from Mitre's Common Weakness Enumeration (CWE) database and, despite being originally developed for the IT domain, we show their relevance in cyber-physical systems such as smart buildings and even Industrial Control Systems (ICSs). Concretely, we illustrate how to encode weaknesses as node patterns to be matched against the graph that models the system. Successful pattern matches reveal the occurrence of weaknesses.

Research Question 2 tackles the problem of finding configuration weaknesses in smart building components such as controllers, HMIs, operator workstation software, among others. Configuration weaknesses might be exploited by adversaries to execute attacks as effectively as smart building application weaknesses. We focus on configuration weaknesses that can be observed in the network communication between smart building components. In particular, we search for components that exhibit (1) invalid behavior; and (2) an erroneous assumption on the behavior of other components. To identify invalid behavior it is possible to compare each component's documented capabilities with their actual behavior. To create a model of a component's valid behavior from its documentation, we (semi-) automate the

interpretation of technical documents in PDF format. After the model for each component is created, it is possible to search for invalid behavior that could indicate configuration weaknesses.

Finally, *Research Question 3* deals with the creation and definition of guiding principles to (semi-) automate a sensitivity assessment of smart building components, deemed crucial in a risk-based weakness prioritization strategy. Such an assessment must be based on technical and environmental aspects unique to the organization hosted in the smart building. Among the technical aspects, we consider the dependencies between different components to create a graph data structure. These dependencies are important because the consequences of exploited weaknesses might trigger a cascade effect on other components. Among the environmental aspects, we consider the business processes supported by the smart building and their relevance for the organization. We then link the smart building components (nodes in the graph) with their corresponding business processes and annotate the nodes with the corresponding business processes' relevance. The final sensitivity assessment for each component is computed by a graph centrality algorithm designed specifically to this end.

1.4 Thesis Overview and Contributions

We begin by presenting background information in Chapter 2, which is needed to understand the remainder of the thesis. This background information includes diverse aspects of smart buildings and neighboring fields such as Industrial Control Systems (ICSs). In Chapter 3, we present a method to identify weaknesses in smart building *applications*. Still in the weakness identification domain, in Chapter 4, we present an approach to identify *configuration* weaknesses in smart buildings. In Chapter 5, we introduce a weakness sensitivity assessment for smart building components, which is an important factor to prioritize weaknesses for later remediation, followed by Chapter 6, where we extend our analysis of the assessment proposed in Chapter 5. Finally, we end the thesis in Chapter 7 by presenting our conclusions and future research directions. Figure 1.1 shows an overview of the thesis contributions, which we now discuss in more detail.

Chapter 3 – Weakness Identification in Smart Building Applications We present a method to identify weaknesses in application software controlling smart buildings. To do so, we consider the application software as a unified computer program like those in the IT domain. Such parallelism between IT systems and smart building systems inspired the identification of weaknesses in the latter by

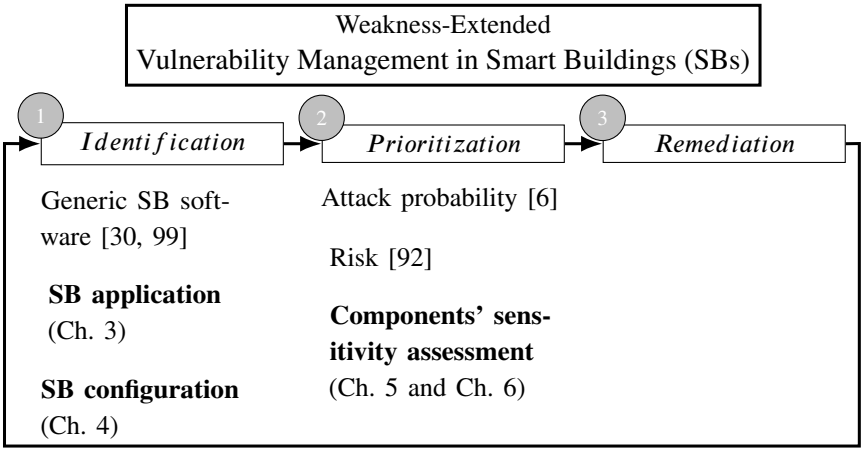


Figure 1.1: Overview of thesis contributions. The weakness-extended vulnerability management process is comprised of 3 steps continuously repeated: identification, prioritization, and remediation of weaknesses. Under each step we list existing work (if any) in the smart buildings domain. We highlight in bold font the contributions detailed in this thesis.

applying well established principles developed in the former. Moreover, we show its applicability to the Industrial Control Systems domain. This chapter is based on the contents of two papers published at refereed conferences [3, 5].

Chapter 4 – Weakness Identification in Smart Building Configurations We develop an approach to identify configuration weaknesses in smart building components by looking at their network traffic behavior and comparing it with their documented behavior. This documentation is considered the ground truth of valid behavior. We focus on the automated extraction of the valid behavior model. This is done by mining from each component's documentation (in PDF format), the information about the component's behavior. This chapter is based on the contents of a paper published at a refereed workshop [2].

Chapter 5 – BACRank: A Sensitivity Assessment of Smart Building Components We describe a sensitivity assessment method for smart building components. These components are arranged as nodes in a graph data structure where the edges represent functional dependencies between them. We annotate the nodes according to their contribution to different business activities and assign weights to the edges based on the strength of the dependencies. Finally, at the core of the proposed approach, we introduce a graph centrality measure called BACRank, which estimates

the final sensitivity of each component. This chapter is based on the contents of a paper published at a refereed conference [4].

Chapter 6 – An Extended Multi-Context Evaluation of BACRank We present a smart building testbed that implements general purpose building services such as illumination, ventilation, and temperature control. Our testbed is designed to easily adapt these services to emulate the environmental requirements of real-world locations (e.g., hospitals, airports, data centers, etc.). This feature allows us to assess the sensitivity of smart building components from multiple business contexts. Specifically, the metric used for our experiments is founded on *BACRank* (described in Chapter 5). This chapter is based on the contents of a paper published at a refereed conference [1].

Chapter 2

Background

Cyber-Physical Systems (CPSs) refer to a variety of applications where computer systems interact with physical aspects of the world [84]. This interaction is typically aimed at controlling physical variables such as speed, temperature, and pressure, which has proved crucial in many applications. There is a wide range of systems that fall under this definition of CPSs; for instance, smart buildings, Industrial Control Systems (ICSs), medical devices, and many more. In all these domains, CPSs perform the control of physical variables in a reliable, flexible, and efficient way [130].

The control that CPSs exert on the real world is possible thanks to a variety of hardware and software components. Concretely, *sensors* measure physical variables from the environment, which allows the system to identify the current state of physical processes. The inputs from sensors are then processed by *computer systems* in charge of automating the control of physical processes. These computer systems may also vary depending on their particular application, ranging from resource-constrained microcontrollers to robust Programmable Logic Controllers (PLCs). Finally, these computer systems are capable of manipulating physical processes by means of *actuators*. There is also a wide range of actuator devices that are used in diverse applications; for instance, valves, pumps, alarms, lights, and many more.

A taxonomy of CPSs proposed by [46] identifies two main types. First, *infrastructural CPSs* are those in which the goal is to control physical machinery. Another common term used to refer to infrastructural CPSs is *Operational Technology (OT)*, which is typically used to contrast it with traditional IT systems. Two examples of infrastructural CPSs are smart buildings and ICSs. The second type are *personal CPSs*. These refer to computer systems enriched with physical data typically related to specific users. These include smart-watches, pacemakers, oximeters,

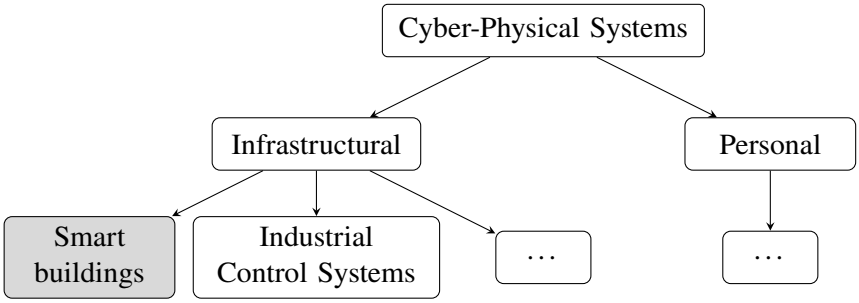


Figure 2.1: CPSs taxonomy proposed in [46]. We highlight in gray color *smart buildings* as the main object of study of this thesis.

and others. Figure 2.1 depicts the taxonomy previously described. The focus of this thesis are infrastructural CPSs and particularly smart buildings. However, some of the proposed approaches can also be applied to the closely related ICSs domain. For this reason, in the remainder of this chapter we discuss in more detail smart buildings in Section 2.1 and ICSs in Section 2.2.

2.1 Smart Buildings

Smart buildings are defined in literature as buildings "equipped with integrated technology systems such as building automation, life safety, telecommunications, user systems, and facility management systems" [137]. The *integration* of these systems is a crucial aspect for a building to be considered *"smart"*. Among the *building automation* services are heating, ventilation, and air conditioning (HVAC), indoor transportation (e.g., elevators, escalators, travelators), illumination, and many more.

The motivations to realize smart buildings have evolved through time. Originally, the main goal was to achieve energy efficiency and comfort for the building occupants [54]. However, nowadays smart buildings help organizations to comply with laws and regulations regarding the environmental conditions in which they must operate [4]. Hospitals, airports, and data centers are examples of critical buildings for society that rely on smart building technologies to comply with the regulations of their corresponding industries (e.g., temperature, ventilation, and illumination requirements). Although energy efficiency and comfort are still desirable features in these locations, the primary goal of these smart buildings is to enable their organization's daily operations.

From a technical perspective, smart buildings are implemented using interconnected specialized devices. According to ISO 16484-1 (Building Automation and Control Systems (BACS) — Part 1: Project specification and implementation), these devices should be organized in a 3-layered architecture as shown in Figure 2.2 [66]. In this architecture the *management* level provides monitoring and control functions to building administrators. The *automation* level is comprised of computerized controllers that execute the logic behind the implemented building services. The controllers receive inputs from the environment through sensors, execute the appropriate logic, and send outputs back to the environment using actuators. Both sensors and actuators are the elements found at the *field* level. Although the components that take part of a smart building are relatively static, it is not uncommon to add, remove, or replace devices in all 3 layers due to diverse reasons; for example, to update obsolete hardware/software or due to the repurposing of buildings or building rooms.

The automation of building services using Internet of Things (IoT) devices is also a common trend [134]. This is, however, a less structured way to implement automated building services. Although this approach has given the general public an affordable access to automated building services, consumer-grade IoT devices are still far from the integrated capabilities of professional-grade smart buildings [54]. The main drawback of IoT-based building services is the use of proprietary software, communication protocols, and cloud services, which hinders their interoperability [40].

The communication between professional-grade smart building components typically leverages well-known protocols in the lower layers (e.g., TCP/IP) and smart building specific protocols in the upper layers. Among the most popular smart building protocols are LonTalk, KNX, and BACnet. Although the approaches proposed in this thesis are mostly protocol independent, our evaluations are often based on real-world systems implemented using the BACnet protocol. For this reason, we elaborate on the particular features of the BACnet protocol in Section 2.1.1.

Security research on smart building systems has focused on their network communications. Researchers have explored network traffic normalizers, domain-specific firewalls, and (mostly) intrusion detection systems [29, 60, 76, 105, 132]. The main smart building communication protocols are standardized, which makes network communications research generalizable across a wide spectrum of devices and manufacturers. On the other hand, smart building applications have not been analyzed from a security perspective. The main reason for this is likely the lack of standardized programming languages to program smart building controllers.

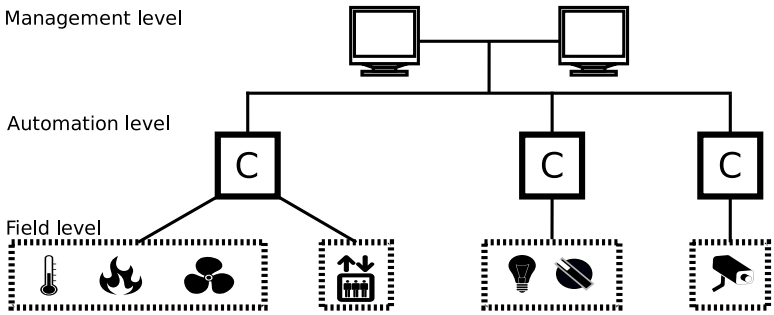


Figure 2.2: Typical 3-layered architecture of smart buildings.

2.1.1 BACnet Protocol Overview

Building automation and control networks, better known as BACnet, is a vendor-neutral communications protocol standardized by the International Organization for Standardization (ISO 16484-5), the American National Standards Institute and the American Society of Heating, Refrigerating and Air-Conditioning Engineers (ANSI/ASHRAE standard 135) [15]. BACnet dictates a set of rules that govern how the devices controlling smart buildings must communicate. Thanks to its standardization, diverse BACnet devices can interoperate regardless of their manufacturer. BACnet's popularity is reflected in more than 1 300 registered manufacturers of BACnet devices worldwide.[1]

The BACnet protocol stores information in predefined variables called *BACnet properties*, which are in turn, encapsulated in predefined data structures called *BACnet objects*. The BACnet standard defines 60 object types to meet the most frequent needs in smart buildings. For each of these objects, the standard dictates which properties are optional, required, and writable. Moreover, the standard allows the possibility to implement proprietary (vendor specific) objects and properties in BACnet devices. There is only one mandatory object for all BACnet devices: the *device* object. This object contains several device-describing properties such as vendor-name, model-name, and firmware-revision. Other common objects are, e.g., analog-input, analog-output, binary-input, and binary-output. The input objects are typically used to store information coming from sensors (e.g., temperature, switches, etc.), whereas output objects are used to control actuators (e.g., valves, fans, etc.). Since the applicability of standard objects vary depending on the purpose of the device, they are not required to implement all standard objects. Additionally, it is important to distinguish between object types and object instances. Object

[1] `http://www.bacnet.org/VendorID/BACnet%20Vendor%20IDs.htm`

types are templates that will be filled in with data when instances are created. BACnet devices that claim support of an object type must be able to operate on instances of it.

The communication between BACnet devices is possible thanks to predefined BACnet *services*. BACnet services refer to specific requests that devices can handle. *Read Property*, *Write Property*, *Reinitialize Device*, and *Atomic Write File* are some examples of BACnet services. Depending on the device's purpose, only a subset of them has to be implemented. Services typically involve two roles, clients that send requests, and servers that reply. In order to clarify which services can be sent or received by different devices, the BACnet standard defines the BACnet interoperability building blocks (BIBBs) [101].

BIBBs are a convenient way to know whether two devices are capable of communicating, i.e., to identify if one of them can make a particular request and the other one can reply to it. BIBBs are often mentioned as acronyms with 3 components: *type*, *task*, and *capability*. First, the *type* refers to 5 broad BIBB categories: data sharing (DS), alarm and event management (AE), scheduling (SCHED), trending (T), device management/network management (DM/NM). Second, the *task* specifies the purpose of the BIBB: e.g., read property (RP), write property (WP), notification (N). Lastly, the *capability* states whether the device acts as a client or a server, denoted as "A" or "B", respectively. For example, the BIBB *DS-RP-A* stands for data sharing (type), read property (task), and client role ("A" capability). Devices implementing DS-RP-A can send Read Property queries to other devices. A device with the complementary BIBB, namely DS-RP-B, would be able to reply to such a query. According to the standard, a particular BIBB demands the implementation of a subset of services (one or more). BACnet devices implement those BIBBs required by the standard according to their *device profile*.

Device profiles categorize BACnet devices based on their functionality. Common examples of BACnet profiles are B-OWS (BACnet Operator Workstation), B-BC (BACnet Building Controller), B-ASC (BACnet Application Specific Controller), B-SS (BACnet Smart Sensor), and B-SA (BACnet Smart Actuator). According to the standard, a particular device profile demands the implementation of a subset of BIBBs. Moreover, it is worth noting that BACnet devices might undergo a certification process intended to ensure their abidance to the standard and claimed profile. This process is done by independent certification laboratories that perform standardized tests [67].

From a security perspective, the BACnet protocol incorporated optional mechanisms to encrypt and authenticate network messages. However, few manufacturers actually implemented this features [101, 137]. Some of the problems that caused a lack of support to this initiative were weak symmetric keys (56-bit Data Encryption

Standard), a broken authentication protocol, and unspecified key distribution methods [59]. More recently, in 2 019, this part of the standard was deprecated in favor of a new approach called BACnet Secure Connect (BACnet/SC) [48]. BACnet/SC incorporates IT security best practices in regard to key sizes and algorithms, and has been well received by the industry.

2.2 Industrial Control Systems

Industrial Control Systems (ICSs) are a type of infrastructural CPS used in industrial processes to achieve a common goal, such as creating a product or delivering a service [7]. Concrete examples of ICSs include power plants, water distribution plants, and manufacturing systems (e.g., cars, electronics, food). These systems execute tasks often deemed as crucial for society. For this reason, they are commonly regarded as *critical infrastructures*.

ICSs are typically built using a 3-layered architecture similar to the smart buildings architecture [57]. At the *management* level, ICSs make use of Supervisory Control and Data Acquisition (SCADA) software that allows operators to monitor the physical processes and react to alert messages. Operators can also use SCADA software to make changes on the physical processes in real time (e.g., direct activation of actuators, modification of operational setpoints, etc.). The *automation* level is typically comprised of Programmable Logic Controllers (PLCs) and Remote Terminal Units (RTUs). These computing devices are built to satisfy strict requirements such as e.g., extended uptimes, real-time responsiveness, and harsh environmental conditions. Finally, at the *field* level, there are sensors and actuators that interact with the real world to implement the desired physical processes. The hardware and software in these 3 layers is rarely replaced. For this reason, ICS networks are often considered static environments [57]. These 3 layers are part of a larger architectural model commonly used for ICSs that aims to connect IT and OT networks, known as the Purdue Reference Model [139].

At the automation layer, PLCs are programmed using standardized languages defined in standard IEC 61131-3 [62]. The latest version of the standard (2 013) describes one textual and three graphical languages. The textual language is called *structured text* and the graphical languages are *ladder diagrams*, *function block diagrams*, and *sequential function charts*. All major PLC manufacturers have implemented development environments that support these standardized languages. These development environments compile the customized programs into binary code that is later on downloaded to the PLCs.

ICSs are implemented using specific communication protocols such as Eth-

erNet/IP, Common Industrial Protocol (CIP), and Modbus. These protocols have similar design features as the BACnet protocol; for instance, a client-server model, service-oriented requests, and an object-based information exchange method. Unlike the BACnet protocol, whose main application domain are smart buildings, these are general purpose protocols suitable for diverse industrial applications.

The security issues of ICSs have been thoroughly studied in both academia and industry [54]. The interest on this domain increased even more after the Stuxnet attack became publicly known [142]. Besides network-based security solutions such as firewalls and intrusion detection systems, ICS software has also been studied leveraging the standardized programming languages for PLCs [28]. Due to the potential impact of ICS failures, there has been a particular interest in formally proving the correctness of the software running this kind of systems [39].

2.3 Summary

Cyber-Physical Systems (CPSs) are computer systems (*cyber*) that interact with the real world (*physical*) through specialized hardware. The physical-interaction hardware are *sensors* and *actuators* that allow the computing devices to receive inputs from the environment and send outputs back to it, respectively. The computers used to implement CPSs are often embedded devices such as microcontrollers, Programmable Logic Controllers (PLCs), and building controllers. A simple taxonomy proposed in literature splits CPSs in two broad categories: *infrastructural* and *personal* CPSs. Two concrete examples of infrastructural CPSs are smart buildings and Industrial Control Systems (ICSs). Personal CPSs include smart-watches, medical devices, and others. The focus of this thesis are infrastructural CPSs and *smart buildings* in particular. However, some of the proposed approaches are also applicable to ICSs; thus, our emphasis on these two kinds of systems.

Smart buildings and ICSs have similarities often shared among infrastructural CPSs. For instance:

- **Architecture:** both kinds of systems are typically implemented using a 3-layered architecture comprised of *management*, *automation*, and *field* layers.
- **Management tools:** both kinds of systems use of Supervisory Control and Data Acquisition (SCADA) software, human-machine interfaces (HMIs), and operator work stations (OWSs) at the management level.
- **Control algorithms:** control algorithms traditionally used in ICSs can also be used for smart buildings. For instance, the same basic principles to reach

a specific temperature in an ICS reactor apply to control the temperature in a building room.

However, there are also some differences that are worth mentioning:

- **Application layer protocols:** while ICSs typically use general purpose industrial protocols such as Modbus, smart buildings are usually implemented using specialized protocols like BACnet. These specialized protocols have high level abstractions that ease the implementation of building services.
- **Network dynamics:** smart building networks change more often than the typical ICS network. This happens because buildings are more dynamic environments than industrial facilities. Whereas ICSs often remain unchanged after deployment, buildings are regularly modified (e.g., rooms are merged, split, change purpose). These changes are reflected in the smart building system, particularly in the field and automation levels, where devices must be added, removed, or repurposed.

Regarding the security of both, ICSs and smart buildings, there are also similarities and differences. Although most attack types affecting ICSs can be adapted to the smart buildings context, only the defenses that are based on their similarities (e.g., architecture, management, control algorithms) can be transferred as well with reasonable effort. The ICS defenses that are based on, e.g., application layer protocols or network dynamics, cannot be easily adapted to the smart buildings context. In more general terms, this is also the case of IT security controls, that can only be adapted to the ICS and smart building domains on their overlapping features (e.g., TCP/IP firewalls, antivirus software in management workstations, OS log analysis).

Chapter 3

Weakness Identification in Smart Building Applications

As discussed in Chapter 1, smart building applications, typically developed in-house, might contain weaknesses that often remain unnoticed. This poses a serious problem because smart building applications directly control the building services offered to the organization hosted in the building. The direct link between smart building applications and physical processes, enables stealthy and service-oriented cyber-physical attacks. These attacks are much harder to detect than attacks caused by, e.g., network flooding or leveraging well-known CVEs.

In this chapter, we address **Research Question 1**: How to (semi-) automate the identification of weaknesses in smart building applications?

Our proposed approach to automatically identify weaknesses in smart building applications focuses on a popular programming pattern known as *closed control loop (CCL)* [35]. The goal of CCLs is to keep physical variables within appropriate operational limits, by means of two steps continuously repeated: (1) computing the difference between a setpoint and current sensor readings; and (2) trying to reduce such a difference using the physical capabilities of an actuator. The specific CCLs configuration is tailored by local engineers to meet specific goals. Beyond smart buildings, this basic programming pattern is also used in other sophisticated Cyber-Physical Systems (CPSs) such as Industrial Control Systems (ICSs), medical devices, satellites, and many more [38, 56, 83, 126]. Since our approach is based on a programming pattern rather than a specific system type, it can be applied to diverse CPSs, as we will show in Section 3.4.1 with a smart building use case and Section 3.4.2 with an ICS use case.

Our proposed approach creates a graph data structure built from the composition

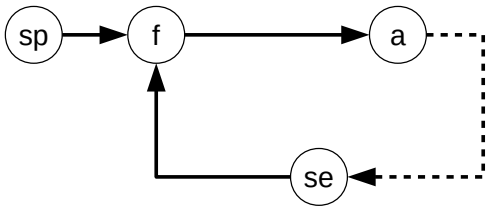


Figure 3.1: A CCL comprised of a setpoint (sp), a sensor (se), a control function (f), and an actuator (a). Solid lines represent communication in the *cyber* domain whereas the dashed line represents interaction in the *physical* realm.

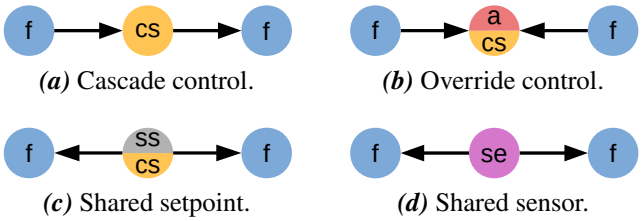
of multiple CCLs that might share components such as sensors, setpoints, and actuators. This graph provides a unified representation of the smart building application. This graph-based model allows us to automatically identify patterns that reveal weaknesses such as, but not limited to, *global variables* and *multi-purpose variables*.

3.1 Preliminaries

Closed control loops (CCLs) constitute a basic programming pattern for smart buildings and other CPSs. A CCL is comprised of four basic components, namely, a setpoint, a sensor, a control function, and an actuator (e.g., valve, heater, light, etc.). The CCL's control function receives inputs from the environment through sensors, compares them with pre-established setpoints, and reacts with a compensatory action intended to minimize the difference. The compensatory action is executed by an actuator in order to control the physical variable measured by the sensor (e.g., pressure, temperature, illumination, etc.). Figure 3.1 depicts the simplest form of a CCL.

The control function is a software component typically running in high availability embedded systems such as building controllers or Programmable Logic Controllers (PLCs). Control functions implement the control logic; for example, arithmetic operations, rate limiters, and other kinds of data processing functions. From a function's perspective, hardware devices such as sensors and actuators are abstracted simply as variables to be read and written. It is worth noting that in distributed control systems these variables might not necessarily reside in the same controller. Therefore, communication to and from the control function might require network transmissions.

Advanced CCL configurations are often needed, e.g., to cope with system disturbances. One of such configurations is called *cascade control*, where one control



(a) Cascade control. ***(b)*** Override control.

(c) Shared setpoint. ***(d)*** Shared sensor.

Figure 3.2: Advanced CCL configurations. Color code: gray: static setpoint (ss), blue: controlling function (f), red: actuator (a), purple: sensor (se), and yellow: calculated setpoint (cs).

function adjusts the setpoint of another control function [135]. This dynamically computed setpoint is called a *calculated setpoint*. In contraposition, we denote user-defined setpoints as *static setpoints*. Although we typically make an explicit distinction between static setpoints and calculated setpoints, in what follows, we use the word *setpoint* to refer to either of them when such a distinction is irrelevant. Graphically, an example of cascade control is shown in Figure 3.2a.

Another advanced CCL configuration is called *override control*. In this setting, one control function manipulates one variable during normal operation, however, a second control function can take over during abnormal operation to prevent some safety, process, or equipment limit from being exceeded [135]. The variable under control by two or more control functions is typically used to manipulate one actuator or calculated setpoint. Figure 3.2b shows a graphical representation of the override control configuration.

It is also common to find setpoints and sensors shared between two or more control functions, as shown in Figure 3.2c and Figure 3.2d, respectively. Shared setpoints provide a convenient centralized configuration for multiple control functions at once. The motivation behind shared sensors is similar to that of shared setpoints, which in addition reflects the fact that, typically, there are limited instances of physical sensors in the system under control. An arbitrary number of the CCL configurations shown in Figure 3.2 can be used to control CPSs [122].

3.2 Proposed Approach to Identify Application Weaknesses

We now present the proposed approach to identify weaknesses in CCL graphs and illustrate it with a motivating example.

3.2.1 Identifying Weaknesses in CCL Graphs

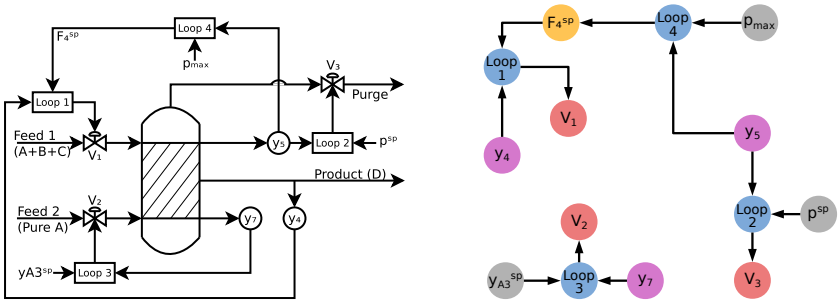
Cyber-physical system (CPS) applications are modeled as graph data structures that abstract the configuration of their CCLs. The graph abstraction of a single CPS might consist of multiple subgraphs. Making an explicit distinction between setpoint types, there are 5 types of nodes in our graphs that match the CCL's components: *static setpoints*, *calculated setpoints*, *sensors*, *control functions*, and *actuators*.

Formally, a CPS is modeled as a directed graph $G(V, E)$ where V is a nonempty set of vertices (or nodes) and E is a set of edges. Every edge has exactly two vertices in V as endpoints. The direction of every edge $e \in E$ models the way information flows in the graph. There are 5 partitions SS, CS, SE, F, and A in V, that segregate the 5 types of nodes (*static setpoint*, *calculated setpoint*, *sensor*, *control function*, and *actuator*, respectively), such that $SS \cup CS \cup SE \cup F \cup A = V$. It is worth noting that although we assume knowledge about the type of the nodes (e.g., sensor), we do not require further details about them (e.g., temperature sensor, pressure sensor, etc.).

There are different options to create CCL graphs. For instance, it is possible to extract the CCL graph of CPSs from piping and instrumentation diagrams (P&IDs). These diagrams show interconnected physical instruments and usually contain information about the CCLs used to control CPSs. Figure 3.3 shows an example of a P&ID in Figure 3.3a, and its corresponding CCL graph abstraction in Figure 3.3b. The creation of a CCL graph from a P&ID could even be automated using diagram digitization techniques such as [20]. Moreover, there are other options to create CCL graphs, such as by analyzing the network traffic of smart buildings implemented using the BACnet protocol [3]. This method is used in the analysis of a real BACnet system in Section 3.4.1.

Pattern Matching

We propose to use the accumulated knowledge about software weaknesses in the IT domain and transfer it to the CPSs domain. In particular, we leverage Mitre's Common Weakness Enumeration (CWE) database, which documents common IT weaknesses. We translate CWE weaknesses into specific node patterns to be searched for in CCL graphs. While we use additional CWE weaknesses in our implementation, here we focus on two entries which proved particularly useful experimentally. Our goal is to provide an intuitive comprehension of the proposed approach without limiting its applicability to a specific subset of weaknesses. More formally, we identify each pattern P_i with an index $i = 1, \ldots, n$. Each pattern



(a) P&ID of a chemical plant.

(b) Graph abstraction of the control system derived from the P&ID.

Figure 3.3: Piping and instrumentation diagram (P&ID) of an ICS and its corresponding graph abstraction.

matching query on the graph returns a subset $S_i \subseteq V$ of matching nodes using pattern P_i. It is important to highlight that those weaknesses identified by more patterns should be prioritized in the final result set T.

CWE-1108: Excessive Reliance on Global Variables. *"The code is structured in a way that relies too much on using or setting global variables throughout various points in the code, instead of preserving the associated information in a narrower, more local context."* [96].

Global variables are generally considered a bad software engineering practice. Their main disadvantage is that malicious or benign-but-buggy changes to them will propagate and possibly disrupt many parts of the code. Global variables can be observed in CCL graphs mainly due to shared setpoints and/or sensors. As explained in Section 3.1, these are typical ways to combine CCLs in CPSs (see, e.g., Figure 3.2c and Figure 3.2d).

A suitable algorithm to identify global variables in CCL graphs is the *out-degree* centrality. This algorithm assigns a score to each node in a graph by counting their number of outgoing edges. More formally, for every node $v \in V \setminus F$, the out-degree of v is denoted as $d^+(v)$. We explicitly disregard *function* nodes in set F since this particular weakness is exclusively about variables. We select as potential targets those nodes whose value $d^+(v) \geq \tau$, for a context dependent threshold τ.

CWE-1109: Use of Same Variable for Multiple Purposes. *"The code contains a callable, block, or other code element in which the same variable is used to*

control more than one unique task or store more than one instance of data." [96].

Overloading a variable with multiple responsibilities might unnecessarily increase the complexity of the code around it. Such complexity becomes an indirect security issue since it obscures the readability of the code.

In control engineering, the usage of *override controllers* deliberately creates a pattern in which two or more control functions manipulate a single variable (see Figure 3.2b). The manipulated variable is, commonly, of type *actuator* or *calculated setpoint*. Due to the widespread implementation of override controllers in CPSs, it is common to find this pattern in real CCL graphs.

The automated identification of override controllers in CCL graphs can be done by means of the in-degree centrality algorithm. This algorithm assigns a score to each node in a graph by counting their number of incoming edges. Our implementation computes the in-degree centrality for all nodes $v \in V \setminus F$. As in the previous case, we explicitly disregard *function* nodes in set F since this particular weakness is exclusively about variables. We denote the number of incoming edges of a node as $d^-(v)$. Thus, we look at nodes whose $d^-(v) \geq \tau$, where typically $\tau = 2$.

3.2.2 Motivating Example

Figure 3.3a shows the model of a chemical plant originally described in [112]. Four chemical components referred to as A, B, C, and D, are part of the process. The first three components are combined in the reactor to create the final product D. The goal of the application controlling the plant is to keep a stable and high-quality production rate while minimizing the waste of raw material. There are four control functions that take as input the values coming from three sensors and four setpoints (1 calculated setpoint and 3 static setpoints). The outputs of the functions aim to control the three valves in the plant.

Figure 3.3b depicts the CCL graph of the same chemical plant, which can be easily derived from its piping and instrumentation diagram (P&ID). This graph shows two kinds of CCL combinations: (1) a shared sensor y_5 between control functions *Loop 4* and *Loop 2*; and (2) a cascade control in which control function *Loop 4* sets a calculated setpoint $F_{4^{sp}}$ to control function *Loop 1*.

The proposed approach looks for weakness patterns in the chemical plant's CCL graph that could identify suitable sensors to target. The search for multi-purpose variables (functions excluded) aims at nodes whose in-degree is greater or equal than two; no results are produced in this case. The search for global variables (functions excluded) looks for nodes whose out-degree is greater than a predefined threshold τ. All setpoints have an out-degree of one which makes them unfit to

be labeled as global variables. However, the three sensors y_4, y_5, and y_7 have an out-degree of 1, 2, and 1, respectively. This small sensors sample shows an average out-degree of 1.33 with standard deviation of 0.58. Defining τ as the sum of the average and one standard deviation ($\tau = 1.91$) sets node y_5 as a potential target according to the proposed approach. Our result set would have sensor y_5 as an ideal candidate to target in this particular infrastructure. This result is concordant with previous works which state that "[i]n general we found that the plant is very resilient to attacks on y_7 and y_4... If the plant operator only has enough budget to deploy advanced security mechanisms for one sensor (e.g., tamper resistance, or TPM chips), y_5 should be the priority" [26]. It is worth noting that previous works reach to the same conclusion but leveraging a detailed simulation of the plant and using it to perform several experiments in advance. Besides, such a detailed simulation is typically unavailable for real CPSs. On the other hand, our approach requires little knowledge about the system (only an abstract CCL graph) and can be fully automated, from the creation of the graph, to the final identification of weaknesses.

3.3 Implementation of the Proposed Weakness Identification Approach

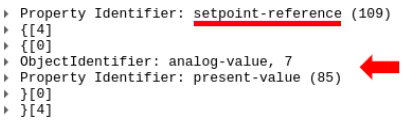
We implement the proposed approach on top of Neo4j version 4.1.2 [100]. Neo4j is a noSQL database engine specialized in graph data structures. Neo4j offers a natural way to store CCL graphs and a high level query language—called *Cypher Query Language*—that allows to perform complex queries in just a few lines of code.

This section describes the automated creation and storage of CCL graphs from smart building systems and the queries needed to pinpoint weaknesses in these graphs.

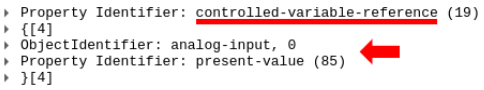
3.3.1 Automated CCL Graph Creation

In this section, we introduce the most important implementation details of a tool we built to create generic BACnet object graphs [15]. We call our tool BACGRAPH and its source code is freely available under the GNU/GPLv3 license.[1] However, here we focus on a subset of the tool's capabilities that allow the creation of CCL graphs. As we will show, it is possible to fully automate the creation of CCL graphs in smart buildings that make use of the BACnet protocol.

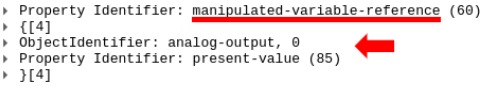
[1] `https://gitlab.com/bacgraph1/bacgraph`.



(a) Setpoint-reference property.



(b) Controlled-variable-reference property.



(c) Manipulated-variable-reference property.

Figure 3.4: Wireshark display of Loop object properties.

BACGRAPH takes as input network traffic samples from the infrastructure under study. All observed BACnet objects that take part of closed control loops become nodes of the CCL graph. Additionally, BACGRAPH looks for specific properties in some BACnet objects, which might contain references to other object instances. These references are used to create the edges of the graph. A deterministic finite-state machine (FSM) is implemented within BACGRAPH to keep track of the *current* object instance (source of the edge) and the *referenced* object instance (destination of the edge), if any. An excerpt of the implemented FSM is shown in Figure 3.5. Although BACGRAPH handles many different object types (and their corresponding references) to create complete graphs of BACnet systems, Figure 3.5 shows only those objects related to the CCL programming pattern.

A walk through the FSM starts with every network packet. If a packet has data to be analyzed, then the FSM transitions to the *(R)ead* state. Once in the *R* state, the software reads the packet in sequential order until it finds a BACnet object instance. If a (L)oop object is found, then the FSM transitions to the corresponding state and the software looks for 3 specific properties (shown in Figure 3.4):

1. a *setpoint-reference* (sr) that points to an object that represents a setpoint and causes a transition to L_1 (see Figure 3.4a);

2. a *controlled-variable-reference* (cvr) that points to an object that represents a sensor and causes a transition to L_2 (see Figure 3.4b); and

3. a *manipulated-variable-reference* (mvr) that points to an object that represents a actuator and causes a transition to L_3 (see Figure 3.4c).

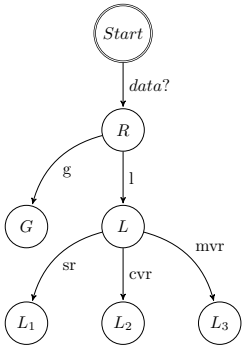


Figure 3.5: Simplified illustration of the finite-state machine (FSM) implemented by BACGRAPH. It contains the most important states and transitions to identify BACnet objects and references that take part of CCLs.

In addition to the *Loop* object, we pay particular attention to *Analog-Input*, *Analog-Output*, *Analog-Value*, *Binary-Input*, *Binary-Output*, and *Binary-Value* objects, which are typically used to represent the setpoints, sensors, and actuators, as it is also shown in Figure 3.4. Due to the similar characteristics of these 6 objects, we treat them in the same (G)eneric way in the FSM.

For the sake of readability, Figure 3.5 omits states that look for additional BACnet objects and returning transitions from each state to previous states.

3.3.2 Automated Weakness Identification

Our implementation includes a *pre-processing* stage that pre-computes information required in the *pattern matching* phase. Since both weakness-related patterns discussed in Section 3.2.1 are based on the in- and out-degree centrality algorithms, the *pre-processing* stage consists of queries that assign the in- and out-degree to all nodes in the graph that are not of type *function*. Both queries are shown in code listing 3.1.

The *pattern matching* phase consists in finding nodes that satisfy a specific condition in the graph's structure. These particular conditions pinpoint weaknesses in the application modeled by the graph. For the sake of brevity, here we discuss two types of weaknesses, namely, global variables and multi-purpose variables. Global variables are nodes (function nodes excluded) whose out-degree is greater than a context dependent threshold τ. In our implementation, we define τ as the average plus one standard deviation of the out-degree of nodes segregated per type. Code listing 3.2 shows the query for the global variable weakness pattern.

The second *pattern matching* query looks for multi-purpose variables. Multi-

purpose variables are nodes (function nodes excluded) whose in-degree is greater or equal than a context dependent threshold τ, where usually $\tau = 2$. This is, indeed, the threshold that we have used in our implementation. Code listing 3.3 shows the query used to find the multi-purpose variable weakness pattern.

Lastly, since sensors have been identified by previous works as ideal targets to compromise CPSs [61, 80], we added to our implementation an optional *post-processing* phase so that the final result set T is comprised exclusively of sensors. In essence, the *post-processing* phase replaces non-sensor nodes found during the *pattern matching* phase, with sensor nodes that have a path to them, thus, influencing their behavior. This ensures that the final list of weaknesses is comprised exclusively of sensor nodes. Code listing 3.4 shows the *post-processing* query. This query looks for all sensor nodes (src) that have a path to the node of interest (dest), regardless of the length of the path. Figure 3.6 depicts the three phases implemented to identify weaknesses in CPSs.

```
MATCH (n)
WHERE NOT n:FUNCTION_TYPE
SET n.indegree = size((n)<-[]-());

MATCH (n)
WHERE NOT n:FUNCTION_TYPE
SET n.outdegree = size((n)-[]->());
```

Listing 3.1: Pre-processing stage: setting the in- and out-degree centrality to each node.

```
MATCH (n:NODE_TYPE)
WITH AVG(n.outdegree) as average,
     stDev(n.outdegree) as stdev
  MATCH(m:NODE_TYPE)
  WHERE m.outdegree > (average+stdev)
  RETURN m;
```

Listing 3.2: Global variable pattern matching query.

```
MATCH(n:NODE_TYPE)
WHERE n.indegree >= 2
RETURN n;
```

Listing 3.3: Multi-purpose variable pattern matching query.

```
MATCH (src:Sensor), (dest {id:NODE_ID})
WHERE (src)-[*]->(dest)
RETURN src;
```

Listing 3.4: Query that finds the path from sensor nodes to non-sensor nodes originally identified as weaknesses.

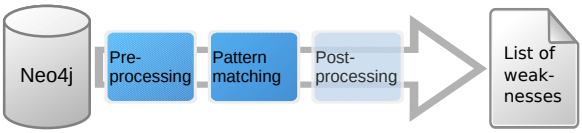


Figure 3.6: Implementation of the proposed approach. The post-processing stage is optional in this workflow.

3.4 Evaluation of the Proposed Weakness Identification Approach

In this section, we apply the proposed approach on two different CPSs. The first scenario is a real smart building application spread over 23 buildings located in the campus of the University of Twente and its surroundings. These buildings are used by over 10 000 students and 3 000 staff members. The services automated in these buildings include illumination, heating, ventilation, and cooling.

The second scenario is a simulated Industrial Control System known as the Tennessee Eastman Plant (TEP). This is a realistic chemical plant for which we analyze two independently developed control applications. Being a realistic system for which there are simulations readily available, the TEP has been extensively used by previous cybersecurity research as well [26, 61, 80, 81, 120]. All simulations are executed on the MATLAB/Simulink environment. Specifically, we use MATLAB version R2015a running on Windows 10.

3.4.1 Smart Building Application

We start by setting up a mirroring port in one of the core network switches of the University to collect BACnet traffic. The strategic location of this switch allows us to observe all *inter*-building communication. For our analysis, we use 7 weeks of BACnet traffic, which accounts for 60,3 GB of data. Leveraging this dataset, we automatically created the CCL graph using the tool described in Section 3.3.1. In total, the automatically generated graph is comprised of 4 771 nodes and 4 425

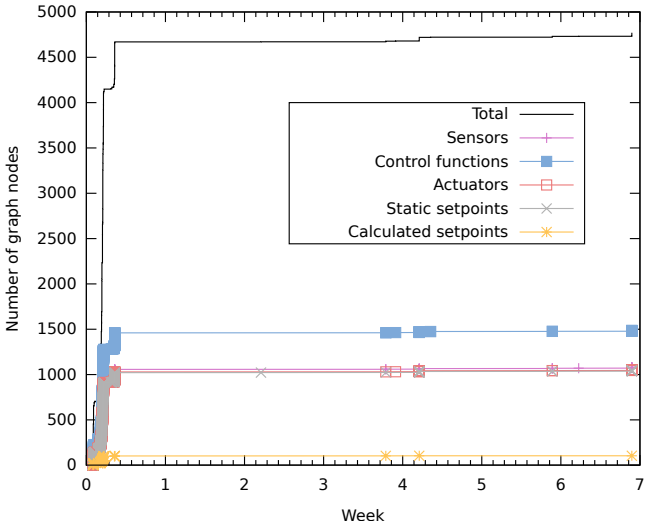


Figure 3.7: Node discovery rate during a 7-week long analysis of real BACnet traffic.

edges. The graph contains 1 487 control function nodes (*Loop* objects), 1 079 sensors, 1 047 setpoints, 104 calculated setpoints, and 1 054 actuators. The graph spreads throughout 216 BACnet devices. Figure 3.7 shows the number of CCL components discovered over a 7 week period during a totally passive capture. It is worth noting that the vast majority of the nodes were discovered in only 3 days. Finally, Figure 3.8 shows the distribution of sensors, actuators, static setpoints, and calculated setpoints that take part of two or more CCLs.

Although due to confidentiality concerns we cannot reveal specific details of the analyzed smart buildings, we show here a summary of our findings on the automatically generated CCL graph.

We start our analysis by identifying potential global variables in the system. We do so computing the degree centrality in the entire graph. Looking specifically at the out-degree for each node, we identified a sensor that is used by 8 different control functions. Such sensor has the greatest out-degree in the infrastructure. Considering that, on average, sensors provide their readings to 1.38 control functions, this particular sensor stands out as a potential weakness.

While the average out-degree among the setpoints is 1,24, there is one setpoint used by 14 different control functions. This is the setpoint with the greatest out-degree in the BAS. The subgraph where it is located is shown in Figure 3.9a. Such setpoint is considered a global variable that, if targeted by an attacker, is likely to disrupt several subsystems simultaneously.

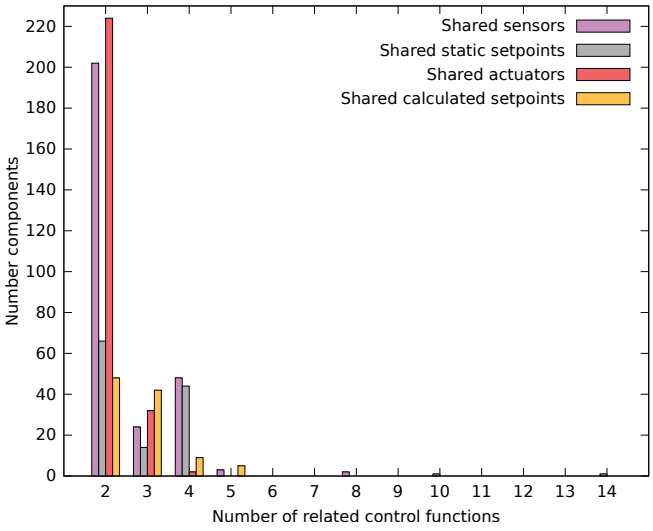


Figure 3.8: Distribution of sensors, actuators, static setpoints, and calculated setpoints that take part of two or more CCLs

The global variable in Figure 3.9a is a setpoint that defines the desired air quality in one of the buildings. The air quality is measured as CO_2 parts per million. The inner ring of control functions considers such setpoint and compares it with sensor measurements from different locations inside the building. The outer ring of control functions is in charge of the temperature control of the building. The shared actuators in between both rings replace stale air from inside with fresh air from the outside. Figure 3.9a shows the interplay between two optimization objectives: air quality and temperature control. An attack on the air quality setpoint is likely to make the building unavailable to its users given the strict regulations on building's air quality, and particularly, under the current COVID-19 pandemic conditions [16, 42].

Our search for multi-purpose variables showed that there are 224 actuator instances managed by 2 control functions, 32 actuator instances managed by 3 control functions, and 2 actuator instances managed by 4 control functions (see Figure 3.8). One of those two actuators controlled by 4 different functions is shown in Figure 3.9b. We did not find calculated setpoints being written by more than one control function. Although only 258 actuator instances are commanded by 2 or more control functions, it highlights the complexity of the control strategy in the smart buildings under study. This might signal potential attackers about the actuators' capacity to disrupt the system, either by a direct attack or by targeting

other nodes in their vicinity (e.g., sensors that exert influence over these actuators).

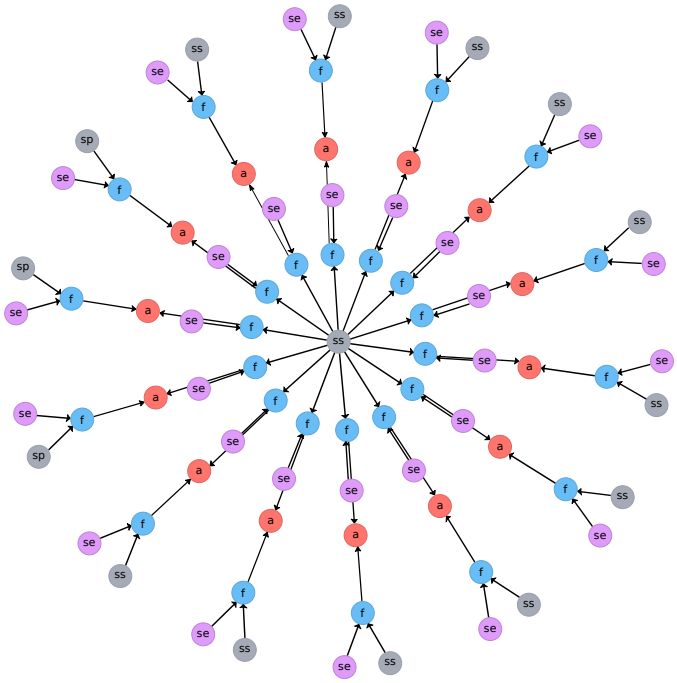
The subgraph shown in Figure 3.9b depicts a single actuator that takes air from the outside and distributes it to 4 different ventilation zones in the building. The fact that all ventilation zones rely on a single air intake device makes it an ideal target to compromise the availability of the building—as in the previous case—due to the potential degradation of the air quality.

Although we found nodes in the smart buildings' graph that match our definitions of weaknesses, it is difficult to assess their actual criticality (from a security perspective) unless experimental attacks are executed. Since the analyzed environment is a real smart buildings network, it was not possible to execute attacks against it. Nevertheless, we show the feasibility of our approach in finding potentially serious weaknesses in real smart buildings. To be able to execute attacks against the weaknesses found by our approach, which we hypothesize are better offensive targets than other components, we turn to simulated environments.

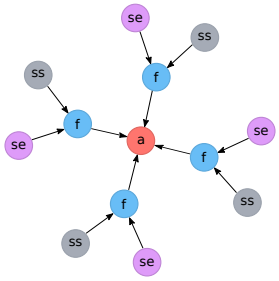
3.4.2 Tennessee Eastman Plant

The seminal 1 993 paper by Downs and Vogel describes an Industrial Control System known as the Tennessee Eastman Plant (TEP) [41]. This is a realistic chemical plant that performs two gas–liquid exothermic reactions. Their description includes, among other details, the expected input and output of the plant, each step of the process from start to end, and the hardware available to control the process. The control hardware includes 41 sensors and 12 actuators, depicted in Figure 3.10. Their paper describes 20 disturbances commonly found in real chemical plants. For instance, sticky valves, changes in chemical reaction kinetics, and random variations in the composition of input streams, each of them with a unique numeric identifier in the range [1–20]. Finally, there are process operating constraints (e.g., maximum reactor temperature, maximum reactor pressure, etc.) that must be satisfied at all times or the plant shuts down. Downs and Vogel present the challenge of implementing a control application for the TEP. To ease engagement in the challenge, the authors provide software to simulate the core components of the TEP such as the sensors, actuators, disturbances, and process operating constraints, leaving space for the missing control application.

Several authors have proposed control applications for the TEP, based on the CCL programming pattern [83, 89, 90, 113]. The main differences between control applications are the robustness against external disturbances, the optimization objectives, and the mechanisms to set the production rate. It is worth noting that the TEP challenge is considered an open-ended problem without a unique correct solution [89].



(a) A static setpoint used by 14 control functions.



(b) An actuator commanded by 4 control functions.

Figure 3.9: Two subgraphs automatically extracted from a real and operational BACnet system.

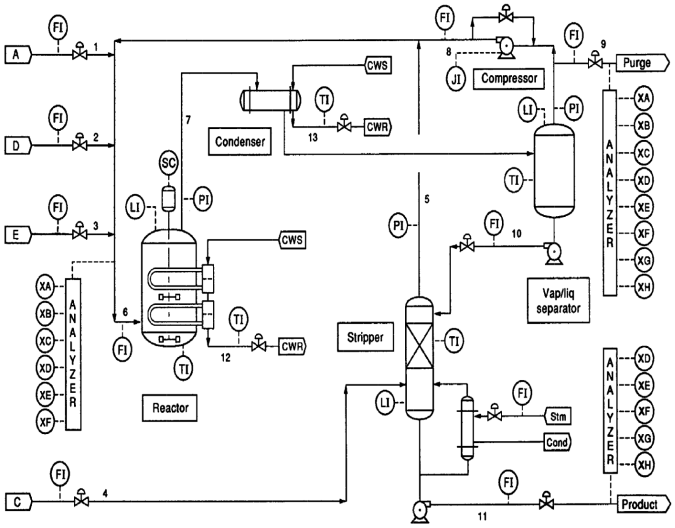


Figure 3.10: Tennessee Eastman Plant diagram taken from the original publication [41].

In this evaluation, we consider two CCL-based control applications for the TEP to investigate whether the weaknesses identified by our approach are, indeed, better offensive targets than other components of the system. The first application, proposed by Larsson et al. [83], is available in the MATLAB/Simulink environment [114][2]. The second application, proposed by Luyben et al., is available in the Fortran programming language [89]. We translated it to the MATLAB/Simulink environment and published it to the research community.[3] For both control applications we proceed as follows:

1. We create the CCL graph and store it in a Neo4j database.

2. We use the proposed approach to identify weaknesses including the pre- and post-processing phases described in Section 3.3.2.

3. We perform data integrity attacks against *all* sensors used in the plant, one at a time, to obtain a ground-truth about each sensor's capacity to cause a plant shutdown. We refer to the time elapsed since the beginning of the attack and until the simulation stops as the shutdown time (SDT).

4. We rank all sensor attacks by their SDT.

[2]Simulation and results available at `https://gitlab.com/eastman_tennessee/larsson`.
[3]Simulation and results available at `https://gitlab.com/eastman_tennessee/luyben`.

5. Finally, we compare the SDT of the automatically identified weaknesses with the SDT of the rest.

In what follows, we refer to an *experiment* as a set of simulated attacks using the same environmental conditions and attack strategy. The *simulations* are configured to run for 72 hours under attack. If a simulation finishes at 72 hours, then we assume that the attack does not cause a shutdown. If the attack causes a violation of the plant's operating constraints, then the simulations stops before the 72 hours. By *environmental conditions* we refer to the disturbances enabled during the simulation. The disturbances considered are those in the range [1–13] and are executed one at a time. According to the original TEP paper, disturbances in the range [14–20] should be used in conjunction with other disturbances [41]. The combinatorial explosion of such constraint deters us from executing simulations with disturbances in the range [14–20]. The *attack strategy* is the way in which the attacker chooses the value used to compromise the integrity of sensors. We use three different attack strategies. First, assuming that the attacker does not have any knowledge about the targeted sensor, we choose the constant 127. This number is small enough to fit in 1 byte (signed int) which ensures that most industrial and smart building protocols will deliver the malicious value in a single packet, thus, executing a stealthy attack. For the second and third attack strategies, we assume that the attacker knows historic sensor readings, in which case we choose the minimum and maximum values observed per sensor, respectively. Although this additional knowledge is not required by our approach, we adopt it as it is a common attack strategy used in previous works.

Control Application #1

The control application by Larsson et al. [83], uses only 9 actuators and 16 sensors out of the 12 actuators and 41 sensors available in the TEP. Additionally, there are 20 control functions, 9 static setpoints, and 12 calculated setpoints. The CCL graph of this control application is comprised of 66 nodes divided in 3 subgraphs. An illustration of the graph is shown in Figure 3.11.

During the weakness identification phase, we run the pattern matching queries that aim at finding global variables and multi-purpose variables. The first query identifies the calculated setpoint number 12 (located in the middle of the largest subgraph) as a *global variable*. Finally, the post-processing phase finds only one sensor that has a path to the global variable: *sensor 17*. Such a path can be visually confirmed following the direction of the edges in Figure 3.11. The second query does not identify *multi-purpose variables* in this control application. Thus, the final target set $T = \{17\}$ contains only one sensor.

We hypothesize that sensor 17 is a weakness in this application. To test our hypothesis we execute 28 experiments comprised of 448 individual simulations that account for 11 047,465 simulated hours (∼1,26 years).

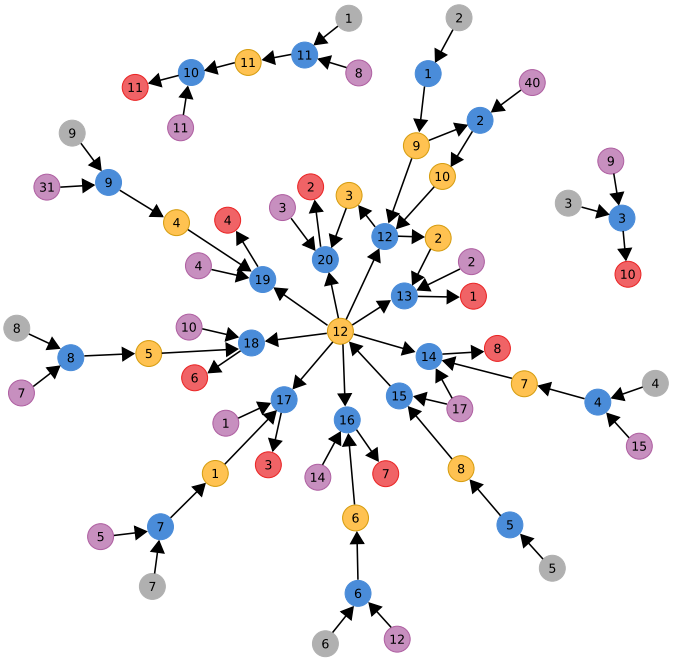
For the first two experiments we use the constant value attack strategy. In one of the experiments we set ideal environmental conditions (no disturbance) and in the other experiment we enable disturbance #8. We choose disturbance #8 because previous works have used exclusively this disturbance for their experiments. The results of the first two experiments are shown in Table 3.1. Regardless of the environmental conditions, the results are consistent in the top half of the table with greater variations in the bottom half. We do confirm that the automatically identified weakness (sensor 17) is the third best target in the infrastructure, with a SDT of 0,19 hours in both experiments (≈ 11 minutes) and a difference of about 5 minutes behind the fastest target (sensor 9 with SDT of 0,10 hours).

For the remaining 26 experiments, we use the minimum and maximum value attack strategies employed by previous works. However, unlike previous works that have used only one environmental condition to execute their experiments, we run simulations under 13 different environmental conditions to gain more confidence in our results. Disturbance #6 is excluded because it is not supported by this particular control application, which means that a shutdown happens even without any attack [83]. Due to space constraints, we summarize our results per attack strategy, which shows the average SDT and standard deviation for each sensor among all the experiments. The results, detailed in Table 3.2 and Table 3.3, show that sensor 17 is the second best target with an average SDT of 1,21 hours and 1,07 hours, respectively. Our results confirm that sensor 17 is a suitable weakness to target not only during specific plant conditions, but in many different situations.

Control Application #2

The second control application, proposed by Luyben et al. [89], uses 10 sensors and 10 actuators from those available in the plant. Additionally, this control application requires 13 static setpoints, 13 control functions, and 1 calculated setpoint. In total, the CCL graph is comprised of 47 nodes. The overall CCL graph of this application, depicted in Figure 3.12, is comprised of 6 subgraphs.

As for the previous control application, we use the queries detailed in Section 4.3 to identify weaknesses, starting with the computation of the in- and out-degree centrality for all the sensors, setpoints, and actuators. The query regarding global variables identifies sensors 8, 12, and 15. The query regarding multi-purpose variables identifies actuators 1, 2, 7, and 11. The post-processing phase looks for sensor nodes that have a path to these actuators and identifies sensors 8, 12, 15, and



Control function	
Node	Function name
1	Rate limiter 1
2	Product %G
3	Reactor temperature
4	Striper level
5	Rate limiter
6	Separator level
7	Recycle rate
8	Reactor pressure
9	%C in purge
10	Separator temperature
11	Reactor level
12	Feedforward
13	D feed rate
14	Stripper liq. rate
15	Production rate
16	Separator liq. rate
17	A feed rate
18	Purge rate
19	C feed rate
20	E feed rate

Setpoint	
Node	Variable name
1	Reactor level
2	Mole % G setpoint
3	Reactor temperature setpoint
4	Stripper level setpoint
5	Production setpoint
6	Separator level setpoint
7	Recycle rate setpoint
8	Reactor pressure setpoint
9	Mole % C setpoint

Actuator	
Node	Variable name
1	xmv 1
2	xmv 2
3	xmv 3
4	xmv 4
6	xmv 6
7	xmv 7
8	xmv 8
10	xmv 10
11	xmv 11

Sensor	
Node	Variable name
1	xmeas 1
2	xmeas 2
3	xmeas 3
4	xmeas 4
5	xmeas 5
7	xmeas 7
8	xmeas 8
9	xmeas 9
10	xmeas 10
11	xmeas 11
12	xmeas 12
14	xmeas 14
15	xmeas 15
17	xmeas 17
31	xmeas 31
40	xmeas 40

Calculated setpoint	
Node	Variable name
1	r1
2	r2
3	r3
4	r4
5	r5
6	r6
7	r7
8	Prodsp
9	PctGsp
10	Eadj
11	SP17
12	Fp

Figure 3.11: CCL graph of control strategy #1. The color of each node represents its type, as described in Figure 3.2. The name of each node corresponds to the variable name in the original MATLAB code.

Table 3.1: Experiments using the constant value attack strategy under two different environmental conditions. The automatically identified sensor 17 is highlighted.

No disturbance		Disturbance 8	
Sensor	SDT (h) ▲	Sensor	SDT (h) ▲
9	0,10	9	0,10
14	0,18	14	0,18
17	0,19	17	0,19
11	0,43	11	0,43
8	0,43	8	0,43
4	0,53	4	0,53
31	0,56	31	0,56
12	0,57	12	0,57
3	1,60	3	1,61
2	1,76	2	1,61
15	2,13	15	2,05
1	7,34	5	5,64
5	7,61	7	6,97
10	8,18	10	7,25
7	8,30	1	9,69
40	72 (no shutdown)	40	72 (no shutdown)

Table 3.2: Summary of experiments considering the minimum value attack strategy under 13 different environmental conditions. The automatically identified sensor 17 is highlighted.

Sensor	Avg. SDT (h) ▲	Std. Deviation
4	0,79	0,67
17	1,21	0,76
9	2,06	0,73
8	2,66	0,41
3	4,11	0,35
2	4,55	0,75
7	7,70	2,19
12	7,97	2,25
14	8,94	2,39
15	11,18	3,96
5	14,53	17,49
31	55,15	27,74
11	66,64	19,31
40	66,69	19,14
10	66,73	19,02
1	66,78	18,82

Table 3.3: Summary of experiments considering the maximum value attack strategy under 13 different environmental conditions. The automatically identified sensor 17 is highlighted.

Sensor	Avg. SDT (h) ▲	Std. Deviation
9	0,57	0,19
17	1,07	0,14
4	1,27	0,21
3	2,60	0,42
8	2,83	0,46
2	3,32	0,66
12	5,44	1,36
14	8,79	2,45
15	11,10	4,36
5	12,64	17,91
31	56,78	27,62
11	66,59	19,50
10	66,64	19,34
40	66,70	19,10
7	66,73	19,00
1	72 (no shutdown)	0,00

29. As described in Section 3.2.1, the final targets subset T contains those sensors linked to more weakness-related patterns. In this particular case, $T = \{8, 12, 15\}$ because these sensors occur in both result sets (global and multi-purpose variable).

We hypothesize that sensors 8, 12, and 15 are suitable weaknesses to target against control application #2. To test our hypotheses we execute 30 experiments comprised of 300 individual simulations that account for 11 079,163 simulated hours (∼1,26 years).

As in the previous control application, we begin with two experiments using the constant value attack strategy under two different environmental conditions. The first experiment without any disturbance and the second experiment under disturbance #8. The results of the first two experiments, detailed in Table 3.4, show no significant differences between both plant conditions. We do confirm that the three automatically chosen weaknesses are in the top half of the table (i.e., fastest targets to cause a shutdown). Particularly, two of them are ranked as the second and third fastest targets to halt the plant. The SDT caused by sensors 15 and 12 is about 1 and 10 minutes behind the fastest target to cause a shutdown (0,14 hours ≈ 8,6 minutes), respectively. Although our third selected weakness—sensor 8—takes 1.03 hours to cause a shutdown, it is more than twice faster than the next target in the ranking (sensor 11).

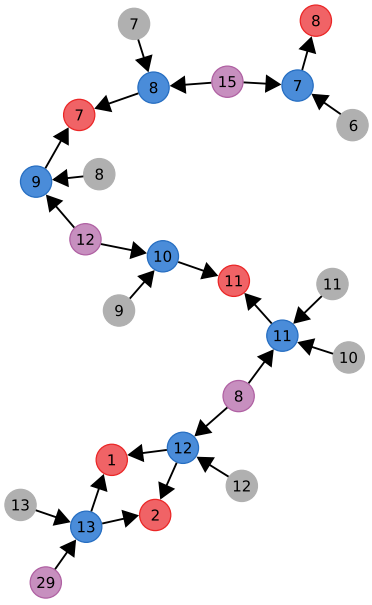
Table 3.4: Experiments using the constant value attack strategy under two different environmental conditions. The 3 sensors automatically identified are highlighted.

No disturbance		Disturbance 8	
Sensor	SDT (h) ▲	Sensor	SDT (h) ▲
7	0,14	7	0,14
15	0,16	15	0,16
12	0,31	12	0,31
9	0,44	9	0,44
8	1,03	8	1,03
11	2,49	11	2,65
23	6,55	23	5,84
18	72 (no shutdown)	18	72 (no shutdown)
29	72 (no shutdown)	29	72 (no shutdown)
30	72 (no shutdown)	30	72 (no shutdown)

For the remaining 28 experiments, we use the minimum and maximum value attack strategies. This time we use all disturbances in the range [1–13] because this control strategy is able to handle all of them. Moreover, we execute experiments without any disturbances, which adds up to 14 different environmental conditions. Again, due to space constraints, we summarize our results per attack strategy. Tables 3.5 and 3.6 show the average SDT and standard deviation for each sensor attack throughout all the experiments. For the minimum value attack, our approach finds the fastest target to halt the plant (sensor 15) with a SDT of only 0,65 hours ($\approx$ 39 minutes). However, the remaining two picks (sensors 12 and 8) take considerably more time. Nevertheless, they are more than 7 times faster than the next sensor in the ranking. For the maximum value attack, our approach finds the top 2 fastest targets (sensors 12 and 15), but this time sensor 8 is ranked fourth with a SDT more than 11 times faster than the next sensor in the ranking.

3.5 Related Work

Basic principles to protect computer systems have been laid out since the 1 970s [117]. The evolution of such work has identified both, secure software development practices and software weaknesses that must be avoided. Large weakness databases have been compiled to systematize such knowledge [96]. This has influenced the proliferation of automated tools that help software developers to spot security weaknesses in their code [31, 106]. Most of these efforts, however, are limited to the IT software domain. Just like their IT counterparts, smart building and other CPS



Control function	
Node	**Function name**
1	TC5
2	Tsep
3	TC6
4	PC4
5	CC7
6	CC8
7	LC-stripper
8	LC3
9	LC3-override
10	LC2
11	LC2-override
12	LC1
13	LC1-override

Setpoint	
Node	**Variable name**
1	tsepset
2	tstrpset
3	pset
4	Abase (literal 32.188)
5	Bbase (literal 13.823)
6	Lsp3 (literal 50)
7	Lsp4 (literal 50)
8	Lsp5 (literal 50)
9	Lsp1 (literal 50)
10	Lsp2 (literal 75)
11	Lsetpoint1 (literal 30)
12	Tsp (literal 75)
13	yasp (literal 30)

Calculated setpoint	
Node	**Variable name**
1	trset

Sensor	
Node	**Variable name**
7	xmeas 7
8	xmeas 8
9	xmeas 9
11	xmeas 11
12	xmeas 12
15	xmeas 15
18	xmeas 18
23	xmeas 23
29	xmeas 29
30	xmeas 30

Actuator	
Node	**Variable name**
1	xmv 1
2	xmv 2
3	xmv 3
4	xmv 4
6	xmv 6
7	xmv 7
8	xmv 8
9	xmv 9
10	xmv 10
11	xmv 11

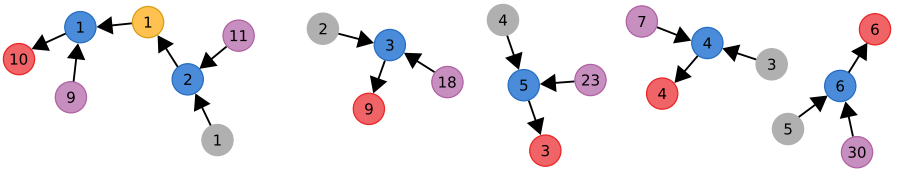


Figure 3.12: CCL graph of control strategy #2. The color of each node represents its type, as described in Figure 3.2. The name of each node corresponds to the variable name in the original Fortran code.

Table 3.5: Summary of experiments considering the minimum value attack strategy under 14 different environmental conditions. The 3 sensors automatically identified are highlighted.

Sensor Id.	Avg. SDT (h) ▲	Std. Deviation
15	0,65	0,20
9	1,28	0,51
7	1,72	0,79
12	4,25	1,72
8	8,29	1,98
23	63,03	22,83
30	66,01	16,82
29	72 (no shutdown)	0,00
11	72 (no shutdown)	0,00
18	72 (no shutdown)	0,00

Table 3.6: Summary of experiments considering the maximum value attack strategy under 14 different environmental conditions. The 3 sensors automatically identified are highlighted.

Sensor Id.	Avg SDT (h) ▲	Std. Deviation
12	0,69	0,16
15	0,70	0,07
9	1,09	0,40
8	4,48	0,55
23	52,01	29,46
7	55,30	27,60
29	67,46	16,97
30	72 (no shutdown)	0,00
11	72 (no shutdown)	0,00
18	72 (no shutdown)	0,00

applications are not exempt from software flaws.

While there are tools that help to formally prove the correctness of CPSs application code (i.e., that it does what it is supposed to do) [39], little attention has been paid to the identification of software weaknesses that could be exploited in presence of a malicious actor. One of the studies closest to our work is presented in [28]. The problem they address is how to find attack paths to perform indirect attacks on a fixed target. For instance, through variables that eventually flow towards the target. Our work, however, focuses primarily on the identification of weaknesses that might become the targets.

Most research on CPSs security argues, or simply assumes, that detailed knowledge about the system is a requirement to identify its weaknesses [26, 55, 80]. Typically, the aim to find such weaknesses comes from an offensive motivation. One of such works has identified the data sources required by an attacker to prepare a successful attack against ICSs [55]. For instance, PLC configuration, HMI/Workstation configuration, historian configuration, network traffic, system/component constraints, and piping and instrumentation diagrams. The main argument being that only through the combination of multiple data sources, an attacker is able to reach the level of "process comprehension" required to find a suitable target and launch an attack against it.

In this context, several researchers have analyzed the Tennessee Eastman Plant (TEP) from the cybersecurity perspective. For instance, in [80], the goal was to get insights on the resilience of the physical process under attack. They focused on compromising sensors and identifying those in best capacity to cause a shutdown on the targeted infrastructure. In this particular case, the targeted infrastructure was Larsson's implementation of the TEP, described in our evaluation as control application #1. Aligned with our results, they found out that sensors 4, 9, and 17 are the best targets under the minimum and maximum value attack strategies. However, they reached that conclusion by using a fully-fledged simulation of the targeted infrastructure. For an attacker to be able to build an accurate simulation of the targeted infrastructure, he would need access to at least, full PLC(s) configuration, P&IDs, and documentation about the system's constraints. On the contrary, we use the simulation only for evaluation purposes, not to find the weaknesses to target.

In another work, the goal was to find out the right time to launch an attack on individual sensor signals to cause a shutdown of the targeted infrastructure [81]. In this case, again, the target was Larsson's implementation of the TEP. The authors focused on DoS attacks on sensor signals, which forces control functions to use a stale value from the sensor. Under the assumption that launching sensor attacks at minimum or maximum peaks is the fastest way to cause a shutdown of the plant, the goal was to identify such peaks in real time. They approached

Table 3.7: Comparing required attacker knowledge in previous works. Data sources are based on the process knowledge data source taxonomy [55]. P&ID: Piping and Instrumentation Diagram, PLC: Programmable Logic Controller, HMI: Human-Machine Interface, WS: Workstation.

Data source	**Work**										
	[8]	[11]	[26]	[33]	[34]	[61]	[80]	[81]	[86]	[120]	**Our**
PLC config.	✓							✓			
HMI/WS config.		✓	✓			✓	✓				
Historian config.	✓	✓	✓	✓	✓	✓			✓	✓	
Network traffic				✓	✓						
P&ID	✓		✓			✓	✓	✓	✓	✓	✓

this challenge using the *best choice problem* methodological framework. Using different learning windows, they identified sensors 4, 9, and 17 as the best candidate targets (i.e., fastest shutdown time). To execute this approach a potential attacker might need at least network traffic access, a notion about the physical process and its potential disturbances (e.g., from P&IDs), and ideally, some historic data.

The works in [26] and [61], analyze diverse attacks against a simplified version of the TEP. Both works incorporate detailed knowledge from the process dynamics to execute the fastest attacks to halt the plant. An attacker leveraging the techniques proposed in these two works would require at least access to the PLCs configuration, historic data, and documentation about the system's constraints. We deem the simplified version of the TEP so small that we use it only as our motivating example (Section 3.2.2). However, as we show in the example, our approach is also applicable to this plant and our results match theirs.

No previous security works have used the TEP implementation by Luyben et al. [89], possibly because the code was only available in Fortran. We hope that our contribution in translating the code to MATLAB eases the challenging task of evaluating CPSs security research for future works.

Beyond the TEP, we have analyzed the information requirements of works aiming to identify weaknesses in other infrastructures for posterior exploitation. Table 3.7 summarizes our findings. Previous approaches combine many data sources and often require manual labor to interpret it. In contrast to previous works, our approach uses limited information to derive the CCL graph (e.g., piping and instrumentation diagram). Our graph-based model of the application is advantageous as it allows to automate the weakness identification process; one of the main goals of this thesis.

3.6 Discussion

We now deepen our discussion on the implementation of additional CWE weaknesses, the TEP results, the implications, and limitations of our approach.

Additional CWE Weaknesses. To describe the proposed approach, we elaborate on two weaknesses from Mitre's CWE database, specifically, global variables (CWE-1108) and multi-purpose variables (CWE-1109) [96]. We emphasize these two because they proved particularly useful to find weaknesses in both TEP implementations analyzed in our evaluation. Although it is not our goal to provide an exhaustive list of software weaknesses that can be mapped to control systems, here we discuss additional weaknesses that could be observed in CCL graphs.

- Circular dependencies (CWE-1047) happen when *"[t]he software contains modules in which one module has references that cycle back to itself."* [96]. In smart buildings, ICSs, and other control systems, circular dependencies can occur, for example, through functions that write their output to a calculated setpoint node that, in turn, is the input of another control function node and so on, until at some point the sequence of references return to the initial node.
- Deep nesting (CWE-1124) manifests in software that *"contains a callable or other code grouping in which the nesting / branching is too deep."* [96]. The control application of CPSs might contain a deep nesting weakness whenever a long sequence of closed control loops are chained together. For example, several cascade controllers concatenated. In this setting, the precise definition of 'long sequence' should be determined by a context dependent threshold.
- Functions with large "fan-out" (CWE-1048) refer to "invokable control element with large number of outward calls". According to Mitre, this weakness happens when the code *"contains callable control elements that contain an excessively large number of references to other application objects external to the context of the callable."* [96]. A CPS interpretation for this IT software weakness could be a control function that influences a large number of actuators and/or calculated setpoints. As before, the concrete definition of 'large number' is determined by a context dependent threshold.

TEP Results. The simulated attacks show that the automatically identified weaknesses are consistently among the targets in best capacity to halt the plant. Moreover, our results are in line with the results of previous research that leverage many more

data sources and are difficult to automate (see Related Work in Section 3.5). Although it is possible to argue that additional sources of information should be used to identify weaknesses more accurately, there is a trade-off between the amount (and complexity) of data sources and the feasibility to automatically process that data to find weaknesses. In the proposed approach, we use only CCL graphs, which allows us to identify weaknesses in control applications using well-known algorithms (e.g., graph centrality, loop detection, and path finding algorithms).

Implications. The goal of the approach presented in this chapter is to identify weaknesses to trigger other activities in a *vulnerability management process* (i.e., prioritization and remediation). Despite our defensive goal, this approach could also be used with offensive intentions. The information required to create CCL graphs (e.g., P&IDs), could be obtained through a variety of illegal means (e.g., phishing, social engineering, bribes) or simply downloaded from public repositories [79]. As it was shown, just by leveraging CCL graphs, a potential adversary could quickly jump from the reconnaissance phase to the identification of weaknesses to target. Thus, we acknowledge the double-edged use of our approach, as it happens with any weakness/vulnerability scanning tool.

Another implication of our approach is its applicability to multiple Cyber-Physical Systems. Since the CCL programming pattern is used in diverse applications (e.g., medical devices, autonomous vehicles, IoT-based systems, etc.), it is possible to search for weaknesses in their CCL graphs using this method given that the CCL graphs can be extracted. Without loss of generality, in this chapter we show its application for smart buildings and Industrial Control Systems.

Limitations. Although CCLs are a popular programming pattern for CPSs, there are other alternatives too. Programming patterns like *model predictive control* and *intelligent control*, are based on a single "black box" that receives inputs from all sensors and sends outputs to all actuators. These 3-layered programming patterns (multiple inputs + single processing unit + multiple outputs) are not suitable to perform the proposed analysis since it would simply identify the "black box" as a weakness due to, e.g., the large fan-out weakness (CWE-1048). Even with knowledge about the inner workings of the control component, it might be difficult to identify weaknesses inside it; for example, in case the "black box" is a neural network. Moreover, a graph abstraction might not be a natural representation for *fuzzy control* and other CPS programming patterns. The mismatch between the system model and the application programming pattern is an important limitation to apply the proposed approach.

3.7 Conclusion

Complex control systems, such as smart buildings and ICSs, are specifically tailored to meet local requirements. Their design is implemented locally and, despite any security guarantees provided by the manufacturers of individual components, the control application that aggregates them could introduce weaknesses. Therefore, the security of the aggregated system is a responsibility of local administrators.

In this chapter, we addressed **Research Question 1**: How to (semi-) automate the identification of weaknesses in smart building applications? To do so, our proposed approach transfers the accumulated knowledge about software weaknesses in the IT domain to the control systems field. We proposed a method to detect weaknesses in graphs comprised of interrelated closed control loops (CCLs). Such a graph is the base data structure to analyze and look for weaknesses including, but not limited to, global variables (CWE-1108) and multi-purpose variables (CWE-1109). Given the widespread use of the CCL programming pattern in diverse Cyber-Physical Systems (CPSs), the proposed method can be applied to other systems besides smart buildings.

Our approach is capable to find system-wide weaknesses. We tested it on a real smart building network comprised of 23 buildings and found several instances of global and multi-purpose variable weaknesses. We identified the root cause of such weaknesses as common practices in CPS control applications: on the one hand, the convenience of shared sensors and setpoints; and on the other hand, advanced control strategies such as *cascade control* and *override control*. Due to the consequences of tampering with real buildings, we were not authorized to deepen our analysis of the identified weaknesses through actual attacks against them. This would have been useful to provide further insights about their criticality. To overcome this problem, we looked at simulated CPS control applications where experimental attacks are safe to conduct.

The second evaluation was executed on two independently developed control applications of the Tennessee Eastman Plant (TEP), one of which was never analyzed from a security standpoint (Luyben et al.). A thorough and systematic effort led to the most extensive security analysis of the TEP to date: 1 392 individual attack simulations against all sensors used in the plant, considering 13 plant disturbances. Our results showed a correlation between the weaknesses identified by our approach and the most availability-critical targets according to the simulations. Our results on the TEP are in line with the results of previous research but using limited information and an automated approach.

Chapter 4

Weakness Identification in Smart Building Configurations

In Chapter 3, we started the discussion on the identification of weaknesses in smart building *applications*. We elaborated on why well-known IT weaknesses happen as well in OT systems that use different programming patterns. Moreover, we showed how to automatically identify weaknesses such as, but not limited to, global variables (CWE-1108) and multi-purpose variables (CWE-1109).

In this chapter, we address **Research Question 2**: How to (semi-) automate the identification of weaknesses in smart building configurations?

We continue our discussion on the automated identification of weaknesses, but now looking at the *configuration* of smart building components. Just like smart building application weaknesses, configuration weaknesses can be exploited by adversaries to attack smart buildings or to disguise malicious behavior. In particular, we look for configuration weaknesses that cause an *invalid* behavior of smart building components. By smart building *components*, we broadly refer to diverse software and hardware tools that are used to implement smart buildings. These include, operator workstation software, building controllers, human-machine interfaces (HMIs), and many more.

Our proposed approach is demonstrated in the context of smart buildings implemented using the BACnet protocol (ISO 16484-5) [15]. We identify weaknesses in smart building configurations leveraging on publicly available technical documentation about BACnet components. However, our approach could be applied to other industrial protocols, such as EtherNet/IP, that use similar documentation for their devices. The main challenge is to correctly interpret these technical documents, typically published in PDF and using non-standard layouts. After the automatic

interpretation of these documents, the goal is to extract a model of valid behavior for each component. The behavior model for each component is comprised of a set of rules that specify their capabilities as they are documented. After the valid behavior rules have been extracted, the weakness identification stage begins.

To find configuration weaknesses in smart buildings, we load the valid behavior rules into a network monitoring system, which passively observes network traffic and logs any rule violations. We analyze the components' outgoing network traffic to look for invalid behavior. Similarly, we analyze the components' incoming traffic to identify mistaken assumptions from other components on the behavior of the receiving component, i.e., whenever components are requested to behave in a way that contradicts their documented capabilities. In the context of Mitre's CWE taxonomy, here we look for *expected behavior violation* (CWE-440) and *inclusion of undocumented features or chicken bits* (CWE-1242) types of weaknesses [96].

4.1 Preliminaries

BACnet networks are comprised of components that fulfill different roles; for example, operator workstation software, building controllers, and human-machine interfaces (HMIs). During the smart building design phase, it is important to ensure compatibility among these diverse components. To ease this task, the manufacturers of certified BACnet components must provide, for each of them, a technical document called Protocol Implementation Conformance Statement (PICS) [15]. PICS specify which BACnet objects, properties, and BACnet interoperability building blocks (BIBBs) are implemented by BACnet components. Using the information provided in PICSs, it is possible to determine whether diverse components can interact and how. Thus, PICSs are both, an authoritative and a reliable source of information about the *valid* behavior of BACnet components. From the protocol's perspective, all components are BACnet devices, although they might not necessarily be specialized hardware. This means that even the operator workstation software, often running in a regular PC, is considered a BACnet device. For this reason, in what follows, we use "BACnet components" or "BACnet devices" as equivalent.

The BACnet standard is strict about the contents that PICSs must have, but lax in the document layout that PICSs may use [15]. For instance, object listings from different PICSs are shown in Figure 4.1. All of them describe the same kind of information using distinct table layouts. BACnet properties and BIBBs are also described in PICSs, however, more complex table layouts are used because they have additional features (e.g., writable properties, BIBBs that must be implemented

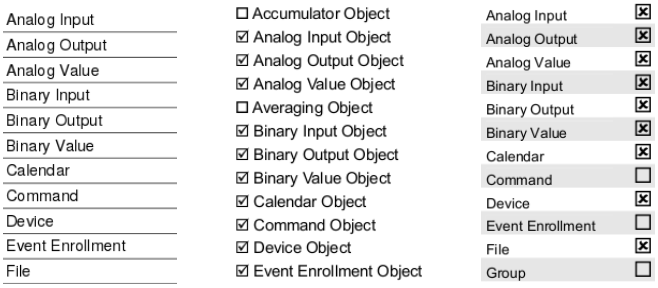


Figure 4.1: Excerpts from different PICSs stating which object types are implemented. PICS (a) lists only the implemented object types, whereas PICSs (b) and (c) use specific and different Unicode characters to denote which of them are implemented.

given the component's claimed profile, etc.).

4.2 Proposed Approach to Identify Configuration Weaknesses

We start the description of our approach with a high level overview in Section 4.2.1. We then zoom-in on the most important parts to explain how to automatically extract component's valid behavior rules from PICSs in Section 4.2.2. Finally, Section 4.2.3 describes the rule-based weakness identification process.

4.2.1 Overview

Our goal is to develop an automated approach to detect configuration weaknesses in smart building devices. To do so, we extract rules that specify the valid behavior of these devices. Such a set of rules is extracted from the PICSs available for each certified BACnet device. Automation of the PICS interpretation process is crucial because BACnet networks can be comprised of thousands of devices, each of them potentially described by its own PICS. Moreover, smart building devices are often added, removed, and replaced in the network, which requires updates on the set of valid behavior rules.

Our approach starts by collecting network traffic and PICSs of the components used in the smart building. The traffic collection is passive (e.g., from a mirroring port in a network switch) in order to avoid any accidental disruption of the system. The PICSs are collected from the public official repository [64]. Then, the PICSs

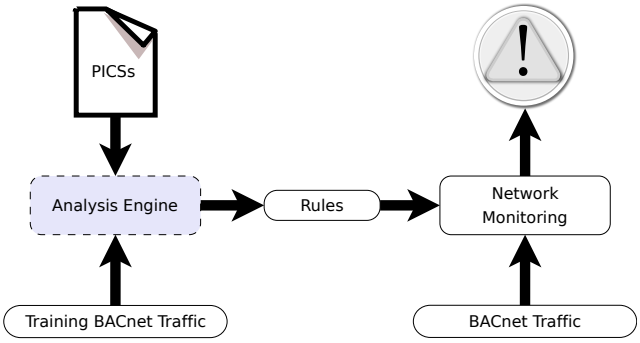


Figure 4.2: Overall workflow of the proposed approach.

are automatically interpreted by a software that we call the *Analysis Engine*. The Analysis Engine pays particular attention to the tables that specify the component's valid capabilities. Aided by the training traffic, the *Analysis Engine* identifies different table layouts used in PICSs and, consequently, correctly interprets the tables. A correct interpretation of tables leads to correct behavior rules for each component in the smart building. Once the rules have been extracted, they are loaded into a network monitoring system. Finally, the network monitoring system logs rule violations observed in the smart building's BACnet traffic. Figure 4.2 shows the overall workflow of our proposed approach.

4.2.2 Behavior Rules Extraction

In our approach, the *Analysis Engine* is in charge of extracting behavior rules from PICSs. A more detailed view of the *Analysis Engine* is shown in Figure 4.3. Its main tasks are: device fingerprinting, PICS matching, PICS interpretation, and rules creation. We now explain in more detail each of these tasks.

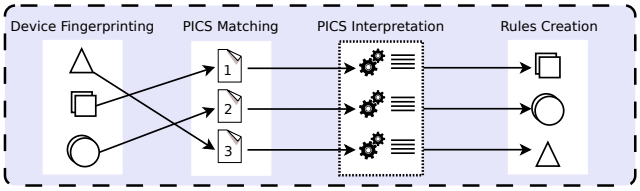


Figure 4.3: Analysis Engine tasks.

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

Our approach fingerprints BACnet devices applying two techniques described originally in [27]. The first technique is *"Device Object Analysis"*, which focuses on network packets that contain properties of the *device* object. The second technique is *"BACnet Address Linking"*. In this case, we link one packet containing the `device_id` and the BACnet address (both of which are unique in the network), with another packet that has the BACnet address but not the `device_id`. If the second packet contains the `brand` and/or `model` of the device, then we can link it to a specific `device_id` found in the first packet. The conditional presence of the `device_id` and the BACnet address in network packets is due to the peculiarities of the BACnet routing protocol.

It is worth noting that our system cannot extract valid behavior rules for devices not fully fingerprinted (i.e., incomplete tuples). No rules for a subset of devices imply that the network monitoring system will not be able to identify their configuration weaknesses. The example in Figure 4.3 shows five devices successfully fingerprinted, depicted as geometric figures.

PICS Matching. The *inputs* of the PICS matching routine are the list of fingerprinted devices and the set of PICS that describe them. Using the `brand` and `model` of the fingerprinted devices as keywords, this routine identifies each component's corresponding PICS. The *output* is a list of extended tuples `[device_id, brand, model, PICS_file]`. Continuing with the example in Figure 4.3, the devices represented by squares were matched with PICS 1, the devices represented by circles were matched with PICS 2, and so on.

PICS Interpretation. The *inputs* of the PICS interpretation process are the list of extended tuples and the training BACnet traffic. For every PICS file there are four *outputs*:

1. the set of implemented object types;
2. the set of implemented properties for each object type;
3. the set of writable properties for each object type; and
4. the set of implemented BIBBs.

The interpretation routine is the core of our rules extraction approach. In what follows, we will broadly refer to BACnet objects, properties, writable properties,

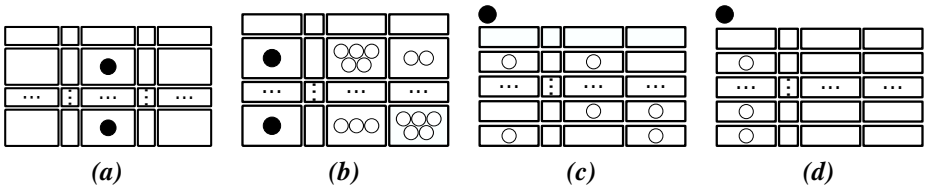


Figure 4.4: Table layouts commonly used in PICSs. Dark circles represent BACnet objects or BIBBs, whereas light circles represent BACnet properties. Empty cells might contain arbitrary text.

and BIBBs, simply as BACnet *elements*. Thus, we will explain, in a generic way, the interpretation routine for the different types of BACnet elements.

The first step is to classify the training traffic by device type. All the traffic sent by devices of the same brand and model is grouped. Looking for confirmations in the traffic (e.g., read service acknowledgments), the interpretation routine identifies the BACnet elements that are implemented for that group of devices. Similarly, looking for errors in the traffic, the interpretation routine identifies which elements are absent (i.e., not implemented) in that group of devices. We refer to the sets of present and absent elements as $\mathscr{P}$ and $\mathscr{A}$, respectively. The training traffic, from which $\mathscr{P}$ and $\mathscr{A}$ are populated, is collected during a limited time span. Hence, it is expected to get incomplete sets of implemented and non-implemented elements.

Since the list of standard BACnet elements is known in advance from the protocol specification, it is possible to automatically pinpoint these elements in each PICS. Figure 4.4 shows four examples of typical table layouts that describe BACnet elements in PICS. Once the elements' location has been identified, this routine groups them using different criteria. Figures 4.4a, 4.4c, and 4.4d suggest a column-wise grouping. Similarly, Figure 4.4b suggest a row-based grouping. Elements within the same cell (e.g., Figure 4.4b) are also grouped together. Finally, for any of the layouts, we create groups of elements with similar nearby text. For example, elements next to a symbol (e.g., ☑), word (e.g., *"Implemented"*) or phrase (e.g., *"Supported Object Types"*). In what follows we will refer to the groups extracted from the PICS as *reference sets*. Several reference sets are extracted for each PICS, but only one of them will contain the set of *all* implemented elements.

$$Jaccard(\mathscr{P}, R_i) = \frac{|\mathscr{P} \cap R_i|}{|\mathscr{P} \cup R_i|} \tag{4.1}$$

Our way to find out which of the *reference sets* has all the implemented elements, is by comparing them with our observations $\mathscr{P}$ and $\mathscr{A}$. The Jaccard coefficient is

a simple yet effective scoring mechanism to quantify the similarity between two sets [70]. Equation (4.1) defines the Jaccard similarity with arguments $\mathscr{P}$ and R_i, where R_i represents the i^{th} *reference set* extracted from a PICS. Using the Jaccard coefficient it is possible to rank the reference sets according to their similarity to $\mathscr{P}$. A *reference set* similar enough to $\mathscr{P}$ is expected to be the set of all the implemented elements. However, this might not always be the case, specially when $\mathscr{P}$ has few elements. To solve this problem we also use the set of absent elements $\mathscr{A}$.

$$Jaccard'(\mathscr{A}, \mathscr{P}, R_i) = \frac{|\mathscr{P} \cap R_i|}{|\mathscr{P} \cup R_i| \cdot (|\mathscr{P} \setminus R_i| + |\mathscr{A} \cap R_i| + 1)} \tag{4.2}$$

Equation (4.2) shows our modified version of the Jaccard similarity, which we call $Jaccard'$. $Jaccard'$ adds new terms to the $Jaccard$ similarity with the following effect:

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

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

3. In case both previous terms are equal to 0, we add 1 as a safeguard to avoid dividing by 0. We note that in this particular case $Jaccard' = Jaccard$, as the denominator is left unchanged (i.e., multiplied by 1). On the other hand, deviations from the ideal cases will increase the penalization (denominator) and cause a decrease in the overall scoring for R_i.

After interpreting each PICS, Figure 4.3 represents with four lines the four sets of rules extracted.

Rules Creation. The *inputs* for this routine are the sets of valid BACnet elements extracted from PICSs and the extended tuples (i.e., `[device_id, brand, model, PICS_file]`). The *output* are rules of the form `[device_id, valid_elements]`. Since all fingerprinted devices have already been matched with their corresponding PICS, it is straightforward to match each device with their valid elements. The rules creation routine is depicted in Figure 4.3 as a link between the rules extracted in the previous step with the devices fingerprinted in the first step, represented by geometric figures.

4.2.3 Weakness Identification

As it was shown in Figure 4.2, the rules generated by the *Analysis Engine* are loaded into a network monitoring system. The network monitoring system verifies that audited devices are concordant with their PICS in terms of valid objects, properties, writable properties, and BIBBs. Deviations from the components' valid behavior are flagged as configuration weaknesses. In this regard, it is worth noting that such weaknesses do not imply any violation of the BACnet communication protocol. In fact, all the weaknesses that our approach can find manifest as syntactically correct BACnet behavior, yet invalid for a particular device.

Verification of BACnet objects and properties is unambiguous because PICSs explicitly mention which of them are implemented or not. BIBBs verification is harder because the network communication is based on BACnet services that can be part of many BIBBs. For this reason, the network monitoring system verifies that the observed service is part of at least one of the supported BIBBs; a weakness is reported otherwise.

4.3 Implementation of the Proposed Weakness Identification Approach

We implement our prototype using third-party software tools and custom scripts. Our scripts are freely available under the GNU/GPLv3 license.[1]

Device fingerprinting is implemented using Zeek [108]. Its output are log files in plain text that are stored in an SQL database to ensure easy access from other modules. We use MySQL as our database management system. Nonetheless, the database scripts are written using standard SQL statements that can be handled by other database management systems as well.

The **PICS matching** process is done by Apache Solr [50]. This tool consists of a web application that ranks documents given a particular set of keywords. The behavior of Apache Solr is similar to Internet search engines but on a predefined set of documents. In our specific setting, the documents are PICSs and the keywords are the brand and model of all fingerprinted devices. The application programming interface provided by Solr allows us to automate the queries. In this way, we can quickly match all the devices with their corresponding PICS.

Prior to the **PICS interpretation** routine, we use Zeek to identify implemented and non-implemented BACnet elements in the training traffic. Since all the devices of the same brand and model are described by the same PICS, we aggregate our

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

observations in the database. Aggregated information per device type allows us to create the sets $\mathscr{P}$ and $\mathscr{A}$, required to use our $Jaccard'$ metric as defined in Equation (4.2).

The *reference sets* are extracted from PICSs only available in Portable Document Format (PDF). We circumvent the problem of directly reading PDF files by creating XML versions of the PICS using *Adobe Acrobat Pro*. Nonetheless, the process is error prone and the format and contents of the original file might be distorted in the output file. Moreover, there are typos and abbreviations in PICSs, that difficult to pinpoint BACnet objects and properties in the document.

We implemented a typo correction function based on the Levenshtein distance [85]. This approach measures how many deletions, additions, or substitutions are required to convert one string into another. Since we know in advance the list of standard BACnet elements, we compute the Levenshtein distance between each word in the PICS with all the elements in the list. If the Levenshtein distance is less than a predefined threshold, then we fix the string read from the PICS; else it is left unchanged. Our empirical studies show that one-fourth of the string length is a suitable fixing threshold for BACnet property names. For BACnet object names we use a fixed threshold set at 3.

Finally, a Python script reads from the database the network traffic observations ($\mathscr{P}$ and $\mathscr{A}$) and computes the $Jaccard'$ metric with all the reference sets extracted from a PICS. The reference set scoring the highest $Jaccard'$ value is considered the set of implemented elements. From this set, we create the rules that are going to be loaded into the network monitoring system.

The **rules creation** routine is also implemented by a Python script that queries the database to retrieve all devices of a particular brand and model, and links them with the automatically extracted rules. The monitoring rules are generated in Zeek's scripting language to easily load them into Zeek and quickly start the **weakness identification** phase. Nonetheless, our implementation can be adapted to support the input format of other monitoring systems too.

4.4 Evaluation of the Proposed Weakness Identification Approach

In this section, we present an evaluation of the implemented prototype. We start the evaluation by describing the intermediate results gotten during the training traffic analysis (Section 4.4.1). Then, we evaluate the precision and recall of the automatically extracted BACnet elements in Section 4.4.2. Finally, in Section 4.4.3, we analyze real BACnet traffic using our automatically extracted rules.

We complement our evaluation by presenting 3 concrete attacks that leverage configuration weaknesses (Section 4.4.4). Despite the *preventive* nature of our weakness identification approach, these scenarios showcase its attack *detection* capabilities too. These attacks are executed in a testbed.

4.4.1 Training Traffic Analysis

For the training phase, we use 4 days (4,5 GB) of BACnet traffic from the smart buildings network of the University of Twente. The training traffic is passively collected from one of the core switches of the University. The analysis of our training traffic fingerprints 646 BACnet devices. A breakdown of our findings is shown in Table 4.1.

Figure 4.5a shows that the discovery rate of BACnet objects in our network becomes stable after 3×10^6 BACnet packets, i.e., six hours or 300 MB of traffic. Since *device fingerprinting* is done using two techniques based on the analysis of BACnet objects (see Section 4.2.2), this means that all fingerprinted devices were identified within this time frame.

Our analysis of the overall infrastructure shows 9 804 object instances that we can confirm are present because the devices acknowledged them (e.g., after a read request); and 6 512 absent objects that triggered an error after being requested by some other device. This two aspects are shown in Figure 4.5a. Similarly, we discovered 40 467 present properties, whereas 1 237 were absent (see Figure 4.5b). Finally, we found 16 713 present services and 8 169 absent services (see Figure 4.5c). These present and absent BACnet elements are used later on to populate the $\mathscr{P}$ and $\mathscr{A}$ sets, required to compute our $Jaccard'$ metric

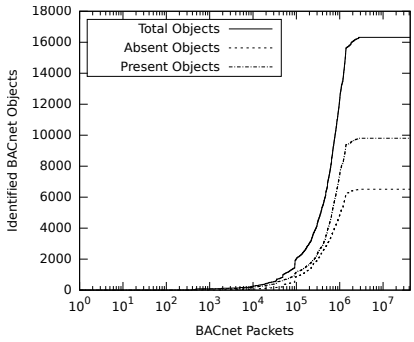
4.4.2 PICSs Interpretation

For all the PICSs shown in Table 4.1, we execute our algorithm to extract valid objects, properties, writable properties, and BIBBs. There are 2 missing PICSs that describe 6 devices in our infrastructure (shown at the bottom of Table 4.1). Thus, we focus our PICS interpretation evaluation on the remaining 10 PICSs. We present our results in terms of *precision* and *recall*. *Precision* refers to the fraction of extracted BACnet elements that are concordant with a PICS. *Recall* is the fraction of BACnet elements that were extracted from a PICS, considering the total amount of elements that can be extracted.

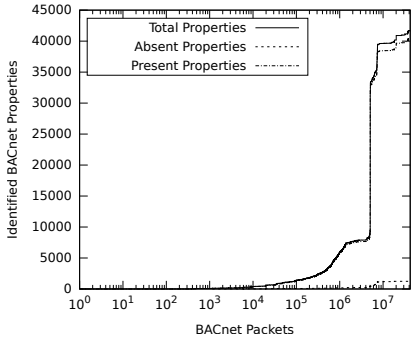
Table 4.2 shows the valid element specifications extracted from our PICSs set. We identify a total of 136 valid BACnet objects, 283 valid BIBBs, and 57 valid writable properties. All valid writable properties are extracted from only one

Table 4.1: Fingerprinted BACnet devices grouped by PICS file.

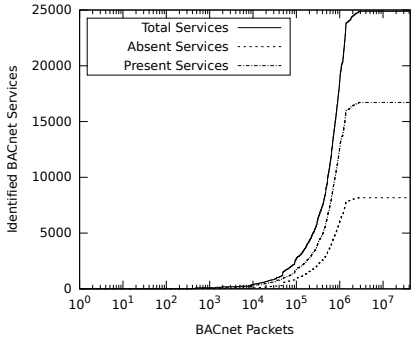
PICS file	Device manufacturer	Device model	Number of devices
PICS-1	DEOS	COSMOS Open	1
PICS-2	Delta Controls	eBCON	3
PICS-3	Kieback&Peter	DDC4400	5
PICS-4	Mitsubishi	BM ADAPTER	1
PICS-5	Priva	Compri HX 3	25
	Priva	Compri HX	7
	Priva	Compri HX 8E	87
	Priva	Compri HX 4	36
	Priva	Compri HX 6E	13
	Priva	Comforte CX	403
PICS-6	Priva	HX 80E	16
PICS-7	Priva	Blue ID S10	31
PICS-8	Priva	Blue ID C4	5
PICS-9	Siemens	PXC00-U	1
	Siemens	PXC128-U	3
	Siemens	PXC64-U	1
PICS-10	Siemens	PXG80-N	2
Not available	LOYTEC	LVIS-3ME15-G2	3
Not available	Siemens	PXR11	3



(a)



(b)



(c)

Figure 4.5: Discovery rate for BACnet objects (a), properties (b), and services (c) during the training phase.

PICS in our repository. This is because, although we observe write requests on different devices, all of them are described by a single PICS (PICS-5). Since we can only interpret PICSs for which we have observations $\mathscr{P}$ and $\mathscr{A}$, the proposed approach cannot extract writable properties from the remaining 9 PICSs. Despite this problem, which our approach is not designed to handle, we achieved 100% precision and recall for those three types of specifications.

Table 4.2: Specification extraction performance of implemented objects, BIBBs, and writable properties.

PICS File	Valid Objects	Valid BIBBs	Valid WProps.	Precision & Recall
PICS-1	17	38	-	100%
PICS-2	19	43	-	100%
PICS-3	22	49	-	100%
PICS-4	9	10	-	100%
PICS-5	16	27	57	100%
PICS-6	2	12	-	100%
PICS-7	16	31	-	100%
PICS-8	16	31	-	100%
PICS-9	18	34	-	100%
PICS-10	1	8	-	100%
TOTAL	**136**	**283**	**57**	**100%**

Our evaluation about the extraction of valid BACnet properties is summarized in Table 4.3. Overall, our algorithm finds 2 064 valid properties with a precision of 99,85%. The reason for precision loss is the misinterpretation of surrounding text strings as BACnet properties. Moreover, the overall recall is 99,57% due to abbreviations in property names that the typo correction routine is not able to fix. It is worth noting that some PICSs show only optional properties instead of all the implemented properties. Since our evaluation is based on the contents of the PICSs, implicit properties are not taken into account for our evaluation.

4.4.3 Smart Buildings Network

Following the workflow described in Section 4.2.1, the automatically extracted capabilities are then used to create behavioral rules for BACnet devices. Specifically, we created rules for the 240 BACnet devices whose PICSs are available.

We analyze BACnet traffic in our smart buildings network for two months, which in our setting consists of roughly 80GB of data. The network monitoring system (Zeek) logs all rule violations observed in the *outgoing* network traffic for

Table 4.3: Specification extraction performance of implemented properties.

PICS File	Property Specs.	Precision	Recall
PICS-1	244	100%	100%
PICS-2	224	99,11%	96,10%
PICS-3	520	100%	100%
PICS-4	46	100%	100%
PICS-5	237	100%	100%
PICS-6	41	100%	100%
PICS-7	242	99,59%	100%
PICS-8	244	100%	100%
PICS-9	259	100%	100%
PICS-10	7	100%	100%
TOTAL	**2 064**	**99,85%**	**99,57%**

each device. We summarize our findings in Table 4.4. There are 25 undocumented BACnet objects in 4 different device models. Since they are not documented objects, their occurrence is considered a weakness as they are not part of the expected behavior of these devices. Two of the undocumented objects are standard objects, namely, *analog-output* and *command*. The remaining 23 objects are proprietary. Similarly, we discovered 103 undocumented BACnet properties in 9 device models. Seven properties are standard (*time-of-device-restart*, *last-notify-record*, *feedback-value*, and four times the *profile-name* property) and 96 are proprietary.

Zeek also logs rule violations in the *incoming* network traffic of BACnet devices. These violations occur when devices are being queried about capabilities that they are not supposed to have according to their PICSs. In our setting such queries occur in less than 1,5% of the traffic mainly due to: (1) operator workstation software that tries to read a predefined set of objects and properties from BACnet devices; and (2) logging servers that regularly poll, via *read requests*, a predefined set of BACnet elements. In both cases, attempts to read not implemented objects and properties cause all the rule violations logged by Zeek. We omit further details of our *incoming* traffic rule violations because these findings are fully dependent on the specific configuration of our network.

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

Table 4.4: Undocumented BACnet objects and properties. Our findings are highlighted in bold font.

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

continues on next page...

Table 4.4 – ...continued from previous page

Device Model	Object	Property
	4 (binary-output)	**3067**
	5 (binary-value)	**3067**
	7 (command)	
	8 (device)	**3019, 3028, 3034, 3061, 3062, 3063, 3064**
	20 (trend-log)	**3019**
PXC64-U	4 (binary-output)	**3067**
	8 (device)	**3028, 3034, 3061, 3062, 3063, 3064**
	20 (trend-log)	**3019**

4.4.4 Experimental Attacks

In addition to the real BACnet network, we complement our evaluation using a simple testbed depicted in Figure 4.6. This testbed is used to illustrate potential attacks that leverage on invalid behavior weaknesses, such as those discussed throughout this chapter. Our testbed consists of two BACnet/IP devices interconnected by a switch. The diagram, in Figure 4.6, also shows a laptop that represents the attacker's position in the network. Both BACnet devices are implemented using Raspberry Pi computers running Raspbian OS. The BACnet software running on top of the OS is BACnet Stack 0.8.4 [74]. In order to reuse the specifications previ-

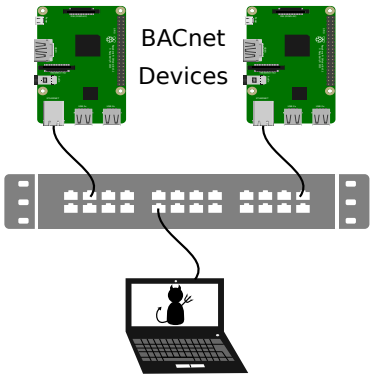


Figure 4.6: Diagram of our experimental BACnet testbed. The two Raspberry Pi computers at the top simulate the *Priva Blue ID S10* and the *Mitsubishi BM ADAPTER*, respectively. The computer at the bottom represents the attacker's access to the system.

ously generated, we configured the BACnet Stack demo server with similar features (i.e., objects, properties, and services) as the *Priva Blue ID S10* and the *Mitsubishi BM ADAPTER* that we have in our real network. The attacker's computer runs Kali Linux 2017.1. The network switch is an HP ProCurve 2610-24, in which we configured a mirroring port to collect the traffic exchanged during the experiments.

We present three concrete BACnet attacks and discuss how our weakness *identification* approach might also be able to *detect* such attacks. First, we show how a backdoor in a smart building device can be disguised as a BACnet object. The second attack is an active device fingerprinting executed by the popular `nmap` scanner [91]. The third and last attack discussed is a denial of service (DoS) in which the attacker attempts to remotely reboot a BACnet device.

Backdoor. Our first attack consists of a BACnet device whose firmware has been tampered with by a malicious actor. We added a backdoor in our BACnet device simulating the Priva controller. The backdoor is located in a standard BACnet object (CharacterString Value) that implements all the mandatory properties (e.g., object-identifier, present-value, status-flags, etc.) and common optional properties (e.g., description, out-of-service, event-state, etc.). The backdoor functionality is implemented by means of a writable property that receives commands from the attacker, and a readable property which holds the output of the attacker's command. Specifically, the writable property is *present-value* and the readable property is *description*. In this way, the attacker is able to send OS commands (e.g., ls, cat, rm, etc.) and to read the output of those commands, all using syntactically valid BACnet objects, properties and services. Since the *CharacterString Value* object is not part of the device's PICS, our specifications spotted the violation even though we sent only one packet consisting of a write request to the *present-value* property. The same functionality could have been implemented in proprietary BACnet objects as we found many in our real BACnet network (see Section 4.4.3). In this case, our proposed approach is able to identify the backdoor by looking at the incoming traffic and the outgoing traffic of the compromised device. The first rule violation happens when the compromised device receives a request to write a property in an invalid object. The second rule violation is triggered when the compromised device complies with requests to read the invalid object.

Active Device Fingerprinting. The second attack consists in executing reconnaissance tools such as `nmap` against our two BACnet devices [91]. Nmap is a popular port scanner that can be extended via scripts. There is a script specifically created to scan BACnet networks, which tries to fingerprint devices executing queries on multiple properties of the *device* object [58]. The requested properties are *vendor-identifier, vendor-name, object-identifier, firmware-revision, application-*

software-version, object-name, model-name, description and *location*. From the attacker's laptop, as shown in Figure 4.6, we execute the `nmap` script against the two emulated BACnet devices in our testbed. The Priva device does not report any rule violations because all 9 properties requested by `nmap` are valid according to its PICS. On the other hand, the rules of the Mitsubishi device trigger two violations because the *description* and *location* properties are not implemented in that specific model according to its PICS. This example shows the main drawback of our approach: attacks that leverage on documented behavior of devices cannot be detected by our rules. At the same time, we show that we can detect stealthy attacks comprised of as few as a single network packet, as long as it represents a violation to the device's PICS.

Denial of Service. A third attack is launched against the simulated Mitsubishi device. We use the emulated Priva device as a pivot to reach the target. Taking advantage of the backdoor in the simulated Priva device, we downloaded a malicious binary file that sends *reinitialize* requests to the target. The binary file is based on the demo `bacrd` command in BACnet-Stack [74].

Besides the rule violations generated because of the backdoor, two additional violations are logged. The first of them is due to the fact that the pivot device is not supposed to send *Reinitialize Device* requests. This violation is detected by analyzing the outgoing traffic of the pivot device. The second violation is triggered because the target device is not supposed to receive such a request. This violation is detected by analyzing the incoming traffic of the target device. Both reported violations demonstrate the usefulness of our BIBBs rules in a realistic attack scenario.

4.5 Related Work

Open communication protocols are usually described in documents such as RFCs and IEEE or ISO standards. Several researchers have used these descriptions to *manually* extract expected communication behavior rules [72, 104, 121]. Their approaches, however, are focused on detecting communication protocol violations, such as syntactically incorrect messages caused by devices with buggy protocol implementations. To mitigate this kind of problems in smart buildings, Kaur et al. developed a traffic normalizer that either drops or fixes (whenever possible) malformed BACnet packets [76]. This traffic normalizer does not deal with specific details of every device. Instead, it considers general aspects of the protocol like header fields length or valid status flags. On the other hand, our approach is not focused on the BACnet communication protocol itself, but on the devices' invalid

behavior that can be observed in well-formed BACnet messages. For this reason, rather than using the BACnet standard specification, our valid behavior rules are extracted from the devices' PICSs [64].

Most security research in the smart buildings domain addresses the problem of intrusion detection. Tonejc et al. [132] evaluate four different machine learning algorithms to perform anomaly-based intrusion detection in smart buildings implemented using the BACnet protocol. A different approach presented in [29] uses network flows to detect attacks in BACnet systems. The closest work to ours was done by Caselli et al. [27]. The authors developed an approach to extract specifications from BACnet PICSs written with a predetermined format. Their goal was to perform specification-based intrusion detection in BACnet systems. Their approach suffers from two major limitations: (1) the parsing process does not yield results from documents using a different format; and (2) even for documents following the dictated format, the combination of device capabilities (text strings) and auxiliary information (arbitrary symbols or text) cannot be disambiguated. Rather than developing a rigid extraction mechanism for each PICS format found in the official repository, our approach generalizes the way to interpret different PICS layouts using network traffic to solve the incompleteness and ambiguity problems in [27]. Our experimental results, based on our new approach, show a significant improvement over the state of the art. Given the fundamental differences between their design and ours, the implementations also differ in many aspects. The gain in flexibility offered by our approach requires additional processing (e.g., network traffic analysis) and infrastructure (e.g., databases). Finally, the experiments presented in [27] demonstrate how rule violations help to discover anomalous behavior that resembles BACnet attacks. Our research not only validates and measures their findings but also presents realistic attack scenarios implemented and executed on an isolated testbed. Unlike the work in [27], the core of our research lies on automatic specification extraction, for this reason, we also evaluate the performance of our algorithm in terms of precision and recall.

The identification of weaknesses has been largely unexplored in the smart buildings domain. There are, however, vulnerability scanners that focus on generic smart buildings software such as the operating systems pre-installed in building controllers, operator workstation software, human-machine interface (HMI) firmware, and others [30, 129]. In contrast, we focus on the identification of weaknesses caused by component misconfigurations. These misconfigurations are often specific to a location and, therefore, tools that search for CVEs cannot find them.

4.6 Discussion

In this section, we discuss three core stages of our approach. In their execution order, we address the training phase, the creation of rules, and the identification of weaknesses.

Training Phase. Our PICS interpretation algorithm requires the availability of training BACnet traffic. We passively collect training traffic from a real network during 4 days. Nonetheless, we learn the vast majority of features during the first 6 hours. However, this might differ in other smart buildings depending on their network activity and the monitoring point where the traffic is collected. The BACnet traffic required for training purposes must reveal as many objects, properties, and services as possible. More important than the amount of traffic is the diversity of information in it. Our own experiments suffer from monotonic traffic regarding write requests. Even though we observe several write requests in our training traffic, all of them are directed to devices described by the same PICS. Therefore, later on we are only able to extract writable properties from one PICS. One way to overcome this problem is by collecting traffic for a longer period, which would increase the chance to observe relatively rare events (e.g., monthly *write* events). Additionally, in smart buildings where passive traffic collection is not a strong requirement, actively probing devices can reveal the required features in short time.

Rules Creation. PICS constitute the source of our valid behavior rules. Although all certified BACnet devices are required to have a PICS, not all BACnet devices are certified and, therefore, the availability of PICS is not guaranteed. In our network, there are 6 non-certified devices that should be described by 2 PICS, however, the documents are not available in the official PICS repository. Thus, we recommend to use only certified devices for two reasons. First, it guarantees compliance with the BACnet standard. Second, this ensures that PICS are publicly available in the BACnet International website [64]. The same recommendation has been backed by local regulatory organizations such as AMEV in Germany [45].

Weakness Identification. In our experiments, most identified weaknesses are requests to non-implemented properties. According to Mitre's CWE taxonomy, this is an *expected behavior violation* weakness (CWE-440). After exploring why this is the case, we find out that operator workstations' software often requests a predefined set of properties (particularly on the *device* object), some of which are not mandatory. Thus, device manufacturers often decide not to implement these optional features and, whenever they are requested, a rule violation is triggered. The fact that legitimate management tools are not properly configured to make only valid requests is problematic because their behavior becomes indistinguishable

from the behavior of reconnaissance tools like `nmap`, as we show in Section 4.4.4. Since every invalid request generates an error from the recipient, the presence of errors in the traffic is common. Such common errors, however, might desensitize operators who will not pay attention when the same errors happen due to malicious activities.

The second cause of alerts are undocumented elements in BACnet devices. This is commonly referred to as an *inclusion of undocumented features or chicken bits* according to Mitre's CWE taxonomy (CWE-1242). This is a weakness that, as we show in Section 4.4.4, can be misused to conceal backdoors or other malicious behavior. It is possible to argue that the same behavior could have been implemented in a *documented* object and, therefore, remain undetected by our approach. This is, indeed, the case because we consider all documented behavior as valid. However, it is worth noting that we found 128 undocumented elements in our real BACnet network. Although in cooperation with the local smart building administrators we discarded any attacks in the analyzed network, undocumented elements are potential hide outs for malicious activities. Thus, they are weaknesses that must be managed accordingly (e.g., replacing devices, disabling features such as proprietary objects, stricter monitoring). The evidence obtained by analyzing real and synthetic traffic supports the importance of our weakness identification approach.

4.7 Conclusion

In Chapter 3, we discussed the identification of security weaknesses in smart building applications and this chapter complements it with an approach to identify configuration weaknesses. Just like application weaknesses, configuration weaknesses pose a serious threat for smart building infrastructures. Adversaries can leverage configuration weaknesses to execute direct attacks or to disguise other malicious activities against smart buildings.

In this chapter, we addressed **Research Question 2**: How to (semi-) automate the identification of weaknesses in smart building configurations? To answer this question, we presented an automated approach to create valid device behavior rules that are monitored at network level. These rules are able to identify *expected behavior violation* (CWE-440) and *inclusion of undocumented features or chicken bits* (CWE-1242) types of weaknesses, often happening due to misconfigurations. We implemented our prototype for smart buildings using the BACnet protocol; more precisely, leveraging Protocol Implementation Conformance Statements (PICSs) of certified BACnet devices. Despite the specific application for smart buildings implemented using the BACnet protocol, the proposed approach could be used on

other industrial protocols, such as EtherNet/IP, with similar protocol syntax and documentation. IoT-based systems could also be subject to the proposed analysis. However, their unregulated documentation requirements would likely represent a serious limitation. Our behavior rules are individually tailored for each device in the network, in such a way that they (1) identify undocumented behavior in devices' PICSs; and (2) identify requested behavior that devices are not supposed to have according to their PICSs.

The high precision and recall achieved during the PICS interpretation process translate into a reliable identification of weaknesses. Particularly on CWE-1242, we showed a widespread practice of BACnet manufacturers to incorporate undocumented features in their device implementations. Whereas in some device models such a behavior can be configured (i.e., enabled or disabled), this is the default and immutable behavior for other devices. On the other hand, about CWE-440, we showed that building administrators may configure components assuming a specific yet incorrect behavior on other components. This mismatch between the expected and actual behavior of devices triggers network protocol errors that become "normal" in the smart building's daily operation; thus, desensitizing administrators to potentially malicious activity that might cause the same type of errors. Our evaluations using both, real and synthetic traffic, proved that PICS derived rules are also useful to identify behavior that closely resembles cyber attacks (e.g., backdoor, active device fingerprinting, and denial of service).

This chapter concludes our contributions on the identification of security weaknesses. The next chapter discusses the assessment of security weaknesses in smart buildings. This assessment is part of the second step of the vulnerability management process, namely, the prioritization of security weaknesses.

Chapter 5

BACRank: A Sensitivity Assessment of Smart Building Components

In Chapter 3 and Chapter 4, we presented two methods to identify security weaknesses in smart building applications and configurations, respectively. The *identification* of security weaknesses is a crucial first step in any vulnerability management process because it sets the ground for the remaining activities of the process, namely, their prioritization and remediation.

The *prioritization* of security weaknesses aims at finding the right order to remediate them. This is a difficult task because of the diverse aspects that can be considered to create such a ranking. Previous works have explored social media discussions, severity, expected exploitation time, and others [32, 47, 97]. Proprietary tools have even home brewed their own secret methods to do it [109, 110, 129]. However, existing prioritization methods are unfit for the smart buildings domain because they fail to capture the potential impact of cyber-physical attacks and their consequences for the targeted organization. We argue that a comprehensive estimation of risk that considers the particular context of the affected organization should be the way to rank security weaknesses for remediation. Although literature typically defines risk as a seemingly straightforward multiplication of *impact* and *probability* of loss events, each of these two aspects is comprised of several other factors that must be estimated first [73]. Particularly on the *impact* aspect, one of the key factors is the *sensitivity* of assets, i.e., the harm that a loss event might cause in terms of, e.g., legal/regulatory consequences, fines, and others [73]. Taking into account that building services in many public and private buildings are subject to

national and/or international regulations, we consider the assets' sensitivity as an important feature to prioritize the security weaknesses that affect them.

In this chapter, we address **Research Question 3**: How to (semi-) automate a sensitivity assessment for smart building components?

We propose to combine technical features of the smart building and business features of the organization hosted in it, to determine the sensitivity of smart building components. On the business side, we identify the business processes that take place and their sensitivity for the organization. Among the technical aspects, we consider smart building components and their dependencies. Using this information, we match the smart building components with the business processes that they support. This match allows us to assign an initial sensitivity score to smart building components based on the sensitivity of their corresponding business processes. However, these scores are adjusted based on the components' dependencies and the role they play in the overall infrastructure.

More concretely, we create a graph data structure where the nodes represent smart building components and the edges represent their functional dependencies. The smart building *components* can be established at diverse granularity levels, ranging from fine grained data points (e.g., individual sensors, setpoints, and actuators) to coarse grained devices (e.g., building controllers and PLCs). To illustrate our approach and without loss of generality, in this chapter we opted for using *software modules* as components (nodes of our graph) at an intermediate level of granularity. We then set an initial score to each software module based on the business processes that they support. Lastly, the final sensitivity quantification is assigned by a novel graph centrality metric specifically designed for this purpose. We call the proposed graph centrality metric *BACRank*, in reference to the Building Automation and Control (BAC) systems that are used to implement smart buildings.

5.1 Preliminaries

Critical infrastructures like hospitals, airports, and data centers are typically demanded to implement business continuity management systems (BCMSs) to improve the resilience of their core operations [36, 140]. Therefore, such organizations follow guidelines like those proposed in ISO standards 22 301 and 27 031, on this matter [68, 69]. The first step in the implementation of a BCMS is to execute a business impact analysis (BIA).

The main purpose of the BIA is to score the sensitivity of business processes. According to ISACA, a *business process* is "[a]n inter-related set of cross-functional activities or events that result in the delivery of a specific product or service to a

customer" [65]. A BIA is typically based on questionnaires answered by the stakeholders of each business processes. These questionnaires ask questions about the consequences of business processes halted during different time ranges (e.g., 0–2 hours, 2–8 hours, etc.), considering the legal, reputational, and financial repercussions for the organization. In this questionnaires, the stakeholders assign a sensitivity level to each hypothetical scenario for the processes that they are involved in. Lastly, the overall sensitivity level of each business process is typically computed as the average of the answers gotten from the questionnaires. A BIA contains additional information about business processes such a calendar that specifies their execution period, which is a useful time reference to prepare for potential loss events. For further details on the business impact analysis, we refer the reader to the ISO standards 22 301 and 27 031 [68, 69].

5.2 Proposed Sensitivity Assessment Approach

We now present our proposed approach to estimate the sensitivity of smart building assets. We begin with a summary of the information required to implement our approach in Section 5.2.1. In Section 5.2.2 we present an overview of our approach and introduce the notation that we will use to describe it. In Section 5.2.3 we explain how to set an initial score for each asset and Section 5.2.4 details the requirements that a graph centrality algorithm, according to the proposed approach, must satisfy to adjust the initial scores and obtain a final sensitivity score. Finally, in Section 5.2.5 we present an instance of a graph centrality algorithm that satisfies the requirements stated in the previous section.

5.2.1 Information Requirements

To implement a comprehensive *sensitivity* assessment for smart building components, technical and business aspects must be taken into consideration. In our proposed approach, it is required to make explicit the relation between smart building services and business processes. We propose to do so by considering their physical overlap in the building. Two views of the building layout are needed: one segregated by business processes and the other segregated by the area of influence of the building services. We combine the two views of the building layout to unveil the mapping between building services and business processes. Moreover, we need to quantify the support of building services on the related business processes. This quantification can be expressed as a percentage where 100% means that the service in question is fully required, whereas lower values characterize weaker dependencies on building services.

On the business side, it is important to identify sensitive business processes for the organization, so that we can map such sensitivity to the corresponding building services. It is also important to know *when* business processes are executed to adjust the sensitivity of their supporting smart building services accordingly. Business processes' sensitivity and execution time can both be found in the BIA that is typically available in organizations that implement a BCMS.

On the technical side, it is important to know *when* the building services are actually needed since they might not be relevant out of their duty periods (e.g., the heating service during summer). Furthermore, it is crucial to understand *how* building services are implemented to take into account possible chain reactions. To obtain this information, we assume access to the smart building's design documentation to identify (1) all building services; (2) their duty cycles; (3) their underlying components; and (4) the functional dependencies among components. Since some dependencies might be stronger than others, a quantification is needed in this regard. We propose to use a percentage value as a simple mechanism to denote the dependence strength, where 100% means that the *consumer* module cannot operate without the *provider* module, and lower values represent, proportionally, weaker dependencies.

Table 5.1 summarizes the information requirements of our proposed approach. While most of the information is typically already available in mature and critical organizations, we emphasize three components where an expert's judgment is required: (1) the building services' support on business processes; (2) the components' dependency strength; and (3) the building services calendar.

We argue that the previously mentioned list of business and technical information constitutes a basic subset that allows us to link the smart building with its business purpose. We do not discard, however, that other business or technical aspects could be included to complement or replace some elements in the proposed list.

5.2.2 Overview

We abstract the smart building as a directed graph data structure where its components are represented by vertices and their functional dependencies are represented by edges. The edge direction denotes the way information flows and its weight represents the dependency strength. Formally, the smart building is defined as a graph $G(V, E)$ where V is a nonempty set of vertices (or nodes) and E is a set of edges. Each edge links two vertices in V. An edge $e \in E$ is represented as $e_{u,v}$ where u and v denote the source and the destination of the edge, respectively. Edge weights are represented as a function $\omega : E \rightarrow [0, 1]$ that assigns each edge $e \in E$

Table 5.1: Information requirements summary. Expert-based information is shown in *italic* font.

Business Information	BACS Technical Information
Business continuity plan	Engineering design
↳ BIA	↳ Building services list
↳ Business process list	↳ *Calendar*
↳ Score	↳ Components
↳ Calendar	↳ Dependencies
	↳ *Strength*
Building layout	Building layout
↳ Segregated by business processes	↳ Segregated by building services
↳ *Building services support on the overlapping business processes* ↲	

a weight $\omega(e)$. The set of edges with destination $m \in V$ is defined as $\Gamma^-(m)$ and the set of vertex origins in $\Gamma^-(m)$ is $N^-(m)$. Analogously, the set of edges with origin m is defined as $\Gamma^+(m)$ and the set of vertex destinations in $\Gamma^+(m)$ is $N^+(m)$.

In graph theory and network analysis, the identification of important vertices in a graph is done by means of graph centrality algorithms. Since our goal is to identify *important* components in smart buildings represented as graphs, we model our approach as a graph centrality metric where the notion of *important* is based on their sensitivity. The proposed metric is comprised of two parts. First, a set up procedure that assigns vertices an *initial score* based on the BIA score of the related business processes and the components' support to them. Second, a *graph centrality metric* that estimates the chain reactions of unavailable components and allows us to rank the smart building components based on their sensitivity scores. The next two sections detail both parts.

5.2.3 Initial Score

Notation. The set of business processes running in the building is defined as $P = \{p_1, ..., p_n\}$. The set of building services offered is defined as $S = \{s_1, ..., s_m\}$. The BIA score assigned to each business process is defined as a function $\beta : P \rightarrow [0, 1]$, where $\beta(p_i)$ for any $p_i \in P$ is proportional to p_i's sensitivity for the organization. Given an arbitrary $s_i \in S$ and $p_j \in P$, the estimated support s_i provides to p_j is defined as a function $\gamma : S \times P \rightarrow [0, 1]$, where higher values denote stronger support. Since building services are comprised of one or more smart building components, we can formally state that $s_j \subseteq V$ for any $s_j \in S$. For an arbitrary

component $m \in V$ that is part of service $s_j \in S$, the support m provides to an arbitrary $p_k \in P$ is given by $\gamma(s_j, p_k)$. Finally, we define a *time* function that takes two inputs: 1) the object whose calendar is going to be inspected (either a business process or a smart building component); and 2) a time point t. The output is binary and indicates whether the object given as first input is running/needed or not at time t, denoted by a 1 or a 0, respectively.

The initial score given to each component in the graph is a numerical value that summarizes three important aspects: 1) the relevance of the supported business process represented by function β; 2) the components' support to each business process represented by function γ; and 3) the time in which both, the business process is running *and* the component's building service is needed.

To determine the initial sensitivity score that component $m \in V$ has over a business process, we multiply $\beta(p_i) \cdot \gamma(s_j, p_i)$ for all $p_i \in P$, given that m is part of $s_j \in S$. Computing the initial score for component m at time t, denoted $\delta(m, t)$, consists of taking the maximum value among active business processes, given that the building service of which component m is part, is also active at time t. Formally,

$$\delta(m,t) = \begin{cases} \max_{1 \le i \le n}(\beta(p_i) \cdot \gamma(s_j, p_i)) & \text{if time}(p_i, t) = \text{time}(m, t) = 1, \\ 0 & \text{otherwise.} \end{cases}$$

5.2.4 Graph Centrality Measure Requirements

We propose to estimate the propagation effect of unavailable components by means of a graph centrality metric. Before describing the requirements for such centrality metric we introduce some definitions.

Definition (Component *equivalence*). Two components $m_1, m_2 \in V$ are *equivalent (at time t)* (denoted as $m_1 \equiv m_2$) if all of the following properties hold:

$$N^+(m_1) = N^+(m_2) \tag{5.1}$$

$$N^-(m_1) = N^-(m_2) \tag{5.2}$$

$$\forall e_{m_1,n} \in \Gamma^+(m_1), e_{m_2,n} \in \Gamma^+(m_2) : \omega(e_{m_1,n}) = \omega(e_{m_2,n}) \tag{5.3}$$

$$\forall e_{n,m_1} \in \Gamma^-(m_1), e_{n,m_2} \in \Gamma^-(m_2) : \omega(e_{n,m_1}) = \omega(e_{n,m_2}) \tag{5.4}$$

$$\delta(m_1, t) = \delta(m_2, t) \tag{5.5}$$

Due to properties 5.1–5.4, components m_1 and m_2 have the same n-degree neighborhood for all possible n values (immediate neighbors, neighbors of neighbors, etc.).

Definition (Component *equivalence with exception*). Two components $m_1, m_2 \in V$ are called *equivalent with exception* if at least one of the above equivalence properties is violated. In this case, we explicitly mention the exception and denote this as $m_1 \equiv_e m_2$ (exception).

In what follows, we define three basic requirements that a centrality measure $\Delta(m, t)$ for component m at time t, must satisfy to coherently measure the sensitivity of smart building components.

1. For any two active components one of which, ceteris paribus, has higher initial score, must score higher.
The aim of the first requirement is to ensure that the sensitivity score difference of two components with identical topological features in the graph is determined by their initial score. Formally, for all active components m_1, m_2:

$$m_1 \equiv_e m_2 \, (\delta(m_1, t) > \delta(m_2, t)) \Rightarrow \Delta(m_1, t) > \Delta(m_2, t)$$

2. For any two active components one of which, ceteris paribus, sends information to an active component with higher sensitivity score than its parallel, must score higher.
The goal of the second requirement is to acknowledge that feedback plays an important role in components dependency graphs. Unlike web centrality metrics, where a node's feedback contribution comes from other nodes that point to it [24], components in our setting get their contribution from the components they send information to. The rationale being that the receiving components depend on the input provided to execute their functions. Formally, for all active components m_1, m_2:

$$m_1 \equiv_e m_2 \, (\exists n_1 \in N^+(m_1), n_2 \in N^+(m_2) : N^+(m_1) \setminus \{n_1\} = N^+(m_2) \setminus \{n_2\} \wedge \omega(e_{m_1,n_1}) = \omega(e_{m_2,n_2}) \wedge \Delta(n_1, t) > \Delta(n_2, t)) \Rightarrow \Delta(m_1, t) > \Delta(m_2, t)$$

3. For any two active components one of which, ceteris paribus, sends information to an active component with stronger dependency than its parallel, must score higher.
The purpose of the third requirement is to emphasize that the link strength regulates the fraction of the sensitivity score to be transferred from the destination vertex to the source vertex. Formally, for all active components m_1, m_2:

$$m_1 \equiv_e m_2 \, (\exists! n' \in N^+(m_1) = N^+(m_2) : \omega(e_{m_1,n'}) > \omega(e_{m_2,n'})) \Rightarrow \Delta(m_1, t) > \Delta(m_2, t)$$

5.2.5 BACRank

Taking into account the requirements stated before, we define a new graph centrality measure called BACRank. The BACRank score of vertex m at time t, is computed as its initial score $\delta(m,t)$ plus a contribution from the vertices m points to. From those vertices, m will get a percentage of their BACRank score determined by the strength of the link, represented by $\omega(e_{m,n})$. This mechanism boosts the scores of vertices that are highly important from a sensitivity standpoint. The algorithm is defined as

$$\text{BACRank}(m,t;i) = \begin{cases} \delta(m,t), \text{ at iteration } i = 0, \\ \delta(m,t) + \sum_{n \in N^+(m)} \text{BACRank}(n,t;i-1) \cdot \omega(e_{m,n}), \text{ for } i > 0. \end{cases}$$

The BACRank scores are computed iteratively. The algorithm is said to converge if for all vertices their score difference in two consecutive iterations is less than a small value ε. An empirical convergence proof is provided in Section 5.4, as shown in Figure 5.3. To keep BACRank scores bounded, all of them are normalized in the range $[0, 1]$ after every iteration.

In what follows, we formally proof the requirements from Sec. 5.2.4. For the sake of brevity, in the proofs we refer to BACRank simply as BR.

Proof of requirement 1. Let m_1, m_2 be two active components such that:

$$m_1 \equiv_e m_2 \; (\delta(m_1,t) > \delta(m_2,t)) \tag{5.6}$$

Then the following holds for any iteration $i > 0$:

$$\begin{aligned} \text{BR}(m_1,t;i) &> \delta(m_2,t) + \sum_{n \in N^+(m_1)} \text{BR}(n,t;i) \cdot \omega(e_{m_1,n}) && \text{(by (5.6))} \\ &= \delta(m_2,t) + \sum_{n \in N^+(m_2)} \text{BR}(n,t;i) \cdot \omega(e_{m_1,n}) && \text{(by equiv. prop. (5.1))} \\ &= \delta(m_2,t) + \sum_{n \in N^+(m_2)} \text{BR}(n,t;i) \cdot \omega(e_{m_2,n}) = \text{BR}(m_2,t;i) \\ & && \text{(by equiv. prop. (5.3))} \end{aligned}$$

Proof of requirement 2. Let m_1, m_2 be two active components such that:

$$\begin{aligned} m_1 \equiv_e m_2 \; (\exists n_1 \in N^+(m_1), n_2 \in N^+(m_2) : N^+(m_1) \setminus \{n_1\} = N^+(m_2) \setminus \{n_2\} \wedge \\ \omega(e_{m_1,n_1}) = \omega(e_{m_2,n_2}) \wedge \text{BR}(n_1,t;i) > \text{BR}(n_2,t;i)) \end{aligned} \tag{5.7}$$

Then the following holds for any iteration $i > 0$:

$$\mathrm{BR}(m_1, t; i) > \delta(m_1, t) + \sum_{n \in N^+(m_1) \setminus \{n_1\}} \mathrm{BR}(n, t; i) \cdot \omega(e_{m_1,n}) + \mathrm{BR}(n_2, t; i) \cdot \omega(e_{m_2,n_2})$$

(by (5.7))

$$= \delta(m_1, t) + \sum_{n \in N^+(m_2)} \mathrm{BR}(n, t; i) \cdot \omega(e_{m_2,n})$$

(by (5.7) and equiv. prop. (5.3))

$$= \delta(m_2, t) + \sum_{n \in N^+(m_2)} \mathrm{BR}(n, t; i) \cdot \omega(e_{m_2,n}) = \mathrm{BR}(m_2, t; i)$$

(by equiv. prop. (5.5))

Proof of requirement 3. Let m_1, m_2 be two active components such that:

$$m_1 \equiv_e m_2 \; (\exists! n' \in N^+(m_1) = N^+(m_2) \; : \; \omega(e_{m_1,n'}) > \omega(e_{m_2,n'})) \tag{5.8}$$

Then the following holds for any iteration $i > 0$:

$$\mathrm{BR}(m_1, t; i) > \delta(m_1, t) + \sum_{n \in N^+(m_1) \setminus \{n'\}} \mathrm{BR}(n, t; i) \cdot \omega(e_{m_1,n}) + \mathrm{BR}(n', t; i) \cdot \omega(e_{m_2,n'})$$

(by (5.8))

$$= \delta(m_1, t) + \sum_{n \in N^+(m_2)} \mathrm{BR}(n, t; i) \cdot \omega(e_{m_2,n}) \qquad \text{(by equiv. prop. (5.1))}$$

$$= \delta(m_2, t) + \sum_{n \in N^+(m_2)} \mathrm{BR}(n, t; i) \cdot \omega(e_{m_2,n}) = \mathrm{BR}(m_2, t; i)$$

(by equiv. prop. (5.5))

5.3 Sensitivity Assessment Implementation

We implement the proposed approach as a set of Python scripts. We leverage Python's `networkx` library to manipulate the graph, i.e., computing BACRank, normalizing the scores, creating visualizations, etc. The main function, in charge of computing BACRank, is shown in code Listing 5.1. This function is called as many times as needed until the desired convergence level ϵ is reached. In our implementation, we define an $\epsilon < 10^{-6}$ to stop the components' sensitivity estimation at any given time t.

The components' graph is stored in a Neo4j database [100]. Neo4j is a noSQL database engine specialized in graph data structures. Specifically, we use Neo4j

version 4.1.2. The Neo4j database is queried (read and updated) from our Python scripts using the official driver.[1] We use Neo4j to store each node's BIA score (based on their related business processes), calendars (of both, building services and related business processes), and their final sensitivity score. Moreover, the weight of the edges is also stored in the database.

```
def bac_rank(G):
  # Variable to keep the max score among all the nodes
  current_max = 0

  # New BACRank iteration for each node
  for s in G.nodes():
    # A node's base score is its BIA
    s_score = G.node[s]['BIA']

    # For each of its destination nodes, add their score * edge weight
    for d in G[s]:
      s_score += float(G.node[d]['score'] * G[s][d]['weight'])

    current_max = s_score if s_score > current_max else current_max
    # Save the new score in temp variable
    # Score updates happen in the next loop because current
    #  scores are computed based on the previous iteration scores
    G.node[s]['tmp'] = s_score

  # Scores normalization in range [0,1]
  for s in G.nodes():
    G.node[s]['score'] = G.node[s]['tmp']
    if current_max > 0:
      G.node[s]['score'] = float(G.node[s]['score'] / current_max)

  return G
```

Listing 5.1: Python code that computes the BACRank centrality metric on a smart building components' graph stored in Neo4j.

Our implementation uses *software modules* as components (nodes of our graph). These software modules run in BACnet controllers and comprise multiple individual data points (e.g., sensors, setpoints, and actuators). Our BACnet controllers are manufactured by Priva, whose programming interface provides pictures of the information flow relationships among software modules. This eases the extraction of the graph under study. The information flow pictures, however, are not exportable to other file formats and had to be manually added to the Neo4j database. A concrete example of such pictures is shown in Figure 5.1, where each box represents a software module and the arrows represent the direction of information flows.

[1] `https://neo4j.com/docs/api/python-driver/current/`.

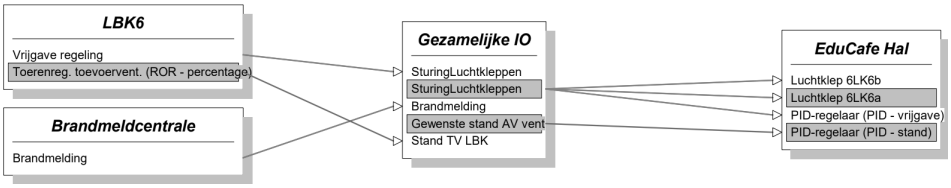


Figure 5.1: Priva programming interface screenshot. Centered on one particular software module (*Gezamelijke IO*), it shows two other software modules that send information to it (*LBK6* and *Brandmeldcentrale*) and one software module that receives information from it (*EduCafe Hal*).

Table 5.2: Business Processes (BPs) and their corresponding BIA scores.

Nº	Business Process	Score
1	Research	0,63
2	Application / admission	0,27
3	Accounting	0,60
4	Technical support	0,43
5	Courses and others (periodic)	0,73
6	Trainings and others (non-periodic)	0,70
7	Introduction week	0,60
8	Administrative support	0,47
9	Education advisory	0,53
10	Marketing & communication	0,43
11	Catering	1,00
12	Student associations	0,50

5.4 Evaluation of the Proposed Sensitivity Assessment

Environment Description. We executed our proposed approach on a 5-story smart building at the University of Twente, hosting 375 employees in 252 rooms. The local building manager provided assistance with the required expert-based information. We identified 12 business processes that take place in this building, some of them running only at specific periods of the year. The BIA revealed the corresponding sensitivity scores as presented in Table 5.2.

The main services of the smart building are implemented using the BACnet protocol [15]. The BACnet system controls the heating, ventilation, cooling, and lighting services. Other building services, such as physical access control, are

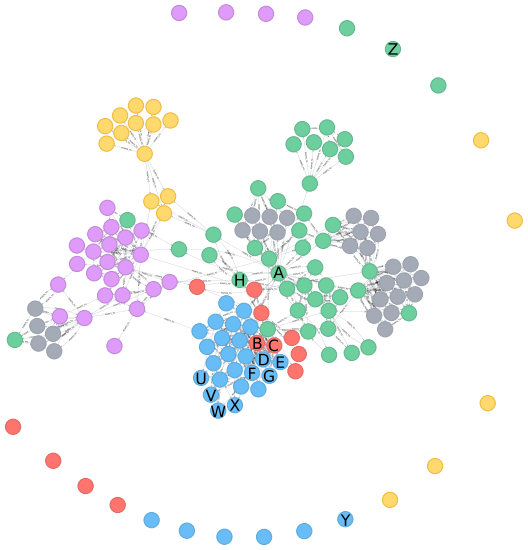


Figure 5.2: Real software modules dependency graph.

implemented using different protocols and tools we did not have access to. Here we consider only the building services implemented using the BACnet system, however, it is worth noting that our proposed approach is protocol-agnostic.

The assets considered in our evaluation are software modules. The BACnet system is comprised of 160 software modules running in 5 multi-purpose controllers (BACnet profile B-BC) and 28 application-specific controllers (BACnet profile B-ASC). Figure 5.2 shows the software modules dependency graph where 22 vertices are isolated and the remaining 138 are connected in the main subgraph. Vertex colors indicate the device they run in. Red modules are part of the heating controller, whereas blue modules are part of the cooling controller. The lighting system is controlled by the yellow modules. Green modules run in a controller in charge of multiple services throughout the building (ventilation, heating, and cooling). Purple modules also implement multiple services but in one specific location of the building. Finally, each gray module represents one application-specific controller running exactly one software module (thermostats).

Results. We used a weekly time resolution in our evaluation, based on the activity of the business processes and software modules analyzed. For this reason, a new sensitivity estimation is computed every week of the year to take into account business processes that start or stop execution. Additionally, it is important to take into consideration only the software modules that implement services needed during

each particular week (e.g., due seasonal climate conditions). The weekly sensitivity estimation can be computed in advance given that the required information is available. For each week, BACRank is executed a number of iterations until the scores' convergence is reached. After the tenth iteration, on average, the difference between two consecutive scores (ϵ) is smaller than 0.0006. After 60 iterations $\epsilon = 0$. To run our evaluation we defined an $\epsilon < 10^{-6}$, which is achieved after only 20 iterations. Figure 5.3 shows the quick convergence of BACRank on our real software modules graph.

The weekly sensitivity estimation of software modules allows us to rank them and analyze the rank changes along the time. Figure 5.4a illustrates the 52 rankings obtained throughout the year, where each colored line represents a software module (following the same color scheme used in Figure 5.2). The vertical axis represents the ranking position where top ranked modules start at position 1. Modules with identical scores in Figure 5.4a are arbitrarily assigned a slot next to their analogous. Figure 5.4b on the other hand, shows the actual BACRank score for each module. The impression of having fewer lines in Figure 5.4b than in Figure 5.4a is due to multiple overlapping lines (i.e., modules with identical score).

BACRank successfully identifies software modules that are part of relevant business processes, and are required by other relevant modules in the infrastructure. Throughout the experiments, module "Multi-purpose Substation" (vertex A in Figure 5.2) was considered the most sensitive because, among others, it supports the most sensitive business process according to the BIA ($\boldsymbol{BP}_{11}$) *and* it provides information required by 19 other important modules in 7 different devices. Vertex A is depicted as an horizontal green line at the top of Figure 5.4a and Figure 5.4b.

At the bottom of the ranking there is a set of approximately 30 modules consistently low ranked. Some of them, starting from the last one—"AirExtraction"—and ascending with "Electricitymeter Experiments", "CoolSection 1_Log", "CoolSection 2_Log", "CoolSection 3_Log", and "CoolSection 4_Log", are shown in Figure 5.2 as Z, Y, X, W, V, and U, respectively. In general, there are two main aspects that justify the poor scoring performance of smart building components. First, vertices whose out-degree is 0 are likely to be low ranked because an important source of BACRank score comes from other vertices that depend on it. In this case, the final BACRank score is derived exclusively from the *initial score* which is, in turn, based on the related business processes sensitivity. Second, if there are no related business processes, as in the case of safety oriented modules; or the module's participation in the business processes is marginal, then the module in question will get a low BACRank score.

Time-*independent* software modules are typically ranked in similar positions throughout the year. Figure 5.4a shows that lighting and thermostat modules (yel-

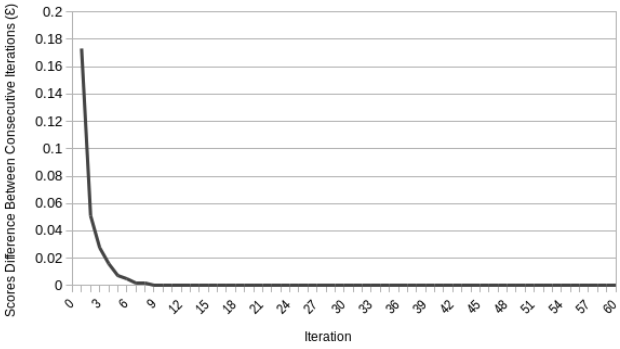


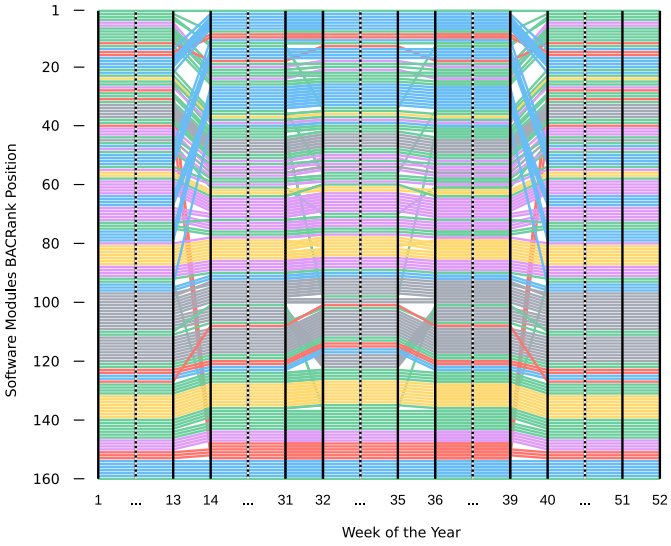
Figure 5.3: BACRank Scores Convergence.

low and gray lines) are good examples of time-independent modules. Their minor shifts up and down respond mostly to score changes in other modules rather than their own scores.

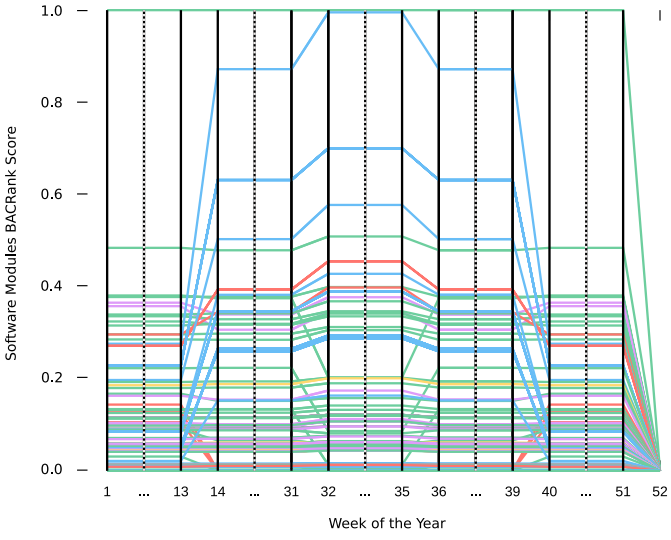
Time-*dependent* modules, on the other hand, are visible in Figure 5.4a between weeks 14 and 39 of the year. This represents roughly the period between April and September, which is warmer than the range between October and March, taking into account the geographical location of the building. Figure 5.4a shows that most cooling modules increase their sensitivity score in this period whereas some heating modules suffer a substantial decrease (blue vs. red lines). Heating modules that remain similar or even increase their rank in the April–September period are benefited from neighboring cooling modules that got their rank increased. For example, heating modules "Radiatorgroup South" and "Radiatorgroup North" in positions 15 and 16 in Weeks 1–13, climbed to positions 9 and 10 in Weeks 14–39. These two modules are labeled as B and C in Figure 5.2, which shows their proximity to cooling modules. Exactly 4 cooling modules—D, E, F, G—depend on B and C.

Weeks 32–35 of the year (August) are part of the organization's summer break in which some of the business processes stop execution. Student related processes (BP_5 and BP_{12}) do not run in this period and, therefore, the related software modules lower their ranking positions. Module "AHU WestLectRoom", for example, decreases its rank from position 13 in week 31 to position 34 in week 32. The main reason for its descend is its support to the halted BP_5. This module is labeled with the letter H in Figure 5.2.

In weeks 40–51 all software modules rank in the same order they ranked at the start of the year, due to identical conditions in terms of business processes running



(a) Rank variation in time.



(b) Score variation in time.

Figure 5.4: Software modules impact variation in time. Plots only legible in color.

and software modules needed. Week 40 marks the start of the winter period in which cooling modules decrease their relevance in favor of heating modules as shown in Figure 5.4a and Figure 5.4b.

Finally, in week 52 the organization is closed and no business processes are running in this building. As explained before, Figure 5.4a will simply assign an arbitrary order to equally ranked modules, whereas Figure 5.4b shows that all the modules get a score of 0 which means that from a *sensitivity* viewpoint all modules are "equally unimportant".

5.5 Related Work

The prioritization of security weaknesses has been studied mostly in the IT domain. The Common Vulnerability Scoring System (CVSS) is a de-facto standard specifically tailored for IT vulnerabilities, whereas the Common Weakness Scoring System (CWSS) is similar to the CVSS but for IT security weaknesses [47, 95]. However, neither of them considers the intrinsic features of cyber-physical systems (e.g., the impact of cyber-physical attacks), which makes them unsuitable for smart buildings and other OT systems.

Still in the IT domain, other strategies have been proposed in literature to prioritize software vulnerabilities. For instance, in [97] the authors propose a prioritization method based on an estimation of the time that an attacker would spend in the exploitation process. They do so taking into account, among other aspects, the network topology of the targeted system, which allows to estimate the attacker's lateral movement time required to reach the vulnerable computer. Another work considers social media discussions to estimate (1) if and when a CVE will be exploited; and (2) its CVSS severity score and attributes [32]. Additionally, several proprietary products have developed their own methods to prioritize vulnerabilities [109, 110, 129], possibly combining other approaches as in [9].

On the other hand, the prioritization of security weaknesses in the OT domain is still an understudied topic. One way to do so, is to estimate the potential effect of exploited weaknesses, which has been mostly analyzed for ICSs [51, 26, 102]. These works are based on the observation of cause-and-effect relations between data points representing sensors and actuators. The observations are taken from simulated environments where changes are induced to log the corresponding effects. Knowing the limits of the physical process (e.g., what is the threshold before the power plant explodes?), data points with the higher potential to reach those limits are prioritized. The only work that is focused on smart buildings prioritizes component

categories rather than individual assets [141]. They automatically analyze work-orders describing building's routine and maintenance operations. Based on the information recorded in the work-orders, such as location, problem description, and priority, they rank equipment categories like "fans", "valves", "pumps", and others.

According to ISO and NIST risk-based activities and controls, we argue that the prioritization of security weaknesses in smart buildings must be based on the risk that they represent for the affected organization [21, 69, 127]. However, an accurate estimation of risk requires the analysis of several other aspects [73]. One of such aspects is the *sensitivity* of assets, which is the particular focus of this chapter.

STRIDE[2] and DREAD[3] have been used to identify and score CPS threats, respectively [125, 145]. These approaches leverage data flow diagrams similar to the graphs used in this chapter. However, it is worth recalling that a comprehensive risk assessment involves the analysis of weaknesses/vulnerabilities, assets, and threats [73]. Unlike STRIDE and DREAD, which focus on threats, the proposed approach a aims at estimating one particular property of assets, namely, their *sensitivity*.

5.6 Discussion

The prioritization of security weaknesses in smart buildings must be based on the *risk* that each of them poses for the affected organization. To estimate the risk, however, is not an easy task. Literature considers at least 29 factors that influence the *probability* and *impact* of loss events, i.e., the main two components of *risk*. As a consequence, an accurate estimation of all those factors might be costly yet necessary for critical organizations. Our observation is that the information needed to estimate one of the crucial factors might already be available and, therefore, can be used as part of the overall risk estimation of security weaknesses. The risk factor that we refer to is *sensitivity*.

The sensitivity of smart building components can be derived from the organization's business impact analysis (BIA). The goal of the BIA is to score the sensitivity of business processes for the organization and is not related to any technologies by itself. However, the fact that the technologies in place support—or even enable—business processes, creates an indirect link between the BIA scores and the corresponding technological assets. We leverage this link to set up the

[2]Mnemonic for Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, and Elevation of privilege.

[3]Mnemonic for Damage, Reproducibility, Exploitability, Affected users, and Discoverability.

initial scores of smart building components.

The BIA is an expensive step in the implementation of a business continuity management system (BCMS). It involves the identification of business processes, the development of questionnaires, interviews with stake holders, and the post-processing and analysis of the questionnaires' answers. Although the BIA is very likely available in critical and mature organizations, this might not be the case for other organizations. Depending on the nature and resources of the organization hosted in a smart building, it might be difficult to implement our proposed approach.

The *sensitivity* alone is insufficient to prioritize security weaknesses. Although it is a crucial factor to estimate the potential impact of exploited weaknesses, there are other aspects to consider. For example, the assets' *criticality* (impact on the organization's productivity) and *cost* (intrinsic value in case a replacement is needed) should also be taken into account [73]. Moreover, regarding the probability of loss events, aspects such as *contact* (the probable frequency, within a given time frame, that a threat agent will come into contact with an asset) and *action* (the probability that a threat agent will act against an asset once contact occurs) are also crucial [73]. The overall prioritization of security weaknesses must be based on potential high-impact and high-probability loss events.

Besides the initial sensitivity score that is assigned to smart building assets, we propose to adjust these values according to the technical implementation of the system. There are 3 of these technical aspects where an expert judgment is required, namely, the dependency strength between components, the support that building services offer to business processes, and the duty cycle of building services (i.e., building services activity throughout time). Although an ideal estimation of risk is fully objective, the involvement of a domain expert is often needed, which introduces subjectivity in the process. This is a common problem even in well-known industry standards such as CVSS [75].

The automation of our sensitivity assessment can only be partially achieved. The information gathering phase, that involves the BIA and the domain expert assessments, is a manual process. However, thanks to our graph-based modeling of smart building components, the adjustment of the initial scores is done through an automatable graph centrality metric. Given that the business and technical conditions do not change, the estimation of sensitivity can be fully automated throughout time, as we showed by computing in advance the sensitivity of software modules 52 times per year (i.e., on a weekly basis).

Although the sensitivity is just one component of the overall risk estimation needed to prioritize security weaknesses, it might be, on its own, a good indicator to perform other cybersecurity tasks. For instance, to schedule the appropriate time to apply updates or patches on high demand systems. As we showed in our

evaluation, the sensitivity of software modules changes throughout the time based on the activity of business processes and the need of building services. Thus, the updates of these modules can be planned ahead during the period when their sensitivity is at a minimum.

5.7 Conclusion

In Chapter 3 and Chapter 4 we proposed two methods to identify security weaknesses in smart building applications and configurations, respectively. The identification of security weaknesses is crucial in any vulnerability management process because all subsequent activities revolve around the weaknesses initially found. The next step after the identification of security weaknesses is their *prioritization*. The *risk* that security weaknesses pose for organizations is the recommended criteria to prioritize them according to the consensus reflected in international standards and best practices guides. In line with this view, the *sensitivity* is one of the main factors to consider in the overall estimation of security risks.

In this chapter, we addressed **Research Question 3**: How to (semi-) automate a sensitivity assessment for smart building components? To answer this question, our proposed approach takes into account business and technical aspects from diverse information sources. More concretely, our proposed method to measure the sensitivity of smart building components considers their support to business processes and neighboring components.

In summary, our proposed method is modeled as a graph centrality measure. The reason behind this decision is that smart building components constitute a dependency graph and that such dependencies might lead to chain reactions. Thus, the components' dependencies must be a core aspect to estimate their sensitivity. To make explicit the role of such dependencies, we formally defined the general requirements that a centrality measure must satisfy to compute scores that reflect the components' sensitivity. Finally, we developed one instance of such a centrality measure, which we called *BACRank* and formally proved that it satisfies the defined general requirements. We implemented our proposed approach and performed an evaluation in a real smart building. In this setting, we found out the sensitivity of smart building components throughout a year considering the time dimension in both, building services and business processes. The evaluation showed that BACRank successfully prioritizes the most *sensitive* components (software modules) for the organization.

The match between the criteria typically used in business impact analyses and the criteria needed to estimate the assets' *sensitivity*, allows us to reuse the business

impact analysis (BIA) results as part of the prioritization of security weaknesses. Moreover, modeling the final sensitivity estimation as a graph centrality measure enables the automation of this task: a transversal goal of all the approaches proposed in this thesis. The business and technical criteria behind the results produced by our approach were ratified by the local smart building administrators.

Beyond the technical aspects, often emphasized in OT security research, we realized that the business aspects play a major role in the assessment of components' sensitivity. For this reason, in Chapter 6 we deepen our analysis on the sensitivity assessment proposed in this chapter by creating a smart building testbed capable of adopting the configuration of diverse business environments. We then use it to estimate the sensitivity of the same infrastructure components from the viewpoint of different organizations.

Chapter 6

An Extended Multi-Context Evaluation of BACRank

In Chapter 5 we introduced a method to estimate the sensitivity of smart building components. The *sensitivity* is the harm that a loss event might cause in terms of, e.g., legal/regulatory consequences, reputational damage, fines, and others [73]. Moreover, it is an important factor of the overall risk assessment needed to prioritize weaknesses for remediation. In the method proposed in Chapter 5, we modeled the smart building infrastructure as a graph data structure where the nodes represent the components of interest that will be assessed, and the edges represent the functional dependencies between them. To perform the assessment, first we set an initial score to each component in the graph based on their support to sensitive business processes, i.e., business processes prone to cause *sensitivity*-related consequences if halted. Then, we refined such scores by means of a graph centrality measure that we called BACRank, which considers the chain reactions that are triggered after the availability or integrity of a component is compromised.

Our evaluation of BACRank in a real smart building led us to realize the importance of the business context to prioritize weaknesses for later remediation. Thus, the goal of this chapter is to explore BACRank's sensitivity quantification upon different contexts but on the same smart building infrastructure. Although it is possible to apply BACRank on any real environment (e.g., hospitals, airports, data centers, etc.), the smart building infrastructure is different for all of them, even for those buildings of the same organization type. To analyze the sensitivity score variations of a fixed infrastructure under diverse business contexts, we create a testbed comprised of four basic building services common to several organization types.

Table 6.1: Required environmental conditions for diverse business process locations.

Business Process	Business Process Location	Illumination (lux)	Ventilation (CO_2 ppm)	Temperature (°C)
Surgeries	Operating Room	[500-600]	≤ 770	[20-24]
Teaching	Lecture Hall	[300-500]	≤ 1400	[20-27]
Server hosting	Data Center	[50-100]	-	[18-27]
Blood tests	Laboratory	[750-1200]	≤ 1400	[20-27]
Physical conditioning	Fitness Gym	[200-300]	≤ 880	[20-22]

The testbed described in this chapter implements ventilation, heating, cooling, and illumination services 6.3. These services are extensively used in smart buildings hosting diverse organizations. Each organization, however, has different requirements for these building services. For instance, the illumination requirements in a hospital operating room are significantly different from the illumination requirements in a data center. This is, precisely, the contrast that we aim to explore through BACRank's sensitivity assessment.

6.1 Preliminaries

Building services play an important role in organizations, supporting the execution of their business processes [4]. However, the configuration of smart building services is different depending on the supported business processes. Each business process location has a set of desired or, in many cases, *required* environmental conditions that it must comply with in order to fit its purpose. Ventilation, temperature, illumination, among other conditions, are specified for different locations in diverse documents such as standards, regulations, and best practices guides. Thus, setpoints, thresholds, and control algorithms change depending on the particular setting. Examples of regulated environmental conditions are shown in Table 6.1, taken from [16, 17, 18, 19, 111, 118]. From the examples in Table 6.1, it is worth noting that there are no ventilation requirements for data centers due to the rare presence of people in such locations.

The recommended environmental conditions for different business process locations are stipulated in domain-specific or service-specific documents. Among the domain-specific documents are ANSI/TIA-569-C-1 ("Revised Temperature and Humidity Requirements for Telecommunications Spaces") [19], ANSI/ASHRAE/ASHE standard 170-2017 ("Ventilation of Health Care Facilities") [17], and "Health/Fit-

ness Facility Standards and Guidelines" by the American College of Sports Medicine [118]. On the other hand, among the service-specific documents are "The IESNA lighting handbook: reference & application" [111] and ANSI/ASHRAE standard 62.1-2016 ("Ventilation for Acceptable Indoor Air Quality") [16]. Table 6.1 shows just a few business processes and their recommended environmental conditions. Our goal is simply to illustrate that business process locations are generally linked to particular environmental conditions, rather than to provide an exhaustive list.

6.2 Testbed for Sensitivity Assessment

Most security testbeds in the smart buildings domain focus on the demonstration of defensive tools. Such demonstrations typically compare the set of attacks that the tools can handle with the set of attacks that it cannot. These attack-based demonstrations draw all the attention to the technicalities of the attack without any business context in place. The smart building testbed described in this chapter adds the context needed to perform a sensitivity assessment, alongside the technical features commonly found in other testbeds.

There are two main goals for our tested:

1. To use the testbed's infrastructure as a reference to estimate the sensitivity of its components under different business contexts.

2. To execute attacks that clearly show the conditions in which organizations are actually affected due to their intrinsic building service requirements.

Although the component's sensitivity is independent of any attacks, it allows to demonstrate their impact in a tangible way. In this section, we describe the development process of our testbed, covering its requirements, design, and implementation.

6.2.1 Requirements

Scenarios. Our testbed must serve as a platform to assess the sensitivity of smart building components in different business contexts. We define *scenario* as the combination of a business process location (in previous sections regarded as *the context*) and its supporting building services. Our observation, as can be derived from Table 6.1, is that a subset of core building services can abstractly represent different business process locations. Based on this observation, the requirement for our testbed is to implement automated illumination, ventilation, and temperature control, so it can reproduce the environmental conditions of diverse locations such

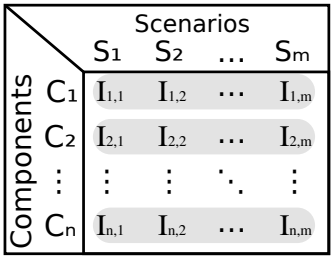


Figure 6.1: Requirements of our testbed: (1) Reproducibility of diverse scenarios modeled through building services; (2) Smart building infrastructure comprised of multiple components; and (3) A measure of the components' sensitivity on multiple scenarios.

as, but not limited to, those listed in Table 6.1. To emulate a particular context in our testbed, a software tool is needed to reconfigure the testbed's building services according to the requirements of that context. Only one scenario at a time can be configured in the testbed.

Smart Building Components. Our tested must be built using real smart building infrastructure. This infrastructure will be comprised of diverse physical and abstract components at different granularity levels (e.g., building services, software modules, data points). Any of these component types can be used later on to estimate their sensitivity (the first goal of our testbed). Also, a concrete implementation of a smart building testbed would allow us to execute real attacks and link the estimated sensitivity of the targeted component with the physical effects observed after the attack (the second goal of our testbed).

Sensitivity Assessment. We use the method proposed in Chapter 5 to estimate the sensitivity of smart building components in our testbed. This approach is based on the BACRank centrality measure. Using this method, we would be able to compare the sensitivity of the same smart building components under different contexts.

To summarize our requirements, Figure 6.1 shows the relation between components, scenarios, and the sensitivity assessment, where $I_{i,j}$ is the sensitivity estimated for component i under scenario j.

6.2.2 Design

Our testbed integrates illumination, ventilation, heating, and cooling as building services. Figure 6.2 depicts these services as implementations of an abstract *BuildingService*. Whereas each *Scenario* uses one or more *BuildingService*(s), a *BuildingService* might not necessarily model a *Scenario*. This is how smart building

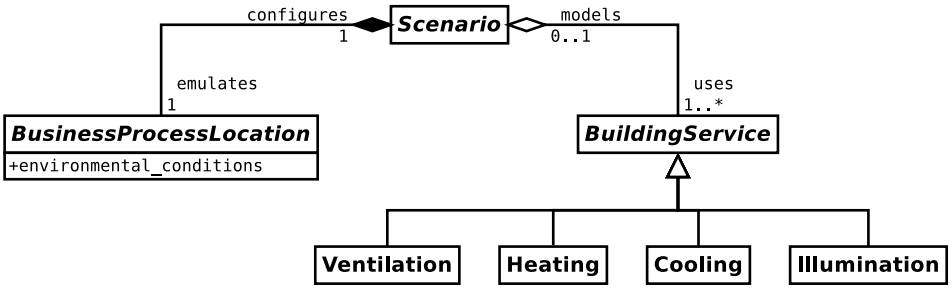


Figure 6.2: UML diagram depicting the high level design of our testbed.

testbeds have been built in the past. It is only by configuring the environmental conditions of a *BusinessProcessLocation* that the overall *Scenario* is embodied in the testbed. From a design perspective, we do not limit the business process locations that can be emulated in our testbed.

The control algorithms differ per building service. Whereas some services are activated upon specific time conditions, others require feedback from the environment. The former is known as *open control loop* (see Alg. 1) and the latter as *closed control loop* (see Alg. 2). In our testbed, the illumination service is handled by an open control loop, whereas the ventilation and temperature control use closed control loops.

Algorithm 1 Simplified open control loop.

```
while True do
  if time_for_action then
    take_action()
  else
    stop_action()
  end if
end while
```

6.2.3 Implementation

Hardware. Smart buildings comprise diverse components in a 3-layered hierarchical arrangement. At the bottom are sensors and actuators commonly referred to as *field devices*. In the middle, embedded computers in charge of taking inputs from sensors and sending output signals to actuators make up the *control layer*. On top, there is a *management layer* which provides unified control and monitoring to

Algorithm 2 Simplified closed control loop.

```
while True do
  if controlled_var > upper_limit then
    decrease_controlled_var()
  else if controlled_var < lower_limit then
    increase_controlled_var()
  end if
end while
```

smart building administrators.

In our testbed, we use the BACnet communication protocol at the control and management layers [15]. Although at these layers we use software and hardware commonly used in real smart buildings, at the field level we use smaller actuators than those used in real buildings. This is due to our down-scaled version of building rooms.

We built two physical modules that represent real building rooms. The first module is a *mechanical room* that contains heating and cooling hardware that emulates a building's boiler and chiller, respectively. The second module is a generic *building room* that requires heating and cooling services from the first module. Moreover, it has a thermostat, illumination, and ventilation hardware. The thermostat contains, among others, temperature and CO_2 sensors (inputs) and relays to interact with the actuators (outputs). Both modules are physically connected to allow the heat/cold transfer. Figure 6.3 shows a picture of both physical modules.[1]

Since the illumination service must be adapted to different lighting requirements, it is controlled by an *analog output* that regulates the light intensity. The analog output provides a maximum of 20 mA at [0-12] VDC, which is too low to feed the high power LEDs installed in the building room. A customized electronic circuit was designed to dim the lights according to the driving analog output. The other actuators are controlled using *binary outputs* linked to the relays in the thermostat controller. Thus, avoiding the need for additional circuitry.

The cost of the testbed's hardware can be divided in three parts. First, the structural components, which includes the aluminum base, profiles, plexiglass, and others, have an approximate cost of $700 USD. Second, the BACnet specific hardware and software cost approximately $3.500 USD (see Table 6.2). Finally, other components including power supplies, actuators, and relays have an approximate cost of $500 USD. After considering outsourced services (e.g., plexiglass

[1]The 3D computer-aided designs, schematics of custom electronics, and bill of materials are published in `https://www.utwente.nl/en/eemcs/scs/downloads/2020_BACS_testbed/`.

Table 6.2: BACnet components used in our testbed.

Vendor	Product	BACnet Profile	Approximate Cost
KMC	BAC-5050	Router	$1.000 USD
KMC	FlexStat BAC-131136CEW	B-ASC	$1.000 USD
MBS	BACeye version 2.1.0.15	B-OWS	$500 USD
Janitza	UMG 604-PRO	B-SA	$1.000 USD

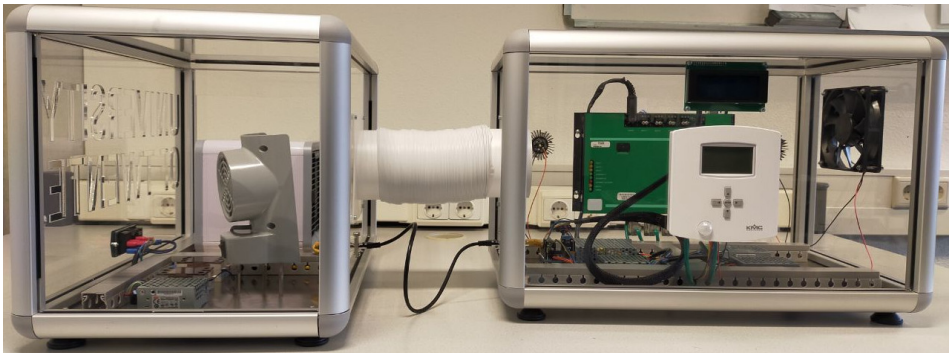


Figure 6.3: Testbed modules. The building room (on the right) is physically connected to the mechanical room (on the left) to allow air flow.

laser cutting), the overall cost of the physical components of the testbed is about $5.000 USD. We consider this as reasonable costs for a small testbed and it should allow other research groups to replicate our testbed.

Communication. As stated above, the chosen smart building communication protocol is BACnet [15]. The underlying protocols include UDP, IP, ICMP, Ethernet, and MS/TP. Moreover, there is a modem connection to the telephone network. A network diagram of our testbed is shown in Figure 6.4.

The IP network implements a star topology. The core switch has been configured with a mirroring port to collect all the network traffic exchanged in the system.

Software. The most important software applications used in our testbed are BACeye 2.1.0.15 and bacnet-stack 0.8.6 [74]. Using bacnet-stack we implement a Linux-based operator work station (OWS 1 in Figure 6.4). It runs a custom application developed to quickly reconfigure the testbed to meet the environmental requirements of predefined business process locations. Additionally, BACeye runs on a Windows-based OWS (OWS 2 in Figure 6.4).

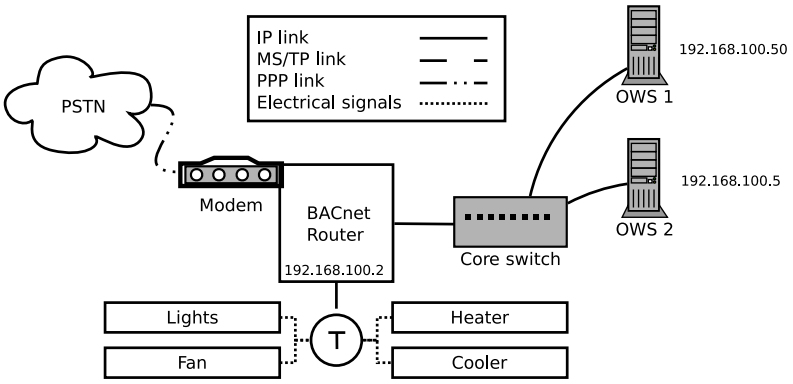


Figure 6.4: Network topology (including electrical signals to actuators).

The firmware version of the FlexStat controller is R2.1.0.18. The BAC-5050 router runs firmware build R1.8.0.1.

6.3 Testbed-based Evaluation of BACRank

As it was mentioned in Chapter 5, the granularity of the components whose sensitivity will be assessed depends on the needs of the organization. To simplify our discussion, in this chapter we use *building services* as high level components to be assessed. This decision reduces the graph size to only three nodes: illumination, ventilation, and temperature control; this last one includes the heating and cooling services.

To achieve our goal of assessing the sensitivity of the same smart building components under different scenarios, we pick three business process locations from Table 6.1, namely the operating room, lecture hall, and data center. These locations are chosen due to their distinctive building service requirements.

To achieve our goal of showing the diverse conditions in which organizations are actually affected by cyber attacks, we target the illumination (I), ventilation (V), and temperature control (T) services implemented in our testbed. The attacks used in our experiments are based on a technique called *Unauthorized Command Message* in which "[a]dversaries may send unauthorized command messages to instruct control systems devices to perform actions outside their expected functionality for process control.".[2] More specifically, we perform a data manipulation attack leveraging BACnet's WriteProperty service [76]. This attack consists of a syntactically valid

[2]https://collaborate.mitre.org/attackics/index.php/Technique/T855

BACnet message that changes a property in a BACnet object.

The sensitivity of smart building components can be estimated in advance by understanding the requirements of business processes on building services. Building services that, if halted, affect business processes in terms of e.g., reputational damage, legal consequences, and fines, are deemed of greater sensitivity that other services. The sensitivity levels assigned to building services are technically-backed choices made by domain experts. In what follows, we present a short description of such technically-backed choices for each location assessed. A summary is presented in Table 6.3.

Operating Room. The World Health Organization deems illumination as "one of the major nonstructural elements in a hospital" [140]. While most people would agree that all environmental conditions in operating rooms are important, the severity and immediacy of an attack on the illumination service are key factors to consider it as the highest priority service, above the ventilation and temperature control, both considered of medium impact.

Lecture Hall. The concern for air quality is common in densely occupied indoor spaces [16]. A high concentration of CO_2 (e.g., $\geq$ 1400 ppm) might lead to illness symptoms such as headaches and dizziness. Moreover, the ventilation is considered a high priority service in lecture halls because it has been shown that improving the air quality increases the students performance [42]. Although illumination and temperature are also relevant, they have been scored as medium impact services.

Data Center. Data centers are extremely sensitive to temperature [18]. Whereas low temperatures increase the chances of electrostatic discharges, high temperatures might damage the servers' hardware, or trigger safety mechanisms to automatically power them off. For these reasons, temperature control is by far considered the most sensitive building service in a data centers. Data centers do not have ventilation requirements (see Table 6.1) mainly because servers do not produce CO_2 *in-situ*. Finally, the illumination service is primarily used to enable video surveillance. For these reasons, the ventilation and illumination are deemed as low impact services.

6.3.1 Sensitivity Assessment of Smart Building Components using BACRank

In this section we follow the methodology described in Chapter 5 to estimate the sensitivity of smart building services. To do so, the building services must be modeled as nodes in a graph data structure, i.e., illumination, ventilation, and temperature control. The edges of the graph represent the information flows according to the specific smart building implementation. In our testbed, the illumination and

Table 6.3: Sensitivity level of building services on different contexts without consideration of the technical implementation. The high sensitivity services per location are highlighted in bold font.

Attack	Operating Room	Lecture Hall	Data Center
Illumination	**High**	Medium	Low
Ventilation	Medium	**High**	Low
Temperature	Medium	Medium	**High**

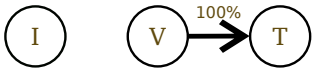


Figure 6.5: Graph used to compute the sensitivity of the illumination (I), ventilation (V), and temperature control (T).

ventilation services do not have external dependencies. The temperature control service, on the other hand, depends on the ventilation service to make the heat/cold transfer from the mechanical room to the building room. From an implementation point of view, the strength of such dependency is 100%. A graphical representation of the graph is shown in Figure. 6.5.

According to the method proposed in Chapter 5, the initial score given to each node (denoted as δ) at time t is defined as:

$$\delta(m,t) = \begin{cases} \max_{1 \leq i \leq n}(\beta(p_i) \cdot \gamma(s_j, p_i)) & \text{if time}(p_i,t) = \text{time}(m,t) = 1, \\ 0 & \text{otherwise,} \end{cases}$$

where function β returns the business impact analysis (BIA) score of business process p_i, out of n business processes in the organization. Moreover, function γ encodes the support of building service s_j (of which m is part) onto business process p_i. Finally, *time* is a binary function that is overloaded to take as input a business process or a smart building component. $time(p_i, t) = 1$ means that business process p_i is runs at time t, and $time(p_i, t) = 0$ means that business process p_i does *not* run at time t. Similarly, $time(m, t) = 1$ means that component m is needed at time t, and $time(m, t) = 0$ means that component m is *not* needed at time t.

Three components of the δ function are simplified in this evaluation with respect to the evaluation in Chapter 5:

Assets. In this chapter we use building services as coarse grained components to assess. This decision simplifies our discussion while preserving all the properties of the original method.

Business processes. Since we consider only one business process per organization (i.e., hospital→operating room, hosting company→data center, and university→lecture hall), subscripts are not needed for business processes. Furthermore, since the BIA scores of different organizations are not comparable, we assume each business process to have identical values for β, i.e., $\beta(p) = 1$.

Time. We assume that all building services and business processes are needed/active at the time of the assessment.

These changes lead to a simplified version of the original function:

$$\delta(s_j) = \gamma(s_j, p).$$

Thus, it is clear that the initial score of each building service s_j is determined by its support to business process p. Table 6.4 specifies the initial scores of the building services implemented in our testbed. Moreover, it contains the final sensitivity score of each service.

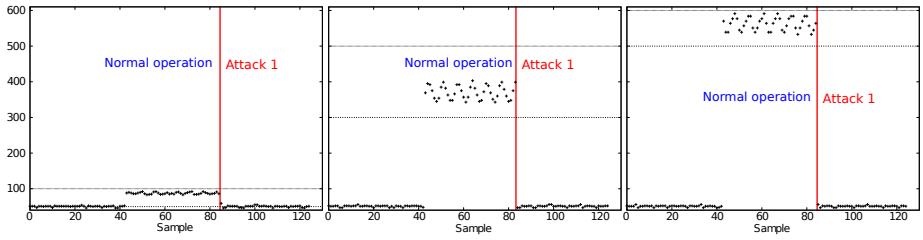
Table 6.4: Initial and final sensitivity scores of the implemented building services. The high sensitivity services from Table 6.3 are also highlighted here in bold font.

Location	$\delta(I)$	$\delta(V)$	$\delta(T)$	BACRank(I)	BACRank(V)	BACRank(T)
Operating Room	1.0	0.5	0.5	**1.0**	1.0	0.5
Lecture Hall	0.5	1.0	0.5	0.3	**1.0**	0.3
Data Center	0.1	0.1	1.0	0.1	1.0	**1.0**

The BACRank-based sensitivity score is normalized in the range [0-1] per organization. By comparing the sensitivity scores from Table 6.4 with the expert-based assessments from Table 6.3, it is possible to observe a match in the most sensitive building services. That is not the case for other services previously considered of *medium* or *low* sensitivity. This is because in addition to business aspects, BACRank considers technical aspects omitted in the first assessment. The BACRank-based sensitivity assessment tends to increase the ventilation service score because other building service (i.e., temperature control) as a strong dependency on it.

6.3.2 Attacks on Smart Building Components

We configured our testbed according to the chosen scenarios to launch three attacks in each of them: turning the illumination off, stopping the ventilation service, and stopping the temperature control service. All attacks are executed against the thermostat FlexStat BAC-131136CEW (BACnet Application-Specific Controller).



(a) Data center (b) Lecture hall (c) Operating room

Figure 6.6: Illumination attack on different scenarios. The y-axis represents lux units for all scenarios. Data points collected during the experiment are shown using the "+" character. Dashed horizontal lines show the minimum and maximum required values.

The specifics of each attack are detailed in Table 6.5. These attacks do not exploit vulnerabilities particular to this device but leverage on the prevalent unauthenticated messages exchanged in the BACnet protocol.[3]

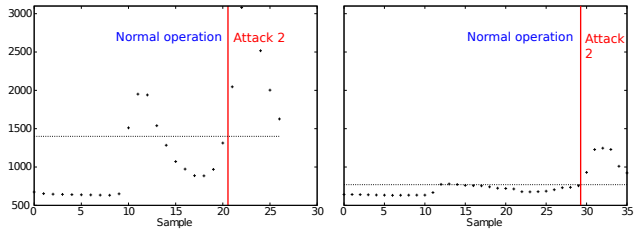
Table 6.5: Attack procedures against the building controller. Object types and instance numbers provided to ease the analysis of the corresponding pcap files.

No.	Attack	BACnet Service	Object Type	Object Instance	Written Value
1	Illumination	WriteProperty	Analog Output	5	0
2	Ventilation	WriteProperty	Binary Output	1	0
3	Temperature	WriteProperty	Binary Output	2,1	0

Illumination. The ambient light in the room where the testbed is located is measured in the range of [46, 52] lux. All the illumination experiments start with sensor readings in this range. After approximately 40 samples of ambient light, the illumination service is turned on at the intensity needed to meet the requirements of each specific scenario. Approximately 40 samples later the first attack is executed, which causes the sensor to report the ambient light intensity again, confirming thus the attack. Figure 6.6 shows the illumination samples collected during our experiments for each scenario.

Ventilation. Unlike lecture halls and operating rooms, data centers do not have CO_2 requirements (see Table 6.1). Since there are no consequences from the business perspective, we did not execute a ventilation attack on the data center scenario.

[3]Network captures of each attack are published in pcap format at `https://www.utwente.nl/en/eemcs/scs/downloads/2020_BACS_testbed/`.



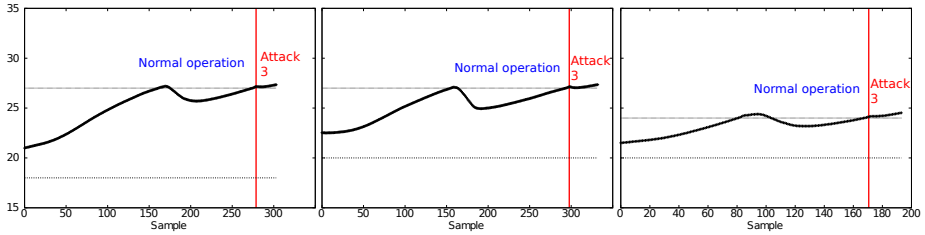
(a) Lecture hall (b) Operating room

Figure 6.7: Ventilation attack on the lecture hall and operating room scenarios. The data center scenario is excluded since it does not have specific CO_2 requirements. The y-axis represents CO_2 ppm units for both scenarios. Data points collected during the experiment are shown using the "+" character. The dashed horizontal line shows the maximum required values.

During the experiments, the ambient CO_2 level is in the range of [632, 674] ppm. For both ventilation attacks we take approximately 10 sensor readings before leaking CO_2 inside the testbed's building room. We use 16 gram cylinders of CO_2 commonly found in bike shops to inflate tires. As expected, the CO_2 values increase but are quickly returned to normal by the ventilation service. Once the CO_2 values are below the threshold, the fan is automatically deactivated which causes the CO_2 level to rise above the maximum limit again. The maximum limit violation triggers the ventilation service a second time. At this point, the attack is executed (i.e., the ventilation is turned off) which causes the CO_2 level to keep increasing. Finally, the CO_2 source depletes its content which drops the sensor readings again. Figure 6.7 shows the CO_2 level in our testbed during both experiments simulating the lecture hall and operating room locations.

Temperature Control. Each experiment starts by recording the ambient temperature of the testbed's *building room*. Afterwards, a source of heat is placed inside the room. For these experiments, three anti-spill aluminum bottles filled with boiling water are used as heat source.

As in the previous experiments, we first let the system react as it was designed to work. Later on, the third attack is executed which turns off both the cooler, physically located in the testbed's *mechanical room*, and the fan, located in the testbed's *building room*. Both devices are controlled from the thermostat by *binary output* object instances 2 and 1, respectively. Although the attack comprises two components, the goal is to increase the temperature regardless of the CO_2 level measured by the ventilation service. Figure 6.8 shows the temperature plots of our three experiments.



(a) Data center (b) Lecture hall (c) Operating room

Figure 6.8: Temperature attack on different scenarios. The y-axis represents degrees Celsius for all scenarios. Data points collected during the experiment are shown using the "+" character. Dashed horizontal lines show the minimum and maximum required values.

6.4 Related Work

Testbeds. The common objective of all OT security testbeds is to execute attacks and to evaluate defenses. On top of that, different goals are set which yield different testbed implementations. Not necessarily mutually exclusive, typical testbed goals are demonstration, education, and impact assessment [77, 131]. Demonstration testbeds are built to convince stakeholders of the applicability of both offensive and defensive research findings [131]. Education testbeds are skill-development platforms where students, researchers, and practitioners can learn hands-on [13, 78]. Finally, impact assessment testbeds use a variety of metrics to quantify the consequences of cyber attacks [10, 87, 107].

Smart building testbeds in particular, have overlooked the relevance of *sensitivity* analyses to mostly focus on the demonstration of security solutions [2, 44, 103]. To show the strengths and weaknesses of these tools, they launch attacks to exemplify success and failure cases. Although we acknowledge the illustrative value of such testbeds, the lack of context makes it difficult to recognize the attacks' potential impact in real-world scenarios and to realize the actual value of the proposed defenses. Our testbed addresses this limitation by incorporating *context* as part of its default operation.

Impact Metrics. Several Industrial Control System (ICS) testbeds have been built to study the *physical impact* of cyber attacks. For instance, a water treatment testbed is used in [133], where the impact is defined as the deviation in the pre-established pH level of the water. In [10], a water distribution testbed is presented where the impact of attacks is measured as the decrease in the supplied water with respect to the normal capacity of the system. Yet another example are the smart-grid testbeds presented in [87, 107], where the impact of attacks is measured in terms of voltage

instability, generation loss, and load shedding increment. Similar metrics have also been proposed for IoT-based smart buildings [40]. In this case, using the Samsung SmartThings platform, the authors compute a physical impact metric proportional to the distance between risky inter-app interactions and benign inter-app interactions.

Other kinds of impact have been analyzed as well. Packet delays have been measured as the impact of communication outages [87], and even the performance decrease after introducing cybersecurity controls has been considered [25]. However, most impact metrics have not considered the *sensitivity* of organizational assets as it was proposed in Chapter 5.

6.5 Conclusion

In this chapter, we have presented an extended evaluation of the sensitivity metric proposed in Chapter 5, using a smart building testbed. The unique feature of our testbed is its capability to emulate different scenarios by automatically configuring the building services according to the requirements of specific business processes. Using our testbed, we showed that the addition of context is required to properly assess the sensitivity of smart building components. More than a requirement, such context is a crucial aspect that can swing a component's assessment from high sensitivity (e.g., illumination in an operating room) to low sensitivity (e.g., the same illumination attack in a data center).

There is no degree of sensitivity inherently attached to any building component; it is always context dependent. Although, intuitively, it would have been possible to assume that there is a baseline sensitivity score for smart building components, and that the context only has a minor influence on such a score, our assessments under different scenarios show otherwise.

The hardware of our testbed is essentially similar to the hardware used in other testbeds. This allows us to perform realistic attacks against the testbed's infrastructure in the same way that previous works have done it, although with an additional feature: the victim's perspective. Beyond its usefulness to estimate the sensitivity of smart building components, the perspective given by specific contexts allows to fully illustrate the potential impact of cyber attacks.

Not all attacks yield a sensitivity impact. Only in those cases where the attacker manages to manipulate the targeted physical variable out of the business-specific limits, the organization is prone to suffer negative consequences. Attacks that do not drift physical variables out of the required limits may even be tolerated in order to keep the availability of building services. This may happen, e.g., when the smart building defenses counteract the attack or the attack simply lacks the "force" to

push the variable beyond these limits. The sensitivity impact materializes only after the physical variable crosses its predefined limits, whatever the means.

This chapter concludes our contributions on the assessment of security weaknesses, thus, closing the scope of this work. The next chapter presents the main conclusions of the overall thesis, including the main lessons of our research and promising future work directions.

Chapter 7

Concluding Remarks

We now present the main conclusions of this thesis. Moreover, we outline future research directions that would complement our current work.

7.1 Summary of Contributions

With the raising popularity of smart buildings, there is an urgent need to improve their current security posture. However, their increased connectivity has only worsened the risk of cyber attacks against them, now possible even from remote locations. This situation positions critical buildings such as hospitals, airports, and data centers as attractive targets for cyber attackers. To solve this problem, the adoption of IT security controls and processes to the smart buildings domain has been tried with limited success [53]. Instead, *smart buildings security* has developed its own body of knowledge by (1) adapting existing defensive methods to the peculiarities of this domain; and (2) developing new defensive methods specifically tailored for smart buildings. Among the peculiarities of smart building systems are the use of specialized protocols and programming environments, their physical interactions with the real world, and the heterogeneous components that take part of the network.

Academic and industrial research on smart buildings security has focused mostly on the detection of cyber attacks but little efforts have been made on their *prevention*. An effective way to prevent cyber attacks is by preemptively handling their vulnerabilities. Regardless of the specific domain, software vulnerabilities are typically handled by means of a *vulnerability management process*: a 3-step cyclic process that identifies, prioritizes, and remediates vulnerabilities. However, existing methods to implement this process mostly apply to IT systems and are

unfit for smart buildings. For instance, the *identification* of weaknesses using vulnerability scanners is ineffective for customized smart building applications that lack CVE records; IT *prioritization* approaches (e.g., CVSS) do not consider the diverse consequences of cyber-physical attacks; and update servers to *remediate* vulnerabilities are not available for heterogeneous components many of which only support manual updates through physical connections. Therefore, new methods were needed for smart buildings. *The development and (semi-) automated implementation of these methods were the primary concern of this thesis.* According to RFC 4949, vulnerabilities are the subset of exploitable weaknesses in a system [124]. Beyond just vulnerabilities, in this thesis we widened the reach of the traditional vulnerability management process to also consider the weaknesses that originate vulnerabilities. In particular, the scope of our work focuses on the first two stages of the vulnerability management process, namely, the identification and prioritization of security weaknesses.

In this context, the main research question addressed in our work was:

How to (semi-) automatically identify and assess security weaknesses in smart building applications and their configuration?

We divided this general question into three specific subquestions, the first of which focuses on the identification of security weaknesses:

Research Question 1. *How to (semi-) automate the identification of weaknesses in smart building applications?*

Smart building applications are typically developed in-house to satisfy specific building requirements [66]. Unfortunately, the security of smart buildings' software is rarely a priority [136]. On top of it, the application development process often involves proprietary and menu-based programming environments. This poses a serious challenge to create a generalizable application analysis method as it would be done in the IT and even the Industrial Control System (ICS) domains. We observed, however, that smart building applications are often built using a programming pattern known as *closed control loop (CCL)*, where sensors, setpoints, actuators, and control functions interact in complex relationships. Leveraging these relationships we created an abstract model of smart building applications as graph data structures where the components and relationships are represented by nodes and edges, respectively. In this way, the CCL graph exposes the application's architecture, which is the basis of our proposed approach to identify security weaknesses.

Inspired by well-known IT security weaknesses listed in Mitre's Common Weakness Enumeration (CWE), we aimed at identifying weaknesses related to *global variables* (CWE-1108), *multi-purpose variable*s (CWE-1109), *circular dependencies* (CWE-1047), and others, just by analyzing the application's graph. To do so, we look for specific patterns in the graph such as node cycles and central nodes. This approach allows us to encode weakness-related patterns as graph queries based on graph-theoretic algorithms.

The automated creation of CCL graphs was a challenge on its own. This is important because CCL graphs of real applications are typically comprised of thousands of nodes and edges. To address this problem, we developed a novel approach to automatically extract such graphs from smart buildings implemented using the BACnet protocol.

Our proposed approach has advantages and disadvantages. Its main drawback is that some weaknesses can only be identified through detailed (insider-like) knowledge of the system that is not available in our high level graph abstractions. For example, a potential division by zero (CWE-369) in the CCL control function. On the other hand, leveraging generic CCL graphs allowed us to apply our approach to diverse Cyber-Physical Systems (CPSs). Besides one real smart building application, we analyzed two simulated ICS control applications. We identified weakness-related patterns in all of them. Particularly, weaknesses related to (1) *global variables* caused by sensors/setpoints that feed multiple CCLs; and (2) *multi-purpose variables* caused by *override controllers*. Moreover, our evaluations showed a correlation between the weaknesses automatically identified by our approach and their potential to cause security breaches.

We deem our proposed approach as a simple and fast method to identify security weaknesses. Similar approaches on the ICS domain require much more information and manual work to achieve comparable results [80, 81]. From a defensive viewpoint, this is an important contribution to incentivize the addition of cyber security activities as part of the regular smart building management tasks. It is worth noting, however, that from an offensive perspective the same approach could be used to identify potential targets without an extensive reconnaissance of the system. This challenges the long-standing claim that a thorough reconnaissance phase is always required to execute successful CPS attacks.

The goal of the second subquestion is to identify security weaknesses in smart building configurations:

Research Question 2. *How to (semi-) automate the identification of weaknesses in smart building configurations?*

Adversaries can leverage configuration weaknesses in smart buildings to launch attacks or to conceal malicious behavior. Such weaknesses may occur in components that take part of the *management*, *automation*, and *field* layers of smart building systems; for instance, operator work stations (OWSs), building controllers, and smart sensors. In particular, we look for configuration weaknesses that manifest as *invalid* behavior of smart building components; specifically, their network interactions with other components. The ground-truth of valid behavior is taken from official documentation issued by the components' manufacturers.

Our proposed approach compares the documented capabilities of smart building components with their actual behavior as observed in the network traffic. Thus, our approach can identify weakness such as *expected behavior violation* (CWE-440) and *inclusion of undocumented features or chicken bits* (CWE-1242). To automate this idea, the main challenge was to automatically interpret the components' documentation, which is only available in PDF and without a standardized structure. To solve this problem, we developed a novel information retrieval method to extract the list of documented valid behavior for diverse smart building components. This list is used to create component behavior rules that are loaded into a network monitoring system (Zeek). We demonstrated our proposed approach in the context of BACnet-based smart buildings. Nonetheless, it could be applied to other systems based on similar protocols such as EtherNet/IP.

The main limitation of our approach is that it cannot extract valid behavior rules of arbitrary components, but only those for which there are network traffic traces showing their typical (yet not necessarily complete) behavior. However, if these network traces are available, our method is capable of extracting valid behavior rules with high precision and recall (above 99.5%). This leads to an effective identification of weaknesses as we showed in our evaluation in a real smart buildings network, where we found instances of both CWE-440 and CWE-1242. In the analyzed system, we found components that request capabilities unavailable in other components; thus, showing an *expected behavior violation* (CWE-440). All instances of this kind of weaknesses were found on local smart building management software. Regarding CWE-1242, we listed 128 instances of undocumented features in components from some of the most important manufacturers worldwide.

Finally, the third research subquestion dealt with the assessment of smart building components, which might be affected by application or configuration weaknesses:

Research Question 3. *How to (semi-) automate a sensitivity assessment for smart building components?*

Security weaknesses can be found on any of the components that take part of a smart building. The *risk* that such weaknesses represent for the organization hosted in the smart building must be the guiding principle to triage them. However, there are multiple factors that influence the estimated risk of a particular weakness. One of the most important risk factors is the *sensitivity* of the affected assets, i.e., the harm that a loss event might cause in terms of legal/regulatory consequences, reputational damage, fines, and others. Nonetheless, other important aspects of *risk* involve an assessment of weaknesses/vulnerabilities and an assessment of the threats that might exploit them. Whereas methodologies such as STRIDE and DREAD could be used to assess the threats, we believe that more research is needed regarding the assessment of smart building weaknesses and vulnerabilities.

To estimate the sensitivity of smart building components, we take business and technical aspects into consideration. On the business side, we consider the different business processes that are part of the organization and their requirements from the smart building, i.e., the building services needed to execute business processes. Furthermore, we leverage a *business impact analysis* to identify the sensitivity of business processes. Using this information, we can link the sensitivity of business processes to their required building services. On the technical side, we create a graph data structure where smart building components and their functional dependencies are represented by nodes and edges, respectively. We proposed three basic requirements that a graph centrality algorithm must satisfy to estimate the *sensitivity* of the components in the graph. Finally, we developed a graph centrality algorithm that complies with such requirements (BACRank). The proposed centrality metric summarizes the *sensitivity* of smart building components in a numeric score.

Despite our efforts to ease the sensitivity assessment of smart building components through automated tools, most of the information required to execute the proposed approach must be gathered manually. For instance, the business impact analysis is typically based on personal interviews with the organization managers. Nevertheless, a business impact analysis is usually already available in critical organizations such as hospitals, airports, and others. For smaller organizations, however, the difficulty to obtain some of the required information might impose a serious limitation to implement the proposed approach.

We executed the proposed assessment of smart building components in a real smart building. This evaluation showed that our approach successfully prioritizes the most sensitive components for the organization, i.e., the components that support the catering service: a business processes with strong contractual obligations between the host organization (building owner) and the external companies that offer the service. The business and technical aspects behind the results produced by

our approach were ratified by the local smart building administrators. Furthermore, we extended our evaluation of BACRank by developing a specialized testbed capable of emulating diverse realistic environments.

Summarizing, this thesis presents concrete methods to identify and assess security weaknesses in smart building applications. Along with the proposed methods, we provide software prototypes that (semi-) automate them. The proposed approach to assess security weaknesses is based on a comprehensive combination of business-related and technical information. On the other hand, our two approaches to identify security weaknesses map CWE entries to the smart buildings domain. In May 2 022, Mitre independently established the CWE-CAPEC Industrial Control System and Operational Technology Special Interest Group, whose work includes a similar mapping of CWE weaknesses to the ICS/OT domain. Thus, our pioneering contributions fit into a wider and current effort to improve the security of industrial and OT systems, where smart buildings are part of.

7.2 Future Research Directions

Throughout our work, we identified interesting future research directions for the identification, prioritization, and remediation of security weaknesses in smart buildings. Here we list potential research ideas that would complement our current work.

- **Weakness identification:** In Chapter 3, we discussed the analysis of smart building applications created using a widespread programming pattern known as *closed control loop (CCL)*. There are, however, new trends in the development of CPS applications. One of them is *intelligent control*, which leverages artificial intelligence computing approaches like neural networks. As it has been documented in other domains, neural networks are susceptible to adversarial attacks that can dangerously alter their expected behavior. Thus, the identification of security weaknesses in *intelligent control*-based systems will require new analysis techniques for smart buildings and other CPSs. The use of *digital twins* could be a promising direction to empirically explore the security risks of complex and obscure control approaches like those based on neural networks [12]. The use of AI-based approaches to identify weaknesses in smart buildings might also be an interesting research direction given the latest advances in this domain [123].

- **Weakness prioritization:** As discussed in Chapter 5, the prioritization of

security weaknesses should be based on the *risk* that such weaknesses represent for the affected organization. However, the concept of risk comprises several factors that must be estimated as accurately as possible to obtain a truthful overall risk estimation. In this thesis, we explored the estimation of one of these factors, namely, the *sensitivity* of smart building components. Other risk factors that are worth exploring in the context of smart buildings security are, for example, the probability of threat agents being in *contact* with vulnerable/weak components and the probability of threat agents taking *action* against them after the initial contact. We envision a honeypot-based study to estimate the attackers' *contact* and *action* probabilities against smart buildings. Additionally, it would be interesting to analyze to what extent the attackers' *action* leverages known vulnerabilities (i.e., those with a CVE record) or living-off-the-land utilities (e.g., valid *read* and *write* commands).

- **Weakness remediation:** After the identification and assessment of security weaknesses, their *remediation* closes the cycle. There are multiple challenges regarding the remediation of weaknesses, particularly on the *automation* layer, where building controllers reside. One of such challenges is the application of security patches, which often requires to stop the physical processes under control. Thus, affecting the building services and users. We envision a patch scheduling approach that aims to optimize different objectives such as (1) the technical compatibility between updated and still outdated components; and (2) the downtime of building services and their supported business processes.

Bibliography

Author References

[1] Herson Esquivel-Vargas, Marco Caselli, Geert Jan Laanstra and Andreas Peter. 'Putting Attacks in Context: A Building Automation Testbed for Impact Assessment from the Victim's Perspective'. In: *Detection of Intrusions and Malware, and Vulnerability Assessment - 17th International Conference, DIMVA, Lisbon, Portugal, June 24-26, 2020, Proceedings*. Ed. by Clémentine Maurice, Leyla Bilge, Gianluca Stringhini and Nuno Neves. Vol. 12223. Lecture Notes in Computer Science. Springer, pp. 44–64. DOI: 10.1007/978-3-030-52683-2_3. URL: https://doi.org/10.1007/978-3-030-52683-2_3.

[2] Herson Esquivel-Vargas, Marco Caselli and Andreas Peter. 'Automatic Deployment of Specification-based Intrusion Detection in the BACnet Protocol'. In: *Workshop on Cyber-Physical Systems Security and PrivaCy, Dallas, TX, USA, November 3, 2017*. Ed. by Bhavani M. Thuraisingham, Rakesh B. Bobba and Awais Rashid. ACM, pp. 25–36. DOI: 10.1145/3140241.3140244. URL: https://doi.org/10.1145/3140241.3140244.

[3] Herson Esquivel-Vargas, Marco Caselli and Andreas Peter. 'BACGraph: Automatic Extraction of Object Relationships in the BACnet Protocol'. In: *Dependable Systems and Networks - 51st Annual IEEE/IFIP International Conference, DSN, Taipei, Taiwan, June 21-24, 2021 - Supplemental Volume*. IEEE, pp. 45–48. DOI: 10.1109/DSN-S52858.2021.00029. URL: https://doi.org/10.1109/DSN-S52858.2021.00029.

[4] Herson Esquivel-Vargas, Marco Caselli, Erik Tews, Doina Bucur and Andreas Peter. 'BACRank: Ranking Building Automation and Control System Components by Business Continuity Impact'. In: *Computer Safety, Reliability, and Security - 38th International Conference, SAFECOMP, Turku,*

Finland, September 11-13, 2019. Ed. by Alexander B. Romanovsky, Elena Troubitsyna and Friedemann Bitsch. Vol. 11698. Lecture Notes in Computer Science. Springer, pp. 183–199. DOI: 10.1007/978-3-030-26601-1_13. URL: https://doi.org/10.1007/978-3-030-26601-1_13.

[5] Herson Esquivel-Vargas, John Henry Castellanos, Marco Caselli, Nils Ole Tippenhauer and Andreas Peter. 'Identifying Near-Optimal Single-Shot Attacks on ICSs with Limited Process Knowledge'. In: *Applied Cryptography and Network Security - 20th International Conference, ACNS, Rome, Italy, June 20-23, 2022, Proceedings*. Ed. by Giuseppe Ateniese and Daniele Venturi. Vol. 13269. Lecture Notes in Computer Science. Springer, pp. 170–192. DOI: 10.1007/978-3-031-09234-3_9. URL: https://doi.org/10.1007/978-3-031-09234-3_9.

Other References

[6] Al-Sudani Mustafa Qahtan Abdulmunem and Vyacheslav S Kharchenko. 'Availability and security assessment of smart building automation systems: combining of attack tree analysis and Markov models'. In: *Mathematics and Computers in Sciences and in Industry - 3rd International Conference, MCSI*. IEEE. 2016, pp. 302–307.

[7] Pascal Ackerman. *Industrial Cybersecurity*. 2nd ed. Birmingham, England: Packt Publishing, Oct. 2021.

[8] Sridhar Adepu and Aditya Mathur. 'Distributed Detection of Single-Stage Multipoint Cyber Attacks in a Water Treatment Plant'. In: *Asia Conference on Computer and Communications Security, AsiaCCS, Xi'an, China, May 30 - June 3, 2016*. Ed. by Xiaofeng Chen, XiaoFeng Wang and Xinyi Huang. ACM, pp. 449–460. DOI: 10.1145/2897845.2897855. URL: https://doi.org/10.1145/2897845.2897855.

[9] Vida Ahmadi, Patrik Arlos and Emiliano Casalicchio. 'Normalization Framework for Vulnerability Risk Management in Cloud'. In: *Future Internet of Things and Cloud - 8th International Conference, FiCloud, Rome, Italy, August 23-25, 2021*. Ed. by Muhammad Younas, Irfan Awan and Perin Ünal. IEEE, pp. 99–106. DOI: 10.1109/FiCloud49777.2021.00022. URL: https://doi.org/10.1109/FiCloud49777.2021.00022.

[10] Chuadhry Mujeeb Ahmed, Venkata Reddy Palleti and Aditya P. Mathur. 'WADI: a water distribution testbed for research in the design of secure cyber physical systems'. In: *International Workshop on Cyber-Physical Systems for Smart Water Networks, CySWATER@CPSWeek, Pittsburgh, Pennsylvania, USA, April 21, 2017*, pp. 25–28. DOI: 10.1145/3055366.3055375.

[11] Chuadhry Mujeeb Ahmed, Jianying Zhou and Aditya P. Mathur. 'Noise Matters: Using Sensor and Process Noise Fingerprint to Detect Stealthy Cyber Attacks and Authenticate sensors in CPS'. In: *Annual Computer Security Applications Conference, ACSAC 2018, San Juan, PR, USA, December 03-07, 2018*. ACM, pp. 566–581. DOI: 10.1145/3274694.3274748. URL: https://doi.org/10.1145/3274694.3274748.

[12] Cristina Alcaraz and Javier López. 'Digital Twin: A Comprehensive Survey of Security Threats'. In: *IEEE Commun. Surv. Tutorials* 24.3 (2022), pp. 1475–1503. DOI: 10.1109/COMST.2022.3171465. URL: https://doi.org/10.1109/COMST.2022.3171465.

[13] Magnus Almgren, Peter Andersson, Gunnar Björkman, Mathias Ekstedt, Jonas Hallberg, Simin Nadjm-Tehrani and Erik Westring. 'RICS-el: Building a National Testbed for Research and Training on SCADA Security (Short Paper)'. In: *Critical Information Infrastructures Security - 13th International Conference, CRITIS, Kaunas, Lithuania, September 24-26, 2018, Revised Selected Papers*, pp. 219–225. DOI: 10.1007/978-3-030-05849-4_17.

[14] Michael P Andersen, Gabe Fierro, Sam Kumar, Joyce Kim, Edward A Arens, Hui Zhang, Paul Raftery and David E Culler. 'Well-connected microzones for increased building efficiency and occupant comfort'. In: *ACEEE Summer Study on Energy Efficiency in Buildings*. 2016.

[15] ANSI/ASHRAE STANDARD 135-2020. *A Data Communication Protocol for Building Automation and Control Networks*. 2020.

[16] ANSI/ASHRAE STANDARD 62.1-2016. *Ventilation for Acceptable Indoor Air Quality*. 2016.

[17] ANSI/ASHRAE/ASHE STANDARD 170-2017. *Ventilation of Health Care Facilities*. 2017.

[18] ANSI/TIA. *ANSI/TIA-492-A Telecommunications Infrastructure Standard for Data Centers*. 2012.

[19] ANSI/TIA. *ANSI/TIA-569-C Telecommunications Pathways and Spaces*. 2012.

[20] Esteban Arroyo, Alexander Fay and Mario Hoernicke. *A Method of Digitalizing Engineering Documents*. [Online; accessed 02-May-2021]. 2016.

[21] Matthew P Barrett et al. *Framework for improving critical infrastructure cybersecurity*. Tech. rep. 2018.

[22] Dan Bilefsky. *Hackers Use New Tactic at Austrian Hotel: Locking the Doors*. https://www.nytimes.com/2017/01/30/world/europe/hotel-austria-bitcoin-ransom.html. [Online; accessed 27-Jun-2019]. 2017.

[23] Brad Bowers. *ShmooCon 2013: How To Own A Building: Exploiting the Physical World With Bacnet*. https://youtu.be/d3jtmv6Y9uk?t=695. [Online; accessed 29-Jan-2022]. 2013.

[24] Sergey Brin and Lawrence Page. 'The Anatomy of a Large-Scale Hypertextual Web Search Engine'. In: *Comput. Networks* 30.1-7 (1998), pp. 107–117. DOI: 10.1016/S0169-7552(98)00110-X. URL: https://doi.org/10.1016/S0169-7552(98)00110-X.

[25] Richard Candell, Keith Stouffer and Dhananjay Anand. 'A cybersecurity testbed for industrial control systems'. In: *Process Control and Safety Symposium*. 2014.

[26] Alvaro A. Cárdenas, Saurabh Amin, Zong-Syun Lin, Yu-Lun Huang, Chi-Yen Huang and Shankar Sastry. 'Attacks against process control systems: risk assessment, detection, and response'. In: *Asia Conference on Computer and Communications Security, AsiaCCS, Hong Kong, China, March 22-24, 2011*. Ed. by Bruce S. N. Cheung, Lucas Chi Kwong Hui, Ravi S. Sandhu and Duncan S. Wong. ACM, pp. 355–366. URL: https://dl.acm.org/citation.cfm?id=1966959.

[27] Marco Caselli, Emmanuele Zambon, Johanna Amann, Robin Sommer and Frank Kargl. 'Specification Mining for Intrusion Detection in Networked Control Systems'. In: *USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*. Ed. by Thorsten Holz and Stefan Savage. USENIX Association, pp. 791–806. URL: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/caselli.

[28] John Henry Castellanos, Martín Ochoa and Jianying Zhou. 'Finding Dependencies between Cyber-Physical Domains for Security Testing of Industrial Control Systems'. In: *Annual Computer Security Applications Conference, ACSAC 2018, San Juan, PR, USA, December 03-07, 2018*. ACM,

2018, pp. 582–594. DOI: 10.1145/3274694.3274745. URL: https://doi.org/10.1145/3274694.3274745.

[29] Pavel Čeleda, Radek Krejčí and Vojtěch Krmíček. '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, Budapest, Hungary, August 29-31, 2012. Proceedings*. Ed. by Róbert Szabó and Attila Vidács. Vol. 7479. Lecture Notes in Computer Science. Springer, pp. 64–75. DOI: 10.1007/978-3-642-32808-4_7. URL: https://doi.org/10.1007/978-3-642-32808-4_7.

[30] Raymond Chan, Forest Tan, Ulric Teo and Brandon Kow. 'Vulnerability Assessments of Building Management Systems'. In: *Critical Infrastructure Protection - 14th IFIP WG 11.10 International Conference, ICCIP, Arlington, VA, USA, March 16-17, 2020, Revised Selected Papers*. Ed. by Jason Staggs and Sujeet Shenoi. Vol. 596. IFIP Advances in Information and Communication Technology. Springer, pp. 209–220. DOI: 10.1007/978-3-030-62840-6_10. URL: https://doi.org/10.1007/978-3-030-62840-6_10.

[31] Checkstyle. *Checkstyle 8.39*. https://checkstyle.sourceforge.io/. [Online; accessed 04-Jan-2021]. 2021.

[32] Haipeng Chen, Jing Liu, Rui Liu, Noseong Park and V. S. Subrahmanian. 'VEST: A System for Vulnerability Exploit Scoring & Timing'. In: *International Joint Conference on Artificial Intelligence, IJCAI, Macao, China, August 10-16, 2019*. Ed. by Sarit Kraus. ijcai.org, pp. 6503–6505. DOI: 10.24963/ijcai.2019/937. URL: https://doi.org/10.24963/ijcai.2019/937.

[33] Yuqi Chen, Christopher M. Poskitt, Jun Sun, Sridhar Adepu and Fan Zhang. 'Learning-Guided Network Fuzzing for Testing Cyber-Physical System Defences'. In: *Automated Software Engineering - 34th IEEE/ACM International Conference, ASE, San Diego, CA, USA, November 11-15, 2019*. IEEE, pp. 962–973. DOI: 10.1109/ASE.2019.00093. URL: https://doi.org/10.1109/ASE.2019.00093.

[34] Yuqi Chen, Bohan Xuan, Christopher M. Poskitt, Jun Sun and Fan Zhang. 'Active fuzzing for testing and securing cyber-physical systems'. In: *International Symposium on Software Testing and Analysis, ISSTA, Virtual Event, USA, July 18-22, 2020*. Ed. by Sarfraz Khurshid and Corina S.

Pasareanu. ACM, pp. 14–26. DOI: 10.1145/3395363.3397376. URL: https://doi.org/10.1145/3395363.3397376.

[35] Jean-Pierre Corriou. *Process control*. Springer, 2004.

[36] S. Corzine. *Operational and Business Continuity Planning for Prolonged Airport Disruptions*. Vol. 93. Transportation Research Board, 2013.

[37] Security Standards Council. *Payment Card Industry Data Security Standard: Requirements and Testing Procedures*. Mar. 2022.

[38] Bettina Crepin, Michael Bunk, Roland Galbas, Ralf Kinder, Matthias Schanzenbach and Christoph Betz. *Closed-loop control of brake pressure using a pressure-limiting valve*. US Patent 8,489,301. July 2013.

[39] Dániel Darvas, Enrique Blanco Vinuela and Borja Fernández Adiego. 'PLCverif: A tool to verify PLC programs based on model checking techniques'. In: *Accelerator and Large Experimental Physics Control Systems - 15th International Conference*. 2015.

[40] Wenbo Ding and Hongxin Hu. 'On the Safety of IoT Device Physical Interaction Control'. In: *Computer and Communications Security - 25th ACM Conference, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*. Ed. by David Lie, Mohammad Mannan, Michael Backes and XiaoFeng Wang. ACM, pp. 832–846. DOI: 10.1145/3243734.3243865. URL: https://doi.org/10.1145/3243734.3243865.

[41] James J. Downs and Ernest F. Vogel. 'A plant-wide industrial process control problem'. In: *Computers & chemical engineering* 17.3 (1993), pp. 245–255.

[42] Bowen Du, Marlie C. Tandoc, Michael L. Mack and Jeffrey A. Siegel. 'Indoor CO2 concentrations and cognitive function: A critical review'. In: *Indoor Air* 30.6 (2020), pp. 1067–1082. DOI: https://doi.org/10.1111/ina.12706. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/ina.12706.

[43] Davide Fauri, Michail Kapsalakis, Daniel Ricardo dos Santos, Elisa Costante, Jerry den Hartog and Sandro Etalle. 'Leveraging Semantics for Actionable Intrusion Detection in Building Automation Systems'. In: *Critical Information Infrastructures Security - 13th International Conference, CRITIS, Kaunas, Lithuania, September 24-26, 2018, Revised Selected Papers*. Ed. by Eric A. M. Luiijf, Inga Zutautaite and Bernhard M. Hämmerli. Vol. 11260. Lecture Notes in Computer Science. Springer, pp. 113–125.

DOI: 10.1007/978-3-030-05849-4_9. URL: https://doi.org/10.1007/978-3-030-05849-4_9.

[44] Davide Fauri, Michail Kapsalakis, Daniel Ricardo dos Santos, Elisa Costante, Jerry den Hartog and Sandro Etalle. 'Role Inference + Anomaly Detection = Situational Awareness in BACnet Networks'. In: *Detection of Intrusions and Malware, and Vulnerability Assessment - 16th International Conference, DIMVA, Gothenburg, Sweden, June 19-20, 2019, Proceedings*. Ed. by Roberto Perdisci, Clémentine Maurice, Giorgio Giacinto and Magnus Almgren. Vol. 11543. Lecture Notes in Computer Science. Springer, pp. 461–481. DOI: 10.1007/978-3-030-22038-9_22. URL: https://doi.org/10.1007/978-3-030-22038-9_22.

[45] Federal Ministry of Transport Building and Urban Affairs. *BACnet in public buildings*. http://www.amev-online.de/AMEVInhalt/Planen/Gebaeudeautomation/BACnet%202011%20V%201.2/bacnet2011v1-2en.pdf. [Online; accessed 20-Mar-2021]. 2011.

[46] Glenn A. Fink, Thomas W. Edgar, Theora R. Rice, Douglas G. MacDonald and Cary E. Crawford. 'Overview of Security and Privacy in Cyber-Physical Systems'. In: *Security and Privacy in Cyber-Physical Systems*. John Wiley & Sons, Ltd, 2017. Chap. 1, pp. 1–23. ISBN: 9781119226079. DOI: https://doi.org/10.1002/9781119226079.ch1. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119226079.ch1. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119226079.ch1.

[47] First. *Common Vulnerability Scoring System version 3.1: Specification Document*. https://www.first.org/cvss/specification-document. [Online; accessed 01-Mar-2022]. 2019.

[48] David Fisher, Bernhard Isler and Michael Osborne. *BACnet Secure Connect*. Tech. rep. 2019.

[49] Dario Forte. 'Vulnerability Management: One Problem, Several Potential Approaches.' In: *Network Security* 5 (2002), pp. 11–13. ISSN: 1353-4858. DOI: https://doi.org/10.1016/S1353-4858(02)05013-4. URL: https://www.sciencedirect.com/science/article/pii/S1353485802050134.

[50] The Apache Software Foundation. *Apache Solr*. https://lucene.apache.org/solr/. [Online; accessed 03-Mar-2021]. 2022.

[51] Béla Genge, István Kiss and Piroska Haller. 'A system dynamics approach for assessing the impact of cyber attacks on critical infrastructures'. In: *Int. J. Crit. Infrastructure Prot.* 10 (2015), pp. 3–17. DOI: 10.1016/j.ijcip.2015.04.001. URL: https://doi.org/10.1016/j.ijcip.2015.04.001.

[52] Javier Gonzalez-Martin, Norbertus Johannes Richardus Kraakman, Cristina Perez, Raquel Lebrero and Raul Munoz. 'A state–of–the-art review on indoor air pollution and strategies for indoor air pollution control'. In: *Chemosphere* 262 (2021), p. 128376.

[53] Wolfgang Granzer and Wolfgang Kastner. 'Security Analysis of Open Building Automation Systems'. In: *Computer Safety, Reliability, and Security, 29th International Conference, SAFECOMP, Vienna, Austria, September 14-17, 2010*. Ed. by Erwin Schoitsch. Vol. 6351. Lecture Notes in Computer Science. Springer, pp. 303–316. DOI: 10.1007/978-3-642-15651-9_23. URL: https://doi.org/10.1007/978-3-642-15651-9_23.

[54] Vitor Graveto, Tiago Cruz and Paulo Simões. 'Security of Building Automation and Control Systems: Survey and future research directions'. In: *Comput. Secur.* 112 (2022), p. 102527. DOI: 10.1016/j.cose.2021.102527. URL: https://doi.org/10.1016/j.cose.2021.102527.

[55] Benjamin Green, Marina Krotofil and Ali Abbasi. 'On the Significance of Process Comprehension for Conducting Targeted ICS Attacks'. In: *Workshop on Cyber-Physical Systems Security and PrivaCy, Dallas, TX, USA, November 3, 2017*. Ed. by Bhavani M. Thuraisingham, Rakesh B. Bobba and Awais Rashid. ACM, pp. 57–67. DOI: 10.1145/3140241.3140254. URL: https://doi.org/10.1145/3140241.3140254.

[56] M. Guelman, A. Kogan, A. Kazarian, A. Livne, M. Orenstein and H. Michalik. 'Acquisition and pointing control for inter-satellite laser communications'. In: *IEEE Transactions on Aerospace and Electronic Systems* 40.4 (2004), pp. 1239–1248. DOI: 10.1109/TAES.2004.1386877.

[57] Dina Hadziosmanovic, Damiano Bolzoni, Sandro Etalle and Pieter H. Hartel. 'Challenges and opportunities in securing industrial control systems'. In: *Complexity in Engineering, COMPENG, Aachen, Germany, June 11-13, 2012*. IEEE, pp. 1–6. DOI: 10.1109/CompEng.2012.6242970. URL: https://doi.org/10.1109/CompEng.2012.6242970.

[58] Stephen Hilt and Michael Toecker. *bacnet-info NSE Script*. https://nmap.org/nsedoc/scripts/bacnet-info.html. [Online; accessed 15-Set-2021].

[59] David Holmberg and D Evans. *BACnet wide area network security threat assessment*. Tech. rep. 2003.

[60] David G Holmberg, Joel J Bender and Michael A Galler. *Using the BACnet (R) firewall router*. Tech. rep. 2006.

[61] Yu-Lun Huang, Alvaro A. Cárdenas, Saurabh Amin, Zong-Syun Lin, Hsin-Yi Tsai and Shankar Sastry. 'Understanding the physical and economic consequences of attacks on control systems'. In: *Int. J. Crit. Infrastructure Prot.* 2.3 (2009), pp. 73–83. DOI: 10.1016/j.ijcip.2009.06.001.

[62] *IEC 61131-3:2013. Programmable controllers - Part 3: Programming languages*. Standard. International Electrotechnical Commission, 2013.

[63] Fortune Business Insights. *Global Smart Building Market*. https://www.fortunebusinessinsights.com/industry-reports/smart-building-market-101198. [Online; accessed 19-Jan-2022]. 2021.

[64] BACnet International. *BACnet Testing Laboratories*. http://bacnetinternational.net/btl/. [Online; accessed 23-Nov-2021].

[65] ISACA. *Cybersecurity Fundamentals Glossary*. https://www.isaca.org/resources/glossary. [Online; accessed 21-Set-2021].

[66] *ISO 16484-1:2010. Building Automation and Control Systems (BACS) — Part 1: Project specification and implementation*. Standard. International Organization for Standardization, 2010.

[67] *ISO 16484-6:2020. Building Automation and Control Systems (BACS) — Part 6: Data communication conformance testing*. Standard. International Organization for Standardization, 2020.

[68] BS ISO. *22301:2012. Societal security. Business continuity management systems. Requirements*. Standard. 2012.

[69] *ISO 27031:2011. Information technology –Security techniques– Guidelines for information and communication technology readiness for business continuity*. Standard. International Organization for Standardization, 2011.

[70] Paul Jaccard. 'The Distribution of the Flora in the Alpine Zone'. In: *New Phytologist* 11.2 (1912), pp. 37–50. ISSN: 1469-8137. DOI: 10.1111/j.1469-8137.1912.tb05611.x. URL: http://dx.doi.org/10.1111/j.1469-8137.1912.tb05611.x.

[71] Tyler A. Jacobson, Jasdeep S. Kler, Michael T. Hernke, Rudolf K. Braun, Keith C. Meyer and William E. Funk. 'Direct human health risks of increased atmospheric carbon dioxide'. In: *Nature Sustainability* 2.8 (2019), pp. 691–701. ISSN: 2398-9629. DOI: 10.1038/s41893-019-0323-1. URL: https://doi.org/10.1038/s41893-019-0323-1.

[72] Paria Jokar, Hasen Nicanfar and Victor C. M. Leung. 'Specification-based Intrusion Detection for home area networks in smart grids'. In: *Second International Conference on Smart Grid Communications, SmartGridComm, Brussels, Belgium, October 17-20, 2011*. IEEE, pp. 208–213. DOI: 10.1109/SmartGridComm.2011.6102320. URL: https://doi.org/10.1109/SmartGridComm.2011.6102320.

[73] Jack Jones. 'An introduction to factor analysis of information risk (fair)'. In: *Norwich Journal of Information Assurance* 2.1 (2006), p. 67.

[74] Steve Karg. *BACnet Stack*. http://bacnet.sourceforge.net/. [Online; accessed 01-May-2019].

[75] Kaspersky ICS CERT. *Vulnerability in ICS: assessing the severity*. https://ics-cert.kaspersky.com/publications/reports/2022/04/20/vulnerability-in-ics-assessing-the-severity/. [Online; accessed 20-Apr-2022]. 2022.

[76] Jaspreet Kaur, Jernej Tonejc, Steffen Wendzel and Michael Meier. 'Securing BACnet's Pitfalls'. In: *ICT Systems Security and Privacy Protection - 30th IFIP TC 11 International Conference, SEC, Hamburg, Germany, May 26-28, 2015*. Ed. by Hannes Federrath and Dieter Gollmann. Vol. 455. IFIP Advances in Information and Communication Technology. Springer, pp. 616–629. DOI: 10.1007/978-3-319-18467-8_41. URL: https://doi.org/10.1007/978-3-319-18467-8_41.

[77] Georgios Kavallieratos, Sokratis K. Katsikas and Vasileios Gkioulos. 'Towards a Cyber-Physical Range'. In: *Cyber-Physical System Security Workshop, CPSS@AsiaCCS, Auckland, New Zealand, 8 July 2019*. ACM, pp. 25–34. DOI: 10.1145/3327961.3329532. URL: https://doi.org/10.1145/3327961.3329532.

[78] Joonsoo Kim, Kyeongho Kim and Moonsu Jang. 'Cyber-Physical Battlefield Platform for Large-Scale Cybersecurity Exercises'. In: *Cyber Conflict - 11th International Conference, CyCon, Tallinn, Estonia, May 28-31, 2019*, pp. 1–19. DOI: 10.23919/CYCON.2019.8756901.

[79] Charalambos Konstantinou, Marios Sazos and Michail Maniatakos. 'Attacking the smart grid using public information'. In: *Latin-American Test Symposium, LATS, Foz do Iguacu, Brazil, April 6-8, 2016*. IEEE, pp. 105–110. DOI: 10.1109/LATW.2016.7483348. URL: https://doi.org/10.1109/LATW.2016.7483348.

[80] Marina Krotofil and Alvaro A. Cárdenas. 'Resilience of Process Control Systems to Cyber-Physical Attacks'. In: *Nordic Conference on Secure IT Systems, NordSec, Ilulissat, Greenland, October 18-21, 2013*. Ed. by Hanne Riis Nielson and Dieter Gollmann. Vol. 8208. Lecture Notes in Computer Science. Springer, pp. 166–182. DOI: 10.1007/978-3-642-41488-6_12.

[81] Marina Krotofil, Alvaro A. Cárdenas, Bradley Manning and Jason Larsen. 'CPS: driving cyber-physical systems to unsafe operating conditions by timing DoS attacks on sensor signals'. In: *Proceedings of the 30th Annual Computer Security Applications Conference, ACSAC, New Orleans, LA, USA, December 8-12, 2014*. Ed. by Charles N. Payne Jr., Adam Hahn, Kevin R. B. Butler and Micah Sherr. ACM, pp. 146–155. DOI: 10.1145/2664243.2664290. URL: https://doi.org/10.1145/2664243.2664290.

[82] Kirill Kruglov. *Threat landscape for smart buildings. H1 2019 in brief*. Research Report. [Online; accessed 27-Apr-2022]. 2019.

[83] Truls Larsson, Kristin Hestetun, Espen Hovland and Sigurd Skogestad. 'Self-optimizing control of a large-scale plant: The Tennessee Eastman process'. In: *Industrial & engineering chemistry research* 40.22 (2001), pp. 4889–4901.

[84] Edward A. Lee. 'Cyber Physical Systems: Design Challenges'. In: *International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2008), 5-7 May 2008, Orlando, Florida, USA*. IEEE Computer Society, pp. 363–369. DOI: 10.1109/ISORC.2008.25. URL: https://doi.org/10.1109/ISORC.2008.25.

[85] Vladimir I Levenshtein et al. 'Binary codes capable of correcting deletions, insertions, and reversals'. In: *Soviet physics doklady*. Vol. 10. 8. Soviet Union. 1966, pp. 707–710.

[86] Qin Lin, Sridhar Adepu, Sicco Verwer and Aditya Mathur. 'TABOR: A Graphical Model-based Approach for Anomaly Detection in Industrial Control Systems'. In: *Asia Conference on Computer and Communications Security, AsiaCCS, Incheon, Republic of Korea, June 04-08, 2018*. Ed. by Jong Kim, Gail-Joon Ahn, Seungjoo Kim, Yongdae Kim, Javier López and

Taesoo Kim. ACM, pp. 525–536. DOI: 10.1145/3196494.3196546. URL: https://doi.org/10.1145/3196494.3196546.

[87] Ren Liu, Ceeman Vellaithurai, Saugata S. Biswas, Thoshitha T. Gamage and Anurag K. Srivastava. 'Analyzing the Cyber-Physical Impact of Cyber Events on the Power Grid'. In: *IEEE Trans. Smart Grid* 6.5 (2015), pp. 2444–2453. DOI: 10.1109/TSG.2015.2432013.

[88] Ponemon Institute LLC. *The Economic Value of Prevention in the Cybersecurity Lifecycle*. Research Report. [Online; accessed 28-Feb-2021]. 2020.

[89] William L. Luyben, Björn D. Tyréus and Michael L. Luyben. *Plantwide process control*. McGraw-Hill, 1999.

[90] Philip Richard Lyman. 'Plant-wide control structures for the Tennessee Eastman process'. MA thesis. Lehigh University, 1992.

[91] Gordon Lyon. *Nmap: the Network Mapper*. https://nmap.org/. [Online; accessed 13-Jan-2022].

[92] John C. Mace, Ricardo Melo Czekster, Charles Morisset and Carsten Maple. 'Smart Building Risk Assessment Case Study: Challenges, Deficiencies and Recommendations'. In: *European Dependable Computing Conference, EDCC, Munich, Germany, September 7-10, 2020*. IEEE, pp. 59–64. DOI: 10.1109/EDCC51268.2020.00019. URL: https://doi.org/10.1109/EDCC51268.2020.00019.

[93] Metropolitan. *DDoS attack halts heating in Finland amidst winter*. https://metropolitan.fi/entry/ddos-attack-halts-heating-in-finland-amidst-winter. [Online; accessed 22-Oct-2019].

[94] Barton P. Miller, Lars Fredriksen and Bryan So. 'An Empirical Study of the Reliability of UNIX Utilities'. In: *Commun. ACM* 33.12 (1990), pp. 32–44. DOI: 10.1145/96267.96279. URL: https://doi.org/10.1145/96267.96279.

[95] Mitre. *Common Weakness Scoring System (CWSS)*. https://cwe.mitre.org/cwss/. [Online; accessed 18-May-2022]. 2014.

[96] Mitre. *Common Weakness Enumeration: A Community-Developed List of Software & Hardware Weakness Types*. https://cwe.mitre.org/. [Online; accessed 29-Mar-2022]. 2020.

[97] Reiko Ann Miura-Ko and Nicholas Bambos. 'SecureRank: A Risk-Based Vulnerability Management Scheme for Computing Infrastructures'. In: *International Conference on Communications, ICC, Glasgow, Scotland, UK, 24-28 June 2007*. IEEE, pp. 1455–1460. DOI: 10.1109/ICC.2007.244. URL: https://doi.org/10.1109/ICC.2007.244.

[98] Hassan Mokalled, Concetta Pragliola, Daniele Debertol, Ermete Meda and Rodolfo Zunino. 'A Comprehensive Framework for the Security Risk Management of Cyber-Physical Systems'. In: *Resilience of Cyber-Physical Systems, From Risk Modelling to Threat Counteraction*. Ed. by Francesco Flammini. Springer, 2019, pp. 49–68. DOI: 10.1007/978-3-319-95597-1_3. URL: https://doi.org/10.1007/978-3-319-95597-1_3.

[99] Cesar Navas. *ICS Vulnerability Summary*. https://www.tenable.com/sc-dashboards/ics-vulnerability-summary. [Online; accessed 21-Mar-2022]. 2019.

[100] Neo4j. *Neo4j Graph Platform – The Leader in Graph Databases*. https://neo4j.com/. [Online; accessed 09-Jan-2021]. 2021.

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

[102] Hamed Orojloo and Mohammad Abdollahi Azgomi. 'A method for evaluating the consequence propagation of security attacks in cyber-physical systems'. In: *Future Gener. Comput. Syst.* 67 (2017), pp. 57–71. DOI: 10.1016/j.future.2016.07.016. URL: https://doi.org/10.1016/j.future.2016.07.016.

[103] Zhiwen Pan, Salim Hariri and Jesus Pacheco. 'Context aware intrusion detection for building automation systems'. In: *Comput. Secur.* 85 (2019), pp. 181–201. DOI: 10.1016/j.cose.2019.04.011. URL: https://doi.org/10.1016/j.cose.2019.04.011.

[104] Christoforos Panos, Christos Xenakis, Platon Kotzias and Ioannis Stavrakakis. 'A specification-based intrusion detection engine for infrastructure-less networks'. In: *Comput. Communications* 54 (2014), pp. 67–83. DOI: 10.1016/j.comcom.2014.08.002. URL: https://doi.org/10.1016/j.comcom.2014.08.002.

[105] Martin Peters, Johannes Goltz, Simeon Wiedenmann and Thomas Mundt. 'Using Machine Learning to Find Anomalies in Field Bus Network Traffic'. In: *Security, Privacy, and Anonymity in Computation, Communication, and Storage - 12th International Conference, SpaCCS , Atlanta, GA, USA,*

July 14-17, 2019. Ed. by Guojun Wang, Jun Feng, Md. Zakirul Alam Bhuiyan and Rongxing Lu. Vol. 11611. Lecture Notes in Computer Science. Springer, pp. 336–353. DOI: 10.1007/978-3-030-24907-6_26. URL: https://doi.org/10.1007/978-3-030-24907-6_26.

[106] PMD. *PMD: An extensible cross-language static code analyzer*. https://pmd.github.io/. [Online; accessed 04-Jan-2021].

[107] Shiva Poudel, Zhen Ni and Naresh Malla. 'Real-time cyber physical system testbed for power system security and control'. In: *International Journal of Electrical Power & Energy Systems* 90 (2017), pp. 124–133.

[108] The Zeek Project. *The Zeek Network Security Monitor*. https://zeek.org/. [Online; accessed 23-Mar-2022].

[109] Qualys. *Vulnerability Management Tool*. https://www.qualys.com/apps/vulnerability-management/. [Online; accessed 18-Mar-2022].

[110] Rapid7. *Vulnerability Management Tool*. https://www.rapid7.com/products/nexpose/. [Online; accessed 18-Mar-2022].

[111] Mark Rea. *The IESNA lighting handbook: reference & application*. New York, NY: Illuminating Engineering Society of North America, 2000. ISBN: 0879951508.

[112] N Lawrence Ricker. 'Model predictive control of a continuous, nonlinear, two-phase reactor'. In: *Journal of Process Control* 3.2 (1993), pp. 109–123.

[113] N. Lawrence Ricker. 'Decentralized control of the Tennessee Eastman challenge process'. In: *Journal of Process Control* 6.4 (1996), pp. 205–221.

[114] N. Lawrence Ricker. *Tennessee Eastman Challenge Archive*. https://depts.washington.edu/control/LARRY/TE/download.html. [Online; accessed 25-July-2020].

[115] Jason Rolleston. *What is Risk-Based Vulnerability Management?* https://www.kennasecurity.com/blog/what-is-risk-based-vulnerability-management/. [Online; accessed 18-Mar-2022]. 2020.

[116] Eyal Ronen and Adi Shamir. 'Extended Functionality Attacks on IoT Devices: The Case of Smart Lights'. In: *IEEE European Symposium on Security and Privacy, EuroS&P, Saarbrücken, Germany, March 21-24, 2016*. IEEE, pp. 3–12. DOI: 10.1109/EuroSP.2016.13. URL: https://doi.org/10.1109/EuroSP.2016.13.

[117] Jerome H. Saltzer and Michael D. Schroeder. 'The protection of information in computer systems'. In: *Proc. IEEE* 63.9 (1975), pp. 1278–1308. DOI: 10.1109/PROC.1975.9939. URL: https://doi.org/10.1109/PROC.1975.9939.

[118] Mary Sanders. *ACSM's health/fitness facilities standards and guidelines*. Champaign, IL: Human Kinetics, 2019. ISBN: 1492567183.

[119] Daniel Ricardo dos Santos, Mario Dagrada and Elisa Costante. 'Leveraging operational technology and the Internet of things to attack smart buildings'. In: *Journal of Computer Virology and Hacking Techniques* 17.1 (2021), pp. 1–20. DOI: 10.1007/s11416-020-00358-8. URL: https://doi.org/10.1007/s11416-020-00358-8.

[120] Esha Sarkar, Hadjer Benkraouda and Michail Maniatakos. 'I came, I saw, I hacked: Automated Generation of Process-independent Attacks for Industrial Control Systems'. In: *Asia Conference on Computer and Communications Security, AsiaCCS, Taipei, Taiwan, October 5-9, 2020*. Ed. by Hung-Min Sun, Shiuh-Pyng Shieh, Guofei Gu and Giuseppe Ateniese. ACM, pp. 744–758. DOI: 10.1145/3320269.3384730. URL: https://doi.org/10.1145/3320269.3384730.

[121] R. Sekar, Ajay Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang and S. Zhou. 'Specification-based anomaly detection: a new approach for detecting network intrusions'. In: *Computer and Communications Security - 9th ACM Conference, CCS, Washington, DC, USA, November 18-22, 2002*. Ed. by Vijayalakshmi Atluri. ACM, pp. 265–274. DOI: 10.1145/586110.586146. URL: https://doi.org/10.1145/586110.586146.

[122] KLS Sharma. *Overview of industrial process automation*. Elsevier, 2016.

[123] Tushar Sharma, Maria Kechagia, Stefanos Georgiou, Rohit Tiwari and Federica Sarro. 'A Survey on Machine Learning Techniques for Source Code Analysis'. In: *CoRR* abs/2110.09610 (2021). arXiv: 2110.09610. URL: https://arxiv.org/abs/2110.09610.

[124] R. Shirey. *Internet Security Glossary, Version 2*. RFC 4949. IETF, 2007. URL: https://tools.ietf.org/html/rfc4949.

[125] Adam Shostack. 'Experiences Threat Modeling at Microsoft'. In: *Proceedings of the Workshop on Modeling Security (MODSEC08) held as part of the 2008 International Conference on Model Driven Engineering Languages and Systems (MODELS) Toulouse, France, September 28, 2008*. Ed. by Jon Whittle, Jan Jürjens, Bashar Nuseibeh and Glen Dob-

son. Vol. 413. CEUR Workshop Proceedings. CEUR-WS.org. URL: http://ceur-ws.org/Vol-413/paper12.pdf.

[126] Garry M. Steil and Kerstin Rebrin. *Closed loop system for controlling insulin infusion*. US Patent 7,267,665. Sept. 2007.

[127] Keith Stouffer, Victoria Pillitteri, Suzanne Lightman, Marshall Abrams and Adam Hahn. *Guide to Industrial Control Systems (ICS) Security*. en. Tech. rep. 2015. DOI: https://doi.org/10.6028/NIST.SP.800-82r2.

[128] De Telegraaf. *Hack met medicijnen: 'Hoe haal je het in je hoofd?!'* https://www.telegraaf.nl/nieuws/2841336/hack-met-medicijnen-hoe-haal-je-het-in-je-hoofd. [Online; accessed 12-Feb-2022]. 2018.

[129] Tenable. *Vulnerability Management Tool*. https://www.tenable.com/products/tenable-io. [Online; accessed 18-Mar-2022]. 2022.

[130] Lane Thames and Dirk Schaefer, eds. *Cybersecurity for industry 4.0*. en. 1st ed. Springer Series in Advanced Manufacturing. Basel, Switzerland: Springer International Publishing, Apr. 2017.

[131] Nils Ole Tippenhauer. 'Design and Realization of Testbeds for Security Research in the Industrial Internet of Things'. In: *Security and Privacy Trends in the Industrial Internet of Things*. Ed. by Cristina Alcaraz. Springer, 2019, pp. 287–310. DOI: 10.1007/978-3-030-12330-7_14. URL: https://doi.org/10.1007/978-3-030-12330-7_14.

[132] Jernej Tonejc, Sabrina Guettes, Alexandra Kobekova and Jaspreet Kaur. 'Machine Learning Methods for Anomaly Detection in BACnet Networks'. In: *J. Univers. Comput. Sci.* 22.9 (2016), pp. 1203–1224. URL: http://www.jucs.org/jucs_22_9/machine_learning_methods_for.

[133] David I. Urbina, Jairo Alonso Giraldo, Alvaro A. Cárdenas, Nils Ole Tippenhauer, Junia Valente, Mustafa Amir Faisal, Justin Ruths, Richard Candell and Henrik Sandberg. 'Limiting the Impact of Stealthy Attacks on Industrial Control Systems'. In: *Computer and Communications Security - 23rd ACM Conference, CCS, Vienna, Austria, October 24-28, 2016*, pp. 1092–1105. DOI: 10.1145/2976749.2978388.

[134] Anurag Verma, Surya Prakash, Vishal Srivastava, Anuj Kumar and Subhas Chandra Mukhopadhyay. 'Sensing, controlling, and IoT infrastructure in smart building: A review'. In: *IEEE Sensors Journal* 19.20 (2019), pp. 9036–9046.

[135] Harold L. Wade. *Basic and Advanced Regulatory Control - System Design and Application (3rd Edition)*. ISA, 2017. ISBN: 978-0-87664-013-5.

[136] Steffen Wendzel. 'How to Increase the Security of Smart Buildings?' In: *Communications of the ACM* 59.5 (), pp. 47–49. ISSN: 0001-0782. DOI: 10.1145/2828636. URL: https://doi.org/10.1145/2828636.

[137] Steffen Wendzel, Jernej Tonejc, Jaspreet Kaur and Alexandra Kobekova. 'Cyber Security of Smart Buildings'. In: *Security and Privacy in Cyber-Physical Systems*. John Wiley & Sons, Ltd, 2017. Chap. 16, pp. 327–351. ISBN: 9781119226079. DOI: https://doi.org/10.1002/9781119226079.ch16. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119226079.ch16. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119226079.ch16.

[138] Steffen Wendzel, Viviane Zwanger, Michael Meier and Sebastian Szlósarczyk. 'Envisioning Smart Building Botnets'. In: *Sicherheit, Schutz und Zuverlässigkeit, Beiträge der 7. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik e.V. (GI), 19.-21. März 2014, Wien, Österreich*. Ed. by Stefan Katzenbeisser, Volkmar Lotz and Edgar R. Weippl. Vol. P-228. LNI. GI, pp. 319–329. URL: https://dl.gi.de/20.500.12116/20053.

[139] Theodore J Williams. 'The Purdue enterprise reference architecture'. In: *Computers in industry* 24.2-3 (1994), pp. 141–158.

[140] World Health Organization and others. *Hospital safety index: Guide for evaluators*. Tech. rep. 2015.

[141] C. Yang, W. Shen, Q. Chen and B. Gunay. 'A practical solution for HVAC prognostics: Failure mode and effects analysis in building maintenance'. In: *J. of Building Engineering* 15 (2018).

[142] Ken Yau, Kam-Pui Chow and Siu-Ming Yiu. 'An Incident Response Model for Industrial Control System Forensics Based on Historical Events'. In: *Critical Infrastructure Protection - 13th IFIP WG 11.10 International Conference, ICCIP, Arlington, VA, USA, March 11-12, 2019, Revised Selected Papers*. Ed. by Jason Staggs and Sujeet Shenoi. Vol. 570. IFIP Advances in Information and Communication Technology. Springer, pp. 311–328. DOI: 10.1007/978-3-030-34647-8_16. URL: https://doi.org/10.1007/978-3-030-34647-8_16.

[143] Yunhui Zheng, Saurabh Pujar, Burn L. Lewis, Luca Buratti, Edward A. Epstein, Bo Yang, Jim Laredo, Alessandro Morari and Zhong Su. 'D2A: A Dataset Built for AI-Based Vulnerability Detection Methods Using Differential Analysis'. In: *International Conference on Software Engineering: Software Engineering in Practice, ICSE (SEIP) 2021, Madrid, Spain, May 25-28, 2021*. IEEE, pp. 111–120. DOI: 10.1109/ICSE-SEIP52600.2021.00020. URL: https://doi.org/10.1109/ICSE-SEIP52600.2021.00020.

[144] Zhiyuan Zheng and A. L. Narasimha Reddy. 'Safeguarding Building Automation Networks: THE-Driven Anomaly Detector Based on Traffic Analysis'. In: *International Conference on Computer Communication and Networks, ICCCN, Vancouver, BC, Canada, July 31 - Aug. 3, 2017*. IEEE, pp. 1–11. DOI: 10.1109/ICCCN.2017.8038393. URL: https://doi.org/10.1109/ICCCN.2017.8038393.

[145] Ioannis Zografopoulos, Juan Ospina, Xiaorui Liu and Charalambos Konstantinou. 'Cyber-Physical Energy Systems Security: Threat Modeling, Risk Assessment, Resources, Metrics, and Case Studies'. In: *IEEE Access* 9 (2021), pp. 29775–29818. DOI: 10.1109/ACCESS.2021.3058403. URL: https://doi.org/10.1109/ACCESS.2021.3058403.